IEEE NCA 2016

The 15th IEEE International Symposium on Network Computing and Applications

Cambridge, MA, USA · October 31-November 2, 2016

Sponsored by:

IEEE Technical Committee on Distributed Processing
Akamai Technologies, Inc.
International Research Institute on Autonomic Network Computing

IEEE computer society

Akamai

Irianc

PROCEEDINGS

2016 IEEE 15th International Symposium on Network Computing and Applications

NCA 2015

PROCEEDINGS

2016 IEEE 15th International Symposium on Network Computing and Applications

30 October – 2 November 2016

Cambridge, MA, USA

Editors

Alessandro Pellegrini, Sapienza University of Rome, Italy

Aris Gkoulalas-Divanis, IBM Watson Health, USA

Pierangelo Di Sanzo, Sapienza University of Rome, Italy

Dimiter R. Avresky, IRIANC, USA/Germany

Sponsored by

IEEE Computer Society Technical Committee on Distributed Processing (TCDP)

Akamai Technologies, Inc.

International Research Institute on Autonomic Network Computing (IRIANC)

ISBN 978-1-5090-3215-0
ISBN 978-1-5090-3216-7

2016 IEEE 15th International Symposium on Network Computing and Applications

NCA 2016

Table of Contents

Session 2: Cloud Computing

Session 3: Network Security I

Session 4: Networking

Session 5: Software-Defined Networks

Session 6: Internet of Things

Session 7: Fault Tolerance

Session 8: Distributed Computing I and Energy Efficiency

Session 11: Performance, QoS, and Monitoring

Session 12: Network Security II

MESSAGE FROM THE STEERING COMMITTEE AND GENERAL CHAIRS

All research topics of IEEE NCA*, for more than one decade, have been contributing to the creation of the foundation of the Network/Cloud Computing research and application do domain. Now, Network/Cloud Computing is becoming the major mode of operation of the Information Technology Industry over the Internet. BIG Data, Software Defined Networks (SDN), Network Functions Virtualization (NFI), Internet of Things (IoT) have profound effect on the innovations and modern society. The challenge for Network Cloud Computing is to realize a true Internet of Things, a network capable of supporting potentially trillions of wireless connected devices and with overall bandwidth one thousand times higher that today's wireless networks. Current IT technologies are approaching their limits.

The organizers of the IEEE NCA2016 would like to recognize the support of the leading scientists in this research field. They contributed to the NCA* as distinguished chairs, keynote speakers or provided sponsorship.

We would like to mention some of the leading institutions and companies, such as Massachusetts Institute of Technology – MIT Cambridge, MA; Akamai Technologies Inc., Cambridge, MA; IBM – Thomas J. Watson Research Center, New York; Cornell University, USA; BBN Cambridge, MA, USA; Microsoft Research, CA, USA; CNRS – LAAS, Toulouse, France; University of Tennessee & Oakridge National Labs; National Science Foundation – NSF,USA; DARPA, USA; Cisco, MA, USA; Argonne National Lab – Chicago; European Research Counsel – ERC, Brussels, Belgium; University of Illinois at Urbana Champaign; International Research Institute on Autonomic Network Computing – IRIANC, Boston/ Munich Germany.

We would like to thank all TPC members, PC Chair A. Gkoulalas-Divanis, PC Co-Chair and Financial Chair A. Pellegrini, PC Co-Chair and Publication Chair P. Di Sanzo, for their voluntary hard work, which has contributed to the success of IEEE NCA2016.

Our special acknowledgments are to the IEEE Computer Society for its Sponsorship and The IEEE Computer Society Technical Committee on Distributed Processing (TCDP).

D. R. Avresky
Co-Founder NCA* and Steering Committee Chair
President of International Research Institute on
Autonomic Network Computing (IRIANC)
Boston, USA / Munich, Germany
autonomic@irianc.com

Mladen A. Vouk
General Chair NCA 2016
NC State University, USA
http://renoir.csc.ncsu.edu/Faculty/Vouk

MESSAGE FROM THE PROGRAM CHAIRS

Dear friends, Dear colleagues, Dear participants,

It is with great pleasure that we welcome you to the 15th edition of IEEE NCA. Over the years, NCA has become a successful series of conferences that serves as a large international forum for presenting and sharing recent research results and technological developments in the fields of Network and Cloud Computing. This edition of NCA features a lively, interesting, and stimulating program with a lot of opportunities for discussing new results, on-going projects, and the future trend in our fields.

As in the previous editions, the conference is organized in thematic sessions that reflect the diversity of topics. This year, we received a total of 105 submissions, and accepted 29 as full papers, which is a selective rate of acceptance of less than 28%. Additionally, 36 contributions have been accepted as short papers, to allow for a more dynamic and interactive event.

The Technical Program Committee, consisting of 36 distinguished experts, made this selection, closely aided by an External Review Committee of 28 additional experts, active in our fields. We thank from our hearts the Technical Program Committee and the external reviewers for their invaluable effort in this process, which made possible this year's excellent program. Special thanks are also due to the NCA Steering Committee Chair Dimiter Avresky for his support in the organization of this event.

The high quality of this edition is also attested by two distinguished papers, winner of the best paper award (*Cost-effective Processing for Delay-sensitive Applications in Cloud of Things Systems* by Yucen Nan, Wei Li, Wei Bao, Flavia C. Delicato, Paulo F. Pires, and Albert Y. Zomaya), and the best student paper award (*Analysis of the Propagation Time of a Rumour in Large-scale Distributed Systems* by Yves Mocquard, Samantha Robert, Bruno Sericola, and Emmanuelle Anceaume). We are also pleased to host a keynote from Tim Strayer on "*Evolving Systems for Situational Awareness*", which is expected to interest everyone attending the conference. The topics of this year's edition are well reflecting the needs of IT technologies. Some of them include: cloud computing, autonomic management, machine learning, SDN, NVF, security, privacy, and virtualization.

The success of a conference is mainly on the side of the participants, we therefore warmly thank all the contributors and participants. The team of chair persons did a huge work – despite all the difficulties and hindrances faced during the process – which finally led to a very attractive program. We wish you a successful conference, and a pleasant stay in the beautiful Boston!

Aris Gkoulalas-Divanis
IBM Watson Health, Cambridge, MA, USA
Program Chair

Alessandro Pellegrini, Sapienza University of Rome, Italy
Pierangelo Di Sanzo, Sapienza University of Rome, Italy
Program Co-Chairs

Conference Committee

Steering Committee Chair
Dimiter R. Avresky, IRIANC, Boston, USA / Munich, Germany

General Chair
Mladen A. Vouk, NC State University, USA

Program Chair
Aris Gkoulalas-Divanis, IBM Watson Health, Cambridge, MA, USA

Program Co-Chairs
Alessandro Pellegrini, Sapienza University of Rome, Italy
Pierangelo Di Sanzo, Sapienza University of Rome, Italy

Financial Chair
Alessandro Pellegrini, Sapienza University of Rome, Italy

Publication Chair
Pierangelo Di Sanzo, Sapienza University of Rome, Italy

Technical Program Commitee

Emmanuelle Anceaume, CNRS / IRISA, France

Spiros Antonatos, IBM Research, Dublin, Ireland

Luciana Arantes, Universite Pierre et Marie Curie-Paris6, France

Dimiter R. Avresky, IRIANC, Boston, USA / Munich, Germany

Arndt Bode, Technische Universität München, Germany

Yann Busnel, Crest (Ensai) / Inria, France

Bruno Ciciani, Sapienza, University of Rome, Italy

Gene Cooperman, Northeastern University, MA, USA

Michel Cukier, University of Maryland, MD, USA

Peter Desnoyers, Northeastern University, MA, USA

Pierangelo Di Sanzo, Sapienza University of Rome, Italy

Aris Gkoulalas-Divanis, IBM Watson Health, Cambridge, MA, USA

Alfredo Goldman, University of São Paulo, Brazil

Doan Hoang, University of Technology, Sydney, Australia

Barry W. Johnson, University of Virginia, VA, USA

Zbigniew Kalbarczyk, Coordinated Science Laboratory, IL, USA

Kostas Katrinis, IBM Dublin Technology Campus, Ireland

Roger Khazan, MIT Lincoln Laboratory, MA, USA

Ioannis Konstantinou, National Technical University of Athens, Greece

Nectarios Koziris, National Technical University of Athens, Greece

Michael Liu, the Chinese University of Hong Kong

Erik Maehle, University of Lübeck, Germany

Grzegorz Malewicz, Facebook, USA

Antonio Manzalini, Telecom Italia Lab, Italy

Philippe Massonet, CETIC, Belgium

Rasmus L. Olsen, Aalborg University, Denmark

Philippe Owezarski, LAAS-CNRS, France

Roberto Palmieri, Virginia Tech, VA, USA

Alessandro Pellegrini, Sapienza University of Roma, Italy

Francesco Quaglia, Sapienza, University of Rome, Italy

Hans-Peter Schwefel, Aalborg University, Denmark

Pierre Sens, LIP6 - University of Paris 6 - CNRS – INRIA, France

Jacek Serafiński, OtherLevels

Wilfried Steiner, TTTech Computertechnik AG, Austria

Mladen A. Vouk, NC State University, NC, USA

Gabriel Wainer, Carleton University, ON, Canada

External Reviewers

Andrea Ceccarelli, Università degli studi di Firenze, Italy

Davide Cingolani, Sapienza University of Rome, Italy

Daniel Cordeiro, University of São Paulo, Brazil

Katerina Doka, National Technical University of Athens, Greece

Simone Economo, Sapienza University of Rome, Italy

Athena Elafrou, National Technical University of Athens, Greece

Rene Gabner, FTW Telecommunications, Austria

Ioannis Giannakopoulos, National Technical University of Athens, Greece

Jan Haase, Universität zu Lübeck, Germany

Pinjia He, The Chinese University of Hong Kong

Michel Hurfin, Inria Rennes, France

Mauro Ianni, Sapienza University of Rome, Italy

Evie Kassela, National Technical University of Athens, Greece

Mohammed Kemal, Aalborg Universitet, Denmark

Jian Li, The Chinese University of Hong Kong

Romaric Ludinard, Crest (Ensai) / Inria Rennes, France

Tatiana Madsen, Aalborg Universitet, Denmark

Romolo Marotta, Sapienza University of Rome, Italy

Nikela Papadopoulou, National Technical University of Athens, Greece

Thomas Paulin, Aalborg Universitet, Denmark

Nuno Pratas, Aalborg Universitet, Denmark

Nicolo Rivetti, Sapienza University of Rome, Italy – Crest (Ensai)/Inria, France

Han Shao, The Chinese University of Hong Kong

Dimitrios Siakavaras, National Technical University of Athens, Greece

Hui Xu, The Chinese University of Hong Kong

Xiaotian Yu, The Chinese University of Hong Kong

Jichuan Zeng, The Chinese University of Hong Kong

Shenglin Zhao, The Chinese University of Hong Kong

KEYNOTE

Evolving Systems for Situational Awareness

Dr. Tim Strayer
Raytheon BBN Technologies
Cambridge, MA, USA

Abstract. The most basic functions of any organism or organized unit requires understanding the environment. This is accomplished through sensing; for organisms, sensing is an evolved faculty. The same is true for military and first responder units such as squads, who are relying on more sophisticated sensing systems to gain greater situational awareness (SA) in order to more effectively and safely conduct their missions. Adding sensing to the squad, however, only solves part of the problem; there must be an appropriate communications system in place to make full use of the acquired SA information. This talk will explore the use of content-based networking solutions for efficiently distributing the SA information in a decentralized manner.

Bio. Tim Strayer is a Principal Scientist in Raytheon BBN Technologies' Network and Communications group, where he leads several Government-funded projects focused on applying content-based networking techniques to soldier communications at the tactical edge. He joined BBN in 1997 from Sandia National Laboratories (California), where he was developing novel transport protocols. At BBN, he has worked on many DARPA and industry sponsored projects in the areas of Active Networking, satellite packet switching, mobile IP, virtual private networks, and routing systems, as well as many projects on attack tracing and botnet detection. He received his PhD from the University of Virginia in 1992. He has written over 30 journal and conference papers, 14 patents, several book chapters, and two Addison-Wesley books.

Crowdsourcing-based architecture for post-disaster Geolocation: a comparative performance evaluation

Florent Coriat*†, Anne Fladenmuller*†, Luciana Arantes*†,‡ and Olivier Marin§
* Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
E-mail: [florent.coriat, anne.fladenmuller, luciana.arantes]@lip6.fr, ogm2@nyu.edu
† CNRS, UMR 7606, LIP6, F-75005, Paris, France
‡ Inria, REGAL project-team, Paris-Rocquencourt, France
§ New York University Shanghai (NYU Shanghai), Shanghai, China

***Abstract*—In the aftermath of a natural or industrial disaster, locating individuals is crucial. However, disasters can cause extensive damage to the network infrastructures and a generalized loss of communication among survivors. In this article, we present a network support solution that provides a post-disaster geolocation-collecting service that relies on inter mobile device connections. On top of this dynamically built network, survivors' mobile devices exchange information about geolocation of others they have encountered. Such information is routed towards predefined data collection centers using either the DTN Epidemic or Spray and Wait DTN protocol. Experiments were conducted on the ONE simulator and performance evaluation results confirm the effectiveness of our proposal.**

***Index Terms*—geolocation, DTN, early post-disaster management, mobile devices**

I. Introduction

After a natural or human-made disaster, regular communication infrastructure is often damaged and/or overloaded by witnesses, survivors, or their relatives who keep repeatedly trying to get information about the current situation, in order to help or simply to reassure themselves. Yet global information assessment is a crucial issue for rescuers. Thus, establishing, as soon as possible, a dedicated communication network, until the regular network works normally, remains a top priority and a great challenge [1]. Ideally, such a communication network should also provide support for real-time mapping of the disaster area, danger zones and resources, as well as geolocation information of victims and survivors. In other words, the support must help to assess and deal with the emergency situation, mapping accident areas, locating users, warning users about the accidents, and maintaining contact between survivors, rescue teams, and emergency operation centers.

Many existing solutions deploy emergency network infrastructures (e.g. mobile cell sites, balloons, etc.). However, these solutions are often expensive, can take several hours after the disaster to be operational, depend on dedicated equipments, or are frequently restricted to rescue teams.

Hence, considering the above constraints, a feasible, relatively cheap, and easily available solution would be to build an ad-hoc network composed of people's (victims and survivors) mobile devices, i.e., to exploit existing mobile devices without counting on additional dedicated network infrastructure. Since users' mobile devices usually have wireless interfaces (Wi-Fi and/or BlueTooth), it would be possible to establish, maybe intermittently, ad-hoc communication links between mobile nodes, similarly to MANET (mobile ad-hoc network) or DTN (disrupted-tolerant network).

In this paper, we present a post-disaster network support solution that exploits such an ad-hoc dynamically built communication approach, requiring a minimum of fixed support. We consider that dedicated software for disaster situations runs on peoples' mobile devices as well as the existence of well-known places, denoted convergence points *CPs* (e.g., hospitals, railway stations, schools, etc.). Some of these CPs are also collecting data centers, denoted *hotspots*. The latter are endowed with processing and storage resources. Seeking to be in safe area when a disaster takes place, people move towards one of the *CPs*, establishing the ad-hoc communication links between his/her mobile device and those of the persons he/she crosses along the path to the *CP*. Furthermore, whenever a communication link exists between two persons' devices, they can exchange information about geolocation of other persons they have met. Therefore, users mobility contributes to both build an ad-hoc network and propagate information about users location. Through such a network, users can also be informed about accidents which might help them to find a better path to the most convenient *CP*.

Since a network composed of persons' mobile devices is intermittently connected, i.e. connections between them and the hotspots are built over time, similarly to DTNs, we applied (and adapted) two well-known DTN routing protocols, *Epidemic* [2] and *Spray and Wait* [3], for routing geolocation data towards the hotspots. Based on such disaster context, we also propose a mobility model, denoted *Danger Movement*. Extensive experiments were conducted with both protocols on *Danger Movement*, varying different parameters, on top of the ONE[1] simulator [4]. Performance evaluation results confirm the effectiveness and feasibility of our approach.

The paper is organized as follows: Section II gives a brief background about DTNs and the protocols *Epidemic* and *Spray and Wait*. In Section III, we summarizes some existing related

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

[1]The Opportunistic Networking Environment

work. Our solution is described in Section IV which also includes the description of the *Danger Movement*. Section V presents and discuss some evaluation performance results with both protocols. Finally, Section VI concludes the paper and gives some directions for future work.

II. Disruption-tolerant Networking

Base radio stations are statically placed aiming at offering network coverage to users. However, if they have been damaged by a disaster or the latter caused a power failure, their re-establishment or the deployment of mobile cell sites to complete or replace the failing infrastructure can take from hours to several days, wasting valuable time for victims and rescuers. Thus, autonomous and fast operational solutions to offer some communication support in the first moments of a disaster are crucial in such a context. To this end, disruption-tolerant networking (DTN) technology has been largely considered for post-disaster communications.

A disruption-tolerant network (DTN) is an opportunistic network where each node acts as a router for the network packets. Nodes can be pedestrians' mobile devices, on-board vehicle specialized devices, fixed relays, etc. In [5], the authors present a classification of these types of nodes, their mobility patterns, and communication abilities (terminals, stationary relays, dedicated fixed or mobile routing facilities, etc.) from the perspective of post-disaster communication and coverage.

Devices like mobile cell stations, satellite relays, etc. can be components of a DTN. However, the main purpose of DTNs is to exploit non specialized, not expensive devices, usually with limited resources, but which are fast deployed and, thus, very suitable for disaster context.

DTN routing: Many protocols have been developed for DTNs and evaluated on different network architectures. "Simple" protocols like First Contact, Epidemic, or Spray and Wait do not rely on any assumption or network structure measures for their routing rules. They are often used as references in comparative studies, regardless of the study context.

Epidemic [2] is one of the most simple DTN routing protocols: whenever two nodes meet, they compare their respective sent and received message histories and exchange their "new" messages. The advantage of the *Epidemic* protocol is its high delivery probability. On the other hand, it consumes a lot of storage, power, and bandwidth resources. To circumvent such a problem, a FIFO policy is usually used to discard messages when the node's storage buffer is full. Hence, dissemination is limited by both bandwidth and storage and, in the long term, by available remaining energy.

Spray and Wait (SnW) [3] is a simple trade-off between replication-based routing protocols, like *Epidemic*, and forward-based routing ones. The source of a message replicates it in L copies. The latter are firstly disseminated to $L-1$ other nodes. This *spray phase* accepts some variants. For instance, in *Binary SnW*, any node that has $n > 1$ message copies (source or relay), and encounters a second node (with no copies), gives to it $\lfloor \frac{n}{2} \rfloor$ and keeps $\lceil \frac{n}{2} \rceil$ for itself. *Binary SnW* has a lower delivery delay than the original *Spray and Wait*. Whenever a node has only one copy left, it switches to *wait phase*: it stores the message until it crosses the destination node to directly transmit the message. In other words, the *Spray and Wait* protocol manages the number of copies messages in the spray phase and uses Direct Delivery in the wait phase. As a result, the protocol presents fewer message transmissions than the *Epidemic* protocol.

Other protocols more "complex" than the previous ones, such as Spray and Focus [6], PRoPHET [7], MaxProp [8], rely on history of nodes' crossing or specific characteristics of a mobility model (Time To Return) which predicts future nodes meeting probabilities. These protocols assume a somewhat redundant contact approach. In particular, a node S can efficiently route a message to a node D if and only if S *knows* D, i.e., if a packet (whichever it is a data or protocol packet) already followed a path from D to S, provided that this path still exists or at least part of it. However, these protocols perform at best as efficiently as the "simple" ones when this condition is not met. As we shall see later, geolocation messages of our protocols mainly follow such unknown paths.

We should also point out that DTN protocols are usually evaluated and compared under various scenarios and/or large randomly-generated data transfers. On the contrary, our scenarios rely on small geolocation messages, whose generation depends on each specific scenario.

III. Related Work

In the context of disasters, we summarize in this section some existing solutions that provide support for diffusion of information and/or communication coverage as well as some mobility models proposed in some works.

A. Existing Solution for Support

Rescue workers typically use a dedicated trunked network infrastructure, based on a specific protocol like P25 or TETRA [9]. These protocols provide support for encrypted voice and data transmission throughout a fixed mesh infrastructure, or using direct (point-to-point) communications. However, these networks are restricted to licensed professionals, and their design would not scale to a large use. Furthermore, P25 and TETRA do not provide multi-hop routing when using point-to-point communications, thus preventing their use for data collection without a resilient infrastructure.

Some hardware solutions have also been proposed with the goal of temporarily replacing or restoring part of the damaged infrastructure, In this case, mobile cell sites (e.g.,"Cell On Wheels", "Cell On Truck", etc.), can be deployed to address emergencies and usually connected through wired connections, parabolic antennas, a satellite network, or a network of helium-inflated balloons, as proposed by the Loon project [10] for restoring access to the internet. Nevertheless, these solutions remain expensive — tens of thousands of dollars per cell — and their post-disaster deployment requires time.

In recent years, different disasters led to the emergence of many projects whose goal is to inform people about

the disaster and to include them in the situation assessment process. For instance, after the earthquake in Japan in 2011, more than 150 applications were developed [11] to face the disaster consequences. Most of these applications rely on crowdsourcing, offering information about the current situation, risks, needs, resources, people locations, etc. to victims, rescuers, and/or authorities. Ushahidi [12], Sahana Eden [13], Google Crisis Response [14], UbAlert Disaster Alert Network [15] or People Locator [16] are examples of projects that make available collaborative maps on top of which the above information can be added. These applications consist of web sites or mobile applications. Even if some of them exploit alternative communications channels, like Ushahidi, which is able to collect information by SMS/MMS, they all rely on regular infrastructures as communication support.

Aiming at tolerating network failures and disruption, some other projects provide solutions that decentralize communication. Twimight [17] ("Twitter in disaster mode"), is a Twitter client that can work without internet connection, exchanging tweets in an opportunistic way between terminals. Firechat [18] is a messaging application that can communicate with or without internet connection: Wi-Fi and BlueTooth interfaces are responsible for building a mesh network where each message can be stored, carried and forwarded on any available link. The Serval project [19] developed a yet experimental messaging application that can transmit all sorts of data (messages, maps, voice, pictures, etc.) over an ad-hoc mesh network. These projects are not specially tailored to tackle with disaster situations but they are general solutions for regions or situations where network coverage is not working properly.

B. Mobility Models

The use of MANET or DTN to deal with a crisis situation has already been proposed [20], [21]. Several studies have compared known protocols in a crisis scenario [5], [22]–[26], or the in everyday life [7], [27], [28]), and have shown the impact of mobility on system performance.

Despite its lack of realism [29], RWP is often used for simple [26] or non-crisis-specific [30] evaluations or even as a reference or a sub-model of a more specific model. In [5], the authors compare a set of DTN routing protocols in the scenario of Uttarakhand floods (India, 2013, 4 days of intense rainfall, 4200 affected villages, 5700 deaths). The impact of RWP, Map Based [4], Cluster Movement and Post Disaster Mobility (PDM) [31] mobility models is evaluated on top of the ONE simulator. Among those, the last two ones also use RWP and Map Based as sub-models. For instance, PDM assigns different roles to nodes, with different mobility patterns: patrol, exploration, round-trip. Some of these patterns are based on RWP. PDM is also used in the experiment scenario in [32], which addresses mobility prediction in a crisis situation.

The Disaster Area (DA) model is used in [25] and [33] to evaluate DTN routing or broadcast protocols. The DA model introduces the concept of zones which are deployed by rescuers. Pedestrians follow a zone-restrained RWP mobility model with obstacles, whereas vehicles go back and forth between two zones.

There exist several other models in the literature which were conceived for disaster situations such as CORPS [34], Dispatched Ambulance [35], Reference Point Group Mobility (RPGM) [36], Human Behaviour for Disaster Areas (HBDA) [37], etc. However, all these models focus on rescue teams mobility, neglecting the movement of other people present in the disaster scenario [38]. In other words, victims and survivors are poorly represented, if not simply ignored, or follow a simplistic generic model (e.g., RWP or similar).

IV. Our Proposed Post-disaster System

In this section, we present our post-disaster system that aims at gathering geolocation and mapping information in collecting centers (*hotspots*) as well as informing users about accident locations. Moving to convergence points *CPs*, which can also be a hotspot, persons' mobile devices that get into contact with others, exchange information about geolocation of other persons they have met. Users mobility contributes then to both build a connected network and propagate information.

Our solution is deployed as a specific application, installed on people's mobile devices beforehand. We denote *holder* a person that holds a mobile device running this application and is within the disaster area. Note that a *holder* can be also a motionless victim, such as injured or dead victims as well as active dropped mobile devices.

Our goal is to provide a post-disaster geolocation system with the following features:

- *Public availability:* the network built by holders' devices must be available and usable by everyone that has a mobile device (smartphones, tablets, etc.).
- *Fast operational capacity / ad hoc routing:* exploiting the holders' mobile devices, the network must be fully operational without considering, except hotspots, any additional fixed support. In other words, no specific devices and support should be used for routing messages, which would, then, require the assumption that holders' devices handle them.
- *Collection of data:* data should be routed to central points (hotspots).
- *Freshness of information:* collected data mainly consist of geolocation of persons' mobile devices in the area. However, only recent information about their respective location should be considered, which means that those messages with old information must be discarded.
- *Informing users about accident locations*: holders should, as much as possible, be informed about where there are accidents in order to allow them to change their path and circumvent risking zones.

In the following, we firstly describe the environment and assumptions that we consider for our system. Then, since the majority of existing mobility models of the literature concern rescue teams (see Section III), we propose a new mobility model, denoted *Danger Movement*, that takes into account the considered environment and the movement of persons towards

CPs. Finally, in this context, we propose to use (and adapt) both *Epidemic* and *Binary Spray and Wait* DTN protocols in order to route persons information to hotspots and disseminate accidents geolocation information.

A. Environment and Assumptions

We consider that there exist fixed places, denoted convergence points *CPs*, like hospitals, railway stations, schools, etc., which are known by all people and towards which they move when a disaster takes place. By default, every user chooses the closest *CP* in relation to his/her position when aware of the disaster. Nonetheless, a person can choose (with a certain probability) to reach another one, randomly chosen. Such an option improves the realism of the model, enriching it with "human" behaviors as, for example, parents fetching their children from school (a *CP*) instead of directly reaching the nearest rescue center.

Holders trying to reach a *CP* can be blocked by accidents. An *accident* is an event that permanently blocks an intersection of roads. Accidents are triggered on randomly chosen intersections following an exponential law with a chosen mean time between two accidents (MTBA).

Some *CPs* are endowed with storage, processing, and energy resources which render them collecting centers, denoted *hotspots*. The latter are interconnected by long-range resilient wireless links (laser, parabolic antenna, satellite, etc.). Periodically, hotspots synchronize themselves, exchanging collected data. Notice that this resilient communication infrastructure is not contradictory with our aim of providing a post-disaster solution which does not rely on a dedicated infrastructure. In our case, the latter is only used for synchronization between hotspots and not for connecting people or routing geolocation information to hotspots. If such a hotspot synchronization does not take place, the system still works with each hotspot having a partial view of people geolocations.

We also assume that both holder's devices and hotspots have Wi-Fi and/or Bluetooth wireless interfaces, which are widely available among the devices of holders nowadays. While, Wi-Fi benefits from better signal ranges than BlueTooth, BlueTooth consumes significantly less energy than Wi-Fi. Each of these interfaces runs a DTN router. We consider that a unique constant identifier (e.g., phone number) is assigned to each holder's device which is also equipped with a global positioning system (GPS).

B. Mobility Model: Danger Movement

Danger Movement is a map-based mobility model which takes into account the context described in the previous section. Basically, it characterizes the movements of survivors who move towards *CPs*, exchanging information about people and accident locations, which are routed to hotspots. A preliminary version of the model was presented in [39].

In accordance with our assumptions, hotspots are placed on the map as static nodes. Initially, *Danger Movement* randomly places holders over the map. The latter can then walk over the map at fixed or random (bounded) speeds.

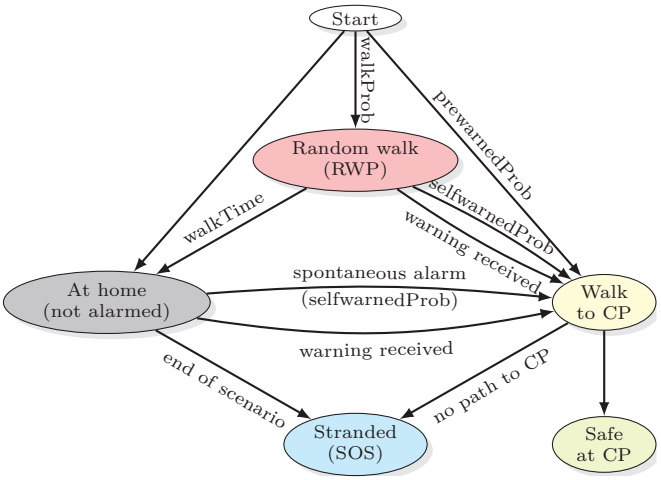


Fig. 1. Danger Movement : state (sub-models) diagram of a holder

Holders running *Danger Movement* may be aware or not of the crisis situation. We denote *alarmed* a holder aware of the situation. Four cases are possible for a holder in *Danger Movement*: (1) he/she is *alarmed* from the beginning; (2) he/she spontaneously realizes the situation during simulation; (3) the holder is warned by another *alarmed* holder; (4) the holder is never *alarmed*.

The map is known by all holders. Therefore, they are always able to find the shortest *off-disaster* path to any *CP*. However, holders have only knowledge of an accident provided they have seen it or another holder informed them of it. Upon having knowledge of a new accident, a holder may recompute his/her current path. Note that an alarmed holder may become *stranded* if there are no more accessible paths left towards his/her target *CP*.

Once arrived to the target *CP*, holders stop moving and, after transmitting all their data to the hotspot (if the *CP* is a hotspot), they disable their interfaces.

Figure 1 describes the different states of a holder, depending on its awareness and whether he/she is at home, walking outside or "stranded" by accidents. Notice that, in order to ensure that simulations based on *Danger Movement* finish, it is not required that all holders reach the safe state. Once a given defined percentage of holders are safe, all remaining holders become stranded ones.

The probabilities $prewarnedProb$ and $walkProb$ respectively concern the probability that a holder is *alarmed* from the beginning of the simulation (case 1) and that he/she starts walking without being *alarmed* (case 3). The $selfwarnedProb$ is the probability, every $1/10\,\mathrm{s}$, for a non-alarmed holder to become spontaneously *alarmed* (case 2).

C. Protocols and Messages

As explained, every holder's interface runs a DTN router. However, since all geolocation data should be directed to hotspots, applying "complex" DTN protocols would be unfeasible in such a disaster context. Furthermore, the size of the messages is very small and those with old information should

be dropped. Therefore, we have chosen to apply and adapt the simple and well-known DTN protocols *Epidemic* and *Spray and Wait* (see Section II).

Three message types are used by the protocols:

- *geolocation*: contains the GPS coordinates of a holder;
- *accident*: the GPS coordinates of an accident;
- *warning*: a special message which renders *alarmed* those holders who received the message.

Since every holder has a GPS receiver, it is able to build a *geolocation message* at any time.

Warning messages are automatically created by already *alarmed* holders while *accident* messages are generated by holders when getting knowledge about an accident. When they cross each other, holders may exchange *geolocation* information about the holders and accidents of which they are aware. Each of this information consists of a single message whose size does not exceed $20\,\mathrm{B}$. However, MANET/DTN routing protocols are usually evaluated with larger messages: from $64\,\mathrm{B}$ for MANETs to $100\,\mathrm{KiB}$ for DTNs. Thus, aiming at preventing holders from sending numerous small messages, which would induce high network overhead, messages are grouped into frames of a predefined maximum transmission unit (MTU) size before being transmitted.

In order to satisfy the *freshness of information* requirement, messages about persons' geolocation must be versioned, i.e., timestamped. In this way, only the most recent ones (last version) are collected and kept by holders and hotspots. A status can also be added to each *geolocation* message, giving more information about the holder (not injured, injured, motionless victim, etc.). Our solution should, thus, ensure a anycast routing service for versioned messages, dropping those with old version.

As argued, we have chosen to implement both the Epidemic and Spray and Wait DTN protocols (see Section V), adapting them to the described disaster environment. In the case of Spray and Wait, we adapted the Binary SnW, where every node (holder), keeping more than one copy of a message, transmits half of them on the next node (holder) contact.

The two protocols have been modified to deal with messages versions: a message is automatically dropped as soon as a more recent version is received. In the case of Spray and Wait protocol, this message dropping may concern several copies of the same message, independently of the number of copies received for the new version of the message. Lastly, a transmission delay is applied to more recent versions of an already sent message, preventing continuous update flow transmissions.

It is worth noting that *warning* and *accident* messages have to be disseminated among all holders, i.e., in an epidemic way, so that only geolocation data are affected by the protocol choice.

Collected data are forwarded to hotspots, which periodically synchronize themselve, keeping only last versioned data.

TABLE I
SIMULATION PARAMETERS – REFERENCE SCENARIO

Map	Santiago Center, $29\,\mathrm{km}^2$
Simulated time	10000s
Walk speed	constant : $1.3\,\mathrm{m\,s^{-1}}$ ($\sim 4.7\,\mathrm{km\,h^{-1}}$)
Accidents	10, MTBA : $500\,\mathrm{s}$
Number of CPs	3 | 5 | 20
of which hotspots	3
Survivors / victims	$1800/200$
walkProb	.14
preWarnedProb	.8
selfWarnedProb	10^{-6} at each $1/10\,\mathrm{s}$-step
CP choice	the nearest one
Interface (range)	Wi-Fi ($100\,\mathrm{m}$) | BlueTooth ($10\,\mathrm{m}$)
Bitrate	$250\,\mathrm{ko\,s^{-1}}$
MTU	$1500\,\mathrm{bytes}$
Retransmission delay	$300\,\mathrm{s}$
Protocol	Epidemic | Spray and Wait
Param. Spray and Wait	binary, 8 copies

V. PERFORMANCE EVALUATION

Experiments were conducted on the ONE simulator [4] using a map of the center of Santiago ($29km^2$). CPs are placed on real amenities, with hotspots chosen among hospitals. Both protocols, Epidemic and Spray and Wait (Binary SnW) have been evaluated. Firstly, we explain and define the metrics used to evaluate these protocols. Then we describe the parameters of the simulation testbed. Finally, some results are presented and discussed.

A. Metrics

In order to estimate the ability/efficiency of routing protocols to deliver messages, performance metrics such as packet delivery fraction (PDF), throughput or end-to-end delay [22] are commonly used, while protocol overhead is evaluated through metrics of normalized routing load (NRL) [25] or energy cost per message.

However, in a disaster scenario, it is important to deliver only up-to-date information, discarding obsolete messages before they can reach their final destination (a hotspot). Hence, usual performance metrics (e.g., throughput, PDF, NRL) become irrelevant or very difficult to evaluate. In our context, we aim at gathering as much geolocation information as possible while keeping low energy consumption. Consequently, to compare the efficiency of different routing protocols, we propose a new metric that better reflects the quantity of the accurate geolocation information gathered on hotspots. This efficiency metric is denote *fraction of discovered holders*, known by at least one hotspot. The overhead entails by the protocols is estimated by measuring the *average number of frames sent* by a holder over the time. Note that this second metric gives an overview of the global energy consumption of our protocol.

B. Simulation Testbed

For each parameters set, 10 simulation runs were executed. The curves of the figures show the average values for these runs.

Table I summarizes the parameters for our reference scenario. Note that the number of convergence points can vary

between scenarios (3, 5, or 20), but the number of hotspots is always fixed to 3.

C. Evaluation Results

We have conducted our simulations on different scenarios. The first one is used as a reference to evaluate the impact of the type of interfaces, routing protocols, and number of *CP*s on the efficiency and energy consumption metrics.

Reference scenario: Figure 2 and the first columns of Table II show results of the simulations, considering the parameter set of Table I.

Firstly, we observe that all simulations stabilize within 90 minutes or less and none of them ends with a complete knowledge of the position of all individuals: the best simulation results yield a knowledge of $\sim 99.5\,\%$ of the holders' positions, leaving only a dozen of holders unreachable.

Efficiency performance: Since only 3 CPs are hotspots which collect geolocation information, in scenarios with more CPs, device holders are statistically less likely to reach a hotspot. Consequently, performance in terms of number of discovered holders clearly drops as the number of CPs increases.

As expected, Wi-Fi interfaces produce better results than BlueTooth ones. The BlueTooth's best performance (BlueTooth / Epidemic) results stand out $\sim 5\,\%$ below Wi-Fi worst case (Wi-Fi / SnW). Since Wi-Fi propagation range is broader, Wi-Fi devices have a higher chance to forward their positions to a holder moving towards a hotspot. Hence, for similar scenarios (same number of CPs), Wi-Fi outperforms Bluetooth, with the difference on the number of detected holders growing with the number of CPs. The gap is of 5% when considering 3 CPs and up to 15% with 20 CPs.

Epidemic and SnW perform similarly on BlueTooth, but Epidemic gives slightly better results on Wi-Fi. The reason is that Epidemic seems to be more resilient to the dispersion of holders when the number of CPs increases: whereas both of them acquire $\sim 99.5\,\%$ knowledge about holder positions with 3 CPs, SnW misses $\sim 5\,\%$ more holders position information than Epidemic with 5 CPs, and $\sim 10\,\%$ with 20 CPs. Such degradation is easily explained by the bound in the number of copies of SnW: some relevant holder relays are *missed* by the protocol since there is no more copy left upon the encounter moment.

We also observe that Epidemic discovery convergence time takes place $10 \sim 15\,\mathrm{min}$ earlier than SnW one, regardless of the number of CPs, probably because SnW messages are stuck in the *wait* phase of the protocol until their respective holders reach a hotspot.

Energy consumption: Table II shows that BlueTooth clearly stands out with less than 30 frames sent per holder on average, regardless of the protocol: forwarding is limited by interfaces propagation range more than by the choice of the protocol.

However, with Wi-Fi broader propagation range, energy consumption becomes more significant: around 20 times more frames than BlueTooth (note the difference in scales between the 2 figures). It also impacts the choice of the protocol since SnW outperforms Epidemic by a factor of about three to five. Thus, SnW significantly reduces energy consumption when deploying Wi-Fi interfaces while keeping relatively acceptable decrease in performance (especially with few non-hotspots *CPs*). We could, therefore, consider SnW as a the best option when deploying with Wi-Fi interfaces.

Given the low consumption of Epidemic, choosing SnW seems rather useless With BlueTooth: both protocols can be used interchangeably, with slightly better results for Epidemic.

From the above discussions of the results, we can say that Wi-Fi / SnW and BlueTooth / Epidemic are interesting tradeoffs, which will, therefore, be taken as reference in the rest of this performance evaluation study.

Intermittent interfaces: In a crisis context, where energy networks may be inoperative, holders who cooperate to define a cartography of the crisis zone must take care about the consumption of their battery-powered devices. In this context, energy aware transmissions which rely on disabling the radio interface for a certain duration of time, seems quite suitable. Hence, it is important to study the impact of intermittent radio interfaces on the propagation of our geolocation information. To this end, we have defined an environment with *intermittent interfaces*: each interface is powered up and shut down periodically. On the other hand, as hotspots are not concerned by energy issues, they always keep their radio interface active.

Since synchronization to get all wireless interfaces active simultaneously is not realistic, we consider that holder devices operate independently from one another. Thus, for the current simulations, we have chosen an intermittency functioning which is based on one of the discovery protocols of Wi-Fi Direct[2].

To circumvent the risk of having devices which are never active at the same time, the intermittency has been implemented as a *pseudo-periodic* process: interfaces are up for $active = 30\,\mathrm{s}$, then down for a random amount of time between $minInactive = 30\,\mathrm{s}$ and $maxInactive = 90\,\mathrm{s}$, and so on. In this way, two holder's devices whose active time are originally mutually exclusive will eventually be able to communicate with each other.

Figure 3 and the second section of Table II present the intermittency results. With a Wi-Fi interface running SnW, intermittency has nearly no impact on the performance (number of discovered devices), whereas the average number of messages is reduced by $\sim 25\,\%$. Conversely, Bluetooth performance degrades due to these interruptions. The number of messages is virtually unchanged but performance efficiency drops. For example, with 3 hotspots, the number of missed holders doubles from $5\,\%$ to $10\,\%$.

Overall, we can conclude from the results that performance of both protocols are similar, though slightly reduced, and that intermittency might be useful with Wi-Fi.

[2]a Wi-Fi standard enabling devices to connect with each other without requiring an infrastructure.

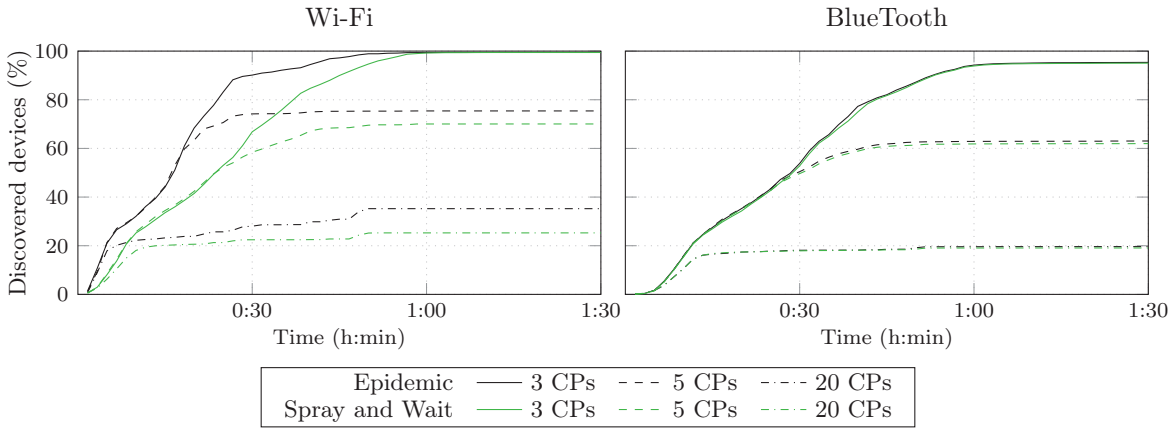


Fig. 2. Reference scenario: Epidemic vs. Spray and Wait – proportion of discovered devices

TABLE II
AVERAGE NUMBER OF SENT MESSAGES PER HOLDER

	Reference				Intermittent		V. Speed		Random CP	
	Wi-Fi (WF)		BlueTooth (BT)		WF	BT	WF	BT	WF	BT
# of CPs	Epi	SnW	Epi	SnW	SnW	Epi	SnW	Epi	SnW	Epi
3	684.4	126.0	26.0	15.8	90.7	13.6	139.3	86.1	120.9	59.7
5	150.6	93.7	13.4	12.1	67.8	8.2	102.4	43.1	96.9	71.3
20	123.3	46.2	6.3	5.9	29.1	3.6	47.9	11.9	58.3	48.2

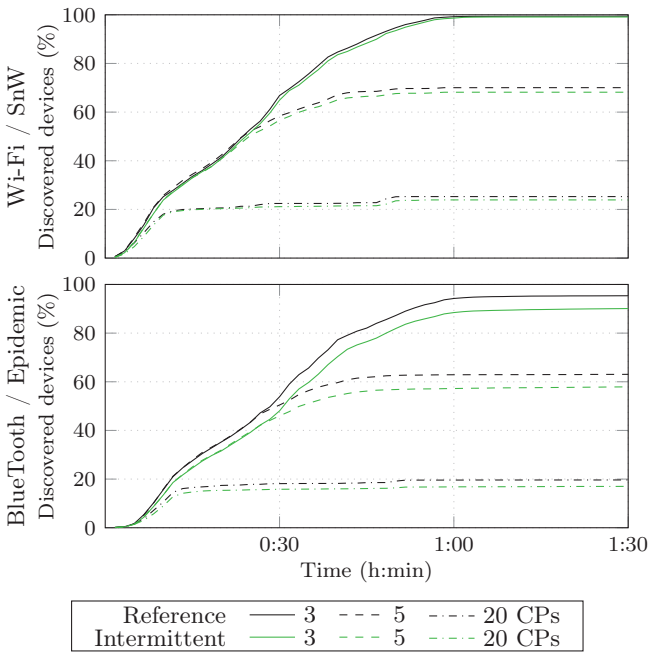


Fig. 3. Intermittent interfaces

Mobility - variable speeds, CP selection: Based on the reference scenario, we change, for the current simulation, the value of some parameters in order to evaluate their influence on the overall system performances. The first experiments concern the walking speed of holders. In our reference scenario, all mobile holders walk with a constant speed of $1.3\,\mathrm{m\,s^{-1}}$. Only victims or individuals unaware of the crisis remain static. On the other hand, speed variations may have an influence on contact occurrences and durations.

Thus, for the simulations, we considered the parameter values of Table I except for the walk speed, that randomly varies from $0.7\,\mathrm{m\,s^{-1}}$ to $2.0\,\mathrm{m\,s^{-1}}$. Figure 4 shows the results. Except for a hardly significant improvement of convergence time with BlueTooth / Epidemic, performances remain comparable: the average speed being still the same, the sets of encountered contacts remain very similar.

However, speed variations have an impact on contact patterns since irregularity in the mobility speed of holders increases both disconnections and new contacts leading to more data exchanges. Such an impact is reflected by the increase in the number of exchanged frames, observed in the third section of Table II.

Finally, simulations are run considering a constant speed but mobility is modified by allowing holders to head towards a random CP, which may not be the closest one in relation to their current positions.

Results are presented in Figure 4 and the last section of Table II. Performance efficiency remains quite similar with 3 CPs. However, with more CPs added, this mobility variation yields to spectacular improvements (more than 60% better with 20 CPs and Wi-Fi / SnW). The reason for such great performance improvement is that divergent holders disseminate information among holders with different target CPs, and, thus, mitigate holders' dispersion.

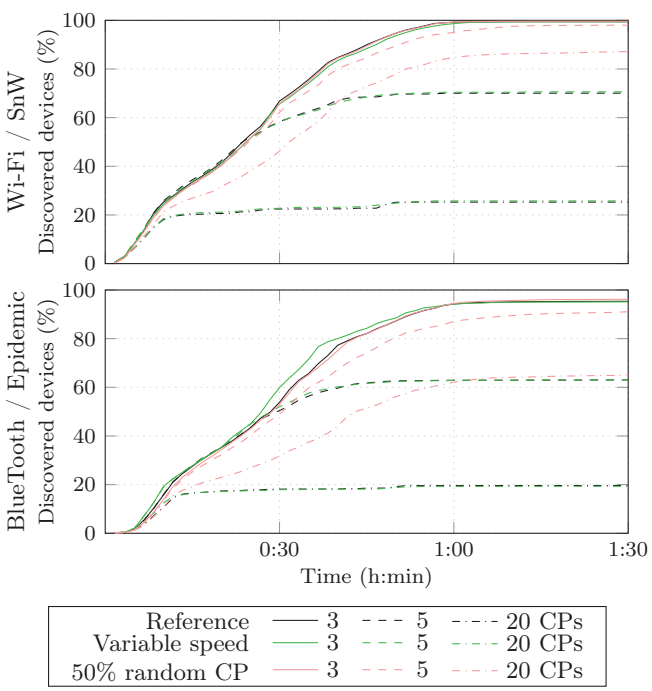


Fig. 4. Mobility variations – walk speed & CP choice

VI. Conclusion

In this paper, we have proposed a new post-disaster ad hoc communication architecture for collecting geolocation information about victims and survivors. Our study focused on efficiency and speed of information gathering as well as energy cost in terms of number of exchanged messages. By defining a reference scenario, we conducted several experiments on top of the ONE simulator to assess the effect of different parameters on the overall performance. Mobility patterns and interface choices were revealed to have a significant impact on both efficiency and energy consumption, whereas protocol choices and interface intermittence have a limited (and interface-range specific) one. Furthermore, walk speed has shown to have little interest in the considered scenario.

Aiming at optimizing protocols routing choices, future directions involve the study of the characteristics of dynamic graphs built over time induced by holders encounters. In a close future, we intend to enrich our disaster scenario considering, for instance, larger disaster areas or adding other participants such as vehicles, rescue teams with specific mobility patterns, etc.

References

[1] C. Reuter, T. Ludwig, and V. Pipek, "Ad hoc participation in situation assessment: supporting mobile collaboration in emergencies," *ACM Trans. Comput.-Hum. Interact.*, vol. 21, no. 5, 26:1–26:26, 2014.

[2] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," 2000.

[3] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: an efficient routing scheme for intermittently connected mobile networks," in *Proceedings of the ACM SIGCOMM Workshop on Delay-tolerant Networking*, 2005, pp. 252–259.

[4] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE simulator for DTN protocol evaluation," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 2009.

[5] S. Bhattacharjee, S. Roy, and S. Bandyopadhyay, "Exploring an energy-efficient DTN framework supporting disaster management services in post disaster relief operation," vol. 21, no. 3, pp. 1033–1046, 2014.

[6] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and focus: efficient mobility-assisted routing for heterogeneous and correlated mobility," in *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, 2007, pp. 79–85.

[7] A. Lindgren, A. Doria, and O. Schelen, "Probabilistic routing in intermittently connected networks," in *SIGMOBILE Mobile Computing and Communication Review*, 2004, p. 2003.

[8] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "MaxProp: routing for vehicle-based disruption-tolerant networks," in *Proc. of IEEE INFOCOM*, 2006.

[9] P. Stavroulakis, *Terrestrial Trunked Radio - TETRA*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ISBN: 978-3-540-71190-2. [Online]. Available: http://link.springer.com/10.1007/3-540-71192-9 (visited on 07/19/2016).

[10] Project loon, [Online]. Available: https://www.google.com/loon/ (visited on 03/08/2016).

[11] A. Utani, T. Mizumoto, and T. Okumura, "How geeks responded to a catastrophic disaster of a high-tech country: rapid development of counter-disaster systems for the great east japan earthquake of march 2011," in *Proceedings of the ACM Special Workshop on Internet and Disasters*, 2011, 9:1–9:8.

[12] Ushahidi, [Online]. Available: http://www.ushahidi.com (visited on 03/08/2016).

[13] SahanaEden, [Online]. Available: http://eden.sahanafoundation.org/ (visited on 10/28/2014).

[14] Google crisis response, [Online]. Available: https://www.google.org/crisisresponse/about/ (visited on 03/08/2016).

[15] Disaster alert network - ubAlert, [Online]. Available: https://www.ubalert.com/ (visited on 06/20/2014).

[16] G. Pearson, M. Gill, S. Antani, L. Neve, G. Miernicki, K. Phichaphop, A. Kanduru, S. Jaeger, and G. Thoma, "The role of location for family reunification during disasters," in *Proceedings of the First ACM SIGSPATIAL International Workshop on Use of GIS in Public Health*, New York, NY, USA, 2012, pp. 11–18.

[17] T. Hossmann, F. Legendre, P. Carta, P. Gunningberg, and C. Rohner, "Twitter in disaster mode: opportunistic communication and distribution of sensor data in emergencies," in *Proceedings of the 3rd ACM Extreme Conference on Communication: The Amazon Expedition*, 2011, p. 1.

[18] FireChat, [Online]. Available: http://opengarden.com/ (visited on 03/22/2016).

[19] The serval project, [Online]. Available: http://www.servalproject.org/ (visited on 03/08/2016).

[20] A. Meissner, T. Luckenbach, T. Risse, T. Kirste, and H. Kirchner, "Design challenges for an integrated disaster management communication and information system," in *Prociding os The First IEEE Workshop on Disaster Recovery Networks*, vol. 24, 2002.

[21] K. Fall, "A delay-tolerant network architecture for challenged internets," in *The 2003 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, pp. 27–34.

[22] D. Reina, S. Toral, F. Barrero, N. Bessis, and E Asimakopoulou, "Evaluation of ad hoc networks in disaster scenarios.," in *the Third International Conference on Intelligent Networking and Collaborative Systems*, 2011, pp. 759–764.

[23] S. Saha, Sushovan, A. Sheldekar, R. J. C, A. Mukherjee, and S. Nandi, "Post disaster management using delay tolerant network," in *Proceedings of Recent Trends in Wireless and Mobile Networks*, 162, 2011, pp. 170–184.

[24] C. Raffelsberger and H. Hellwagner, "A hybrid MANET-DTN routing scheme for emergency response scenarios," in *The IEEE International Conference on Pervasive Computing and Communications Workshops*, 2013, pp. 505–510.

[25] A. Martín-Campillo, J. Crowcroft, E. Yoneki, and R. Martí, "Evaluating opportunistic networks in disaster scenarios," vol. 36, no. 2, pp. 870–880, 2013.

[26] L. E. Quispe and L. M. Galan, "Behavior of ad hoc routing protocols, analyzed for emergency and rescue scenarios, on a real urban area," in *Expert Systems with Applications*, 5, vol. 41, 2014, pp. 2565–2573.

[27] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark, "Scenario-based performance analysis of routing protocols for mobile ad-hoc networks," in *The 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1999, pp. 195–206.

[28] M. Boc, A. Fladenmuller, M. D. de Amorim, L. Galluccio, and S. Palazzo, "Price: hybrid geographic and co-based forwarding in delay-tolerant networks," in *Computer Networks*, 9, vol. 55, 2011, pp. 2352–2360.

[29] J. Yoon, M. Liu, and B. Noble, "Random waypoint considered harmful," in *INFOCOM 2003*, vol. 2, 2003, pp. 1312–1321.

[30] B. Karp and H. T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *The 6th Annual International Conference on Mobile Computing and Networking*, 2000, pp. 243–254.

[31] M. Uddin, D. Nicol, T. Abdelzaher, and R. Kravets, "A post-disaster mobility model for delay tolerant networking," in *Simulation Conference (WSC), Proceedings of the 2009 Winter*, 2009, pp. 2785–2796.

[32] S. Ganguly, S. Basu, S. Roy, and S. Mitra, "A location based mobility prediction scheme for post disaster communication network using DTN," in *Applications and Innovations in Mobile Computing), 2015*, 2015, pp. 25–28.

[33] D. G. Reina, J. M. León-Coca, S. L. Toral, E. Asimakopoulou, F. Barrero, P. Norrington, and N. Bessis, "Multi-objective performance optimization of a probabilistic similarity/dissimilarity-based broadcasting scheme for mobile ad hoc networks in disaster response scenarios," in *Soft. Computing*, 9, vol. 18, 2013, pp. 1745–1756.

[34] Y. Huang, W. He, K. Nahrstedt, and W. Lee, "CORPS: event-driven mobility model for first responders in incident scene," in *IEEE Military Communications Conference, 2008. MILCOM 2008*, pp. 1–7.

[35] N. Aschenbruck, E. Gerhards-padilla, M. Gerharz, M. Frank, and P. Martini, "Modelling mobility in disaster area scenarios," in *The 10th ACM 10th ACM Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, 2007.

[36] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang, "A group mobility model for ad hoc wireless networks," in *The 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, 1999, pp. 53–60.

[37] L. Conceição and M. Curado, "Modelling mobility based on human behaviour in disaster areas," in *Wired/Wireless Internet Communication*, 7889, Springer Berlin Heidelberg, 2013, pp. 56–69.

[38] D. G. Reina, M. Askalani, S. L. Toral, F. Barrero, E. Asimakopoulou, and N. Bessis, "A survey on multihop ad hoc networks for disaster response scenarios," in *International Journal of Distributed Sensor Networks, International Journal of Distributed Sensor Networks*, vol. 2015, 2015, p. 647 037.

[39] F. Coriat, L. Arantes, O. Marin, A. Fladenmuller, N. Hidalgo, and E. Rosas, "Towards distributed geolocation for large scale disaster management," in *WSDP - Chilean Workshop on Distributed and Parallel Systems*, 2014.

GeoTrie: A Scalable Architecture for Location-Temporal Range Queries over Massive GeoTagged Data Sets

Rudyar Cortés[1], Xavier Bonnaire[2], Olivier Marin[3], Luciana Arantes[1], and Pierre Sens[1]

[1]Sorbonne Universités, UPMC Univ Paris 06, CNRS
INRIA - LIP6, Paris, France
E-mail: [rudyar.cortes, luciana.arantes, pierre.sens]@lip6.fr
[2]Universidad Técnica Federico Santa María, Valparaíso, Chile
E-mail: xavier.bonnaire@inf.utfsm.cl
[3]New York University, Shanghai, China
E-mail: ogm2@nyu.edu

***Abstract*—The proliferation of GPS-enabled devices leads to the massive generation of *geotagged* data sets recently known as Big Location Data. It allows users to explore and analyse data in space and time, and requires an architecture that scales with the insertions and location-temporal queries workload from thousands to millions of users. Most large scale key-value data storage solutions only provide a single one-dimensional index which does not natively support efficient multidimensional queries. In this paper, we propose GeoTrie, a scalable architecture built by coalescing any number of machines organized on top of a Distributed Hash Table. The key idea of our approach is to provide a distributed global index which scales with the number of nodes and provides natural load balancing for insertions and location-temporal range queries. We assess our solution using the largest public multimedia data set released by Yahoo! which includes millions of geotagged multimedia files.**

***Index Terms*—Big Location Data; Location-Temporal Range Queries; Scalability.**

I. Introduction

An ever increasing number of GPS-enabled devices such as smartphone and cameras generate geotagged data. On a daily basis, people leave traces of their activities involving mobile applications, cars, unmanned aircraft vehicles (UAVs), ships, airplanes, and IoT devices. These activities produce massive flows of data which cannot be acquired, managed, and processed by traditional centralized solutions within a tolerable timeframe. The time and geographic analysis of such data can be crucial for life saving efforts by helping to locate a missing child, or for security purposes allowing to know which smartphone and therefore who was present in a given area and during a specific event. An emergent field of research, known as *Big Location Data* [1], [2], addresses this issue.

The essential characteristic of *Big Location Data* is that it associates every data with meta-data which includes a geotag and a timestamp. *Location-temporal range queries* [3] represent a major challenge for data extraction, exploration and analysis, within both a geographic area and a time window.

978-1-5090-3216-7/16/31.00

Services like automatic smartphones location or smart cities, can generate meta-data on an even larger scale. Allowing millions of users to explore these data sets could help get more insight about what happens in particular geographic areas within given timeframes. For instance, people who attended a concert between 20:00 p.m. and 01:00 a.m. may want to review public pictures and comments from people who attended the same concert. The concert manager may want to acquire more feedback about the overall concert experience by analyzing pictures, comments and tags. Or people might want to verify whether the referee of a football match missed a foul by catching multiple amateur videos and pictures of the action.

The main challenge presented by such services is to guarantee scalability, fault tolerance, and load balancing for a high number of concurrent insertions and location-temporal range queries. Relational databases cannot reach such scales: their response time significantly increases when concurrent insertions and queries have to cover billions of objects. Other approaches that rely on a centralized global index [4]–[6] contend with a bottleneck. More recent approaches use *space-filling curves* [7] in order to collapse the latitude and longitude coordinates into a one-dimensional index and distribute it over multiple nodes [8]–[10]. Although they allow scalable location-temporal query processing, the *curse of dimensionality* [7] introduces several processing overheads which impact the query response time.

In this paper, we present a scalable architecture for location-temporal range queries. The main component of the architecture is a distributed multidimensional global index which supports location-temporal range querying on a large scale, and provides natural insertion and query load balancing.

The main contributions of this paper are:

- An approach to introduce location-temporal data locality in a fully decentralized system such as a distributed hash table. This approach is based on a multidimensional distributed index which maps

$latitude, longitude, timestamp$ tuples into a distributed prefix octree, thus efficiently filtering false positives while providing fault tolerance and load balancing for internet-scale applications.
- A theoretical evaluation of our solution, which shows that its message complexity for insertions and range queries is logarithmic with respect to the number N of nodes involved.
- A practical evaluation, which shows that our solution alleviates the bottleneck on the root node during insertions and range queries. This property has an impact on the query response time. For instance, queries which avoid the root node presents an average query response time up to 1.9x faster than queries which starts at the root level.

The rest of this paper is organized as follows. We detail our solution in Section II, and then evaluate it in Section III using a large public multimedia dataset from Yahoo!. We give an overview of the related work in Section IV, before concluding in Section V.

II. System Design

We aim to propose a scalable architecture for indexing and querying meta-data extracted from geotagged data sets.

We choose to work on top of a distributed hash table (DHT) such as Pastry [11] or Chord [12] because it provides a strong basis for scalable solutions as opposed to centralized server architectures. DHTs provide an overlay with properties such as self-organisation, scalability to millions of nodes (usually routing has a message complexity of $O(log(N))$, where N is the number of nodes) and dependability mechanisms via replication. However, a DHT by itself cannot constitute a satisfactory solution for scalable location-temporal data storage and retrieval because the hash function usually destroys data locality for the sake of load balancing.

The proposed architecture must create data locality over DHTs based on three core dimensions: latitude, longitude and timestamp. We assume that every meta-data entry references data stored on an external storage service. We intent for our architecture to achieve the following properties.

- *Self-organisation:* The architecture must be self-organised. That is, every node must run a local protocol without needing to add a central third party.
- *Scalable query processing:* Meta-data insertions and location-temporal range queries must scale with the number of available nodes in the DHT.
- *Query load balance:* Meta-data insertions and location-temporal range queries must be distributed in the architecture in order to avoid a single bottleneck.
- *High Availability:* The architecture must tolerate node failures.

Geotrie builds location-temporal data locality over a DHT following two main steps: *mapping* and *indexing*. The *mapping* step associates every (latitude,longitude,timestamp) coordinate with a tuple key $T_k = (T_{lat}, T_{lon}, T_t)$, where every coordinate of T_k is a 32-bit word. The *indexing* step inserts every tuple key T_k in a fully distributed prefix octree which allows efficient location-temporal range queries.

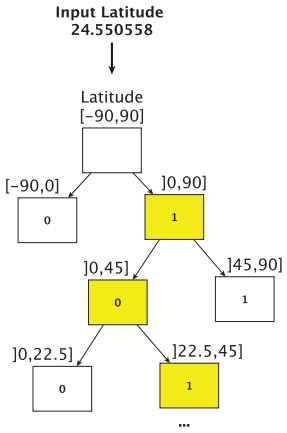


Fig. 1: Latitude mapping example. The latitude value 24.550558 is mapped to the binary string $T_{lat} = 10100010111010101001010111011 01$

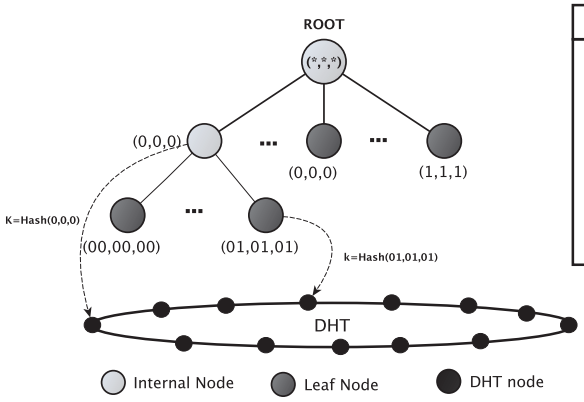
A. Mapping

Let $lat \in [-90, 90]$, $lon \in [-180, 180]$, and $t \in [0, 2^{32} - 1]$ be the latitude, longitude and timestamp parameters of a given geotagged object. The timestamp coordinate corresponds to the *Unix epoch* timestamp. We map every coordinate to the multidimensional tuple key $T_k = (T_{lat}, T_{lon}, T_t)$, where every tuple belongs to the same domain $\{0, 1\}^{32}$ (i.e, binary string of 32 bits per coordinate).

Geotrie performs domain partitioning for all the coordinates as follows. First, the space domain $[a, b]$ of every coordinate is divided in two buckets $D_{left} = [a, (a + b)/2]$ and $D_{right} =](a+b)/2, b]$. A bit value of 0 represents a point that belongs to the left sub domain, while a bit value of 1 positions it in the right subdomain. This process continues recursively until all $D = 32$ bits are set. For instance, the first bit of latitude -45 is 0 because it belongs to the first left subdomain [-90,0]. This mapping function reaches its lower bound at latitude value $lat = -90$ represented as the identifier $\{0\}^{32}$, and its upper bound at $lat = 90$ represented as the identifier $\{1\}^{32}$. Longitudes incur the same process, with the lower bound at $lon = -180$ and the upper bound at $lon = 180$. The mapping of the timestamp coordinate follows the same principle. It results in the binary representation of the *Unix epoch*. Figure 1 presents an example of the mapping strategy for latitude value 24.550558.

Every tuple T_k encloses coordinates into a location-temporal cell which covers an area of 0.46 cm $\times$ 0.93 cm at the equator, and ensures a one second time precision. It also induces the following properties.

- *Recursive prefix domain partition*. The mapping function presented above recursively partitions every latitude, longitude, and timestamp coordinates into eight areas represented by prefixes. For instance, the tuple key



GeoTrie Node	State	Tuple Keys
(*,*,*)	Internal Node	--
(0,0,0)	Internal Node	--
(00,00,00)	Leaf Node	00,00,00
(01,01,01)	Leaf Node	01,01,01
(0,0,1)	Leaf Node	00,00,10
...	...	...
(1,1,1)	Leaf Node	10,10,10

Fig. 2: GeoTrie Example: Distributed data structure for keys of D=2 bits size

Input: Tuple key T_k
Output: Leaf node
$lower = 0$;
$higher = D$;
while $lower \leq higher$ **do**
  $middle = (lower + higher)/2$;
  // Extract the prefix of size middle of every coordinate of T_k and route the message
  $node = DHT\_lookup(T_{middle}(T_k))$;
  if *node is a leaf node* **then**
    return $node$;
  else
    if *node is an internal node* **then**
      $lower = middle + 1$;
    else
      // node is external node
      $higher = middle - 1$;
    end
  end
end
return $failure$;

Algorithm 1: LocateLeaf pseudocode

Input: Tuple key T_k
Meta-data m
node = LocateLeaf(T_k);
node.insert(leafAddr,T_k,m);

Algorithm 2: Insertion pseudocode

$T_k = (0, 0, 0)$ represents all the latitude,longitude, and timestamp tuples within the interval latitude ($[-90, 0]$), longitude ($[-180, 0]$), and timestamp $[0, (2^{32} - 1)/2]$). The next subdomain $T_k = (00, 00, 00)$ represents the domain latitude ($[-90, -45]$), longitude ($[-180, -90]$), and timestamp = ($[0, (2^{32} - 1)/4]$), and so on.

- *Prefix data locality*. Shared prefixes imply closeness. That is, all keys with the same prefix necessarily belong to the single and same area represented by this prefix. For instance, all keys that share the same prefix key (00,00,00) belong to the interval covered by this prefix. Note that the converse is not true; for instance 011 and 100 are adjacent but they do not share a common prefix.

B. Indexing

The GeoTrie structure exploits both properties presented above. It indexes every tuple key $T_k = (T_{lat}, T_{lon}, T_t)$ into a distributed prefix octree-like indexing structure built on top of a DHT.

Every GeoTrie node holds a label l, a state s, and the range it covers. The label l is a prefix of T_k and the state s can either be *leaf node*, *internal node*, or *external node*. Only *leaf nodes* store data; *internal nodes* stand on the path to leaf nodes that do hold data, while *external nodes* stand outside any such path. A DHT node declares itself to be an *external node* for a label l, when it receives a query asking for a non existing label l on its local data structure. The range covered by a GeoTrie node is locally computed by using its label l and the *recursive prefix domain partition* property presented above. Furthermore, every *internal node* stores links to its direct children nodes and the range they cover.

Every GeoTrie node is assigned to the DHT node whose identifier in the ring is closest to the key $k = Hash(l)$.

Upon start-up, GeoTrie consists of a single root node with label $l = (*, *, *)$ and *leaf node* state. When a node becomes full, GeoTrie scales out by switching it to internal and dispatching its load onto eight new *leaf nodes* created via recursive prefix domain partitioning.

Figure 2 shows a representation of the GeoTrie data structure for tuple keys of size $D = 2$ bits. GeoTrie can take advantage of any replication mechanism used by DHTs in order to provide fault tolerance. For instance, in Pastry [11] Geotrie can use the *leaf set* in order to replicate and maintain the state of every Geotrie node.

Insertions and deletions. Storing and deleting an object with key T_k consists first in locating the *leaf node* whose label is prefix of T_k, and then in carrying out the insertion

or deletion operation directly on this node. For instance in Figure 2, tuple key $T_{k_1} = (00, 00, 00)$ is stored on the *leaf node* with label $l_1 = (00, 00, 00)$ and tuple key $T_{k_2} = (10, 10, 10)$ is stored on the leaf node with label $l_2 = (1, 1, 1)$. Algorithm 2 presents a pseudocode of the insertion operation; it uses the *locateLeaf* procedure detailed in Algorithm 1. *locateLeaf* receives a tuple key T_k as input and returns the *leaf node* whose label is prefix of T_k. It does so by performing a binary search over different possible prefixes of T_k until a node with state *leaf node* is reached. For every tuple key T_k, there are exactly D possible prefix labels plus the root node, and only one them can designate a *leaf node*. The client starts by sending a lookup message to a label prefix of T_k of size $D/2$ (i.e, the middle of the space of possible candidates). This label is computed by extracting the first $D/2$ bits of every coordinate. If the state of this node is *external*, it means that the target *leaf node* keeps a label of shorter prefixes. In this case, the binary search cuts the space to prefixes of higher size $D/2 - 1$ and it propagates the lookup to shorter prefixes. Instead, if the target node is an *internal node* it means that the *leaf node* keeps a longer prefix. It cuts the space to a lower prefix of size $D/2 + 1$ and propagates the search down GeoTrie. This process continues recursively and ends upon reaching a *leaf node* whose label is prefix of T_k. If the *locateLeaf* algorithm fails to return a leaf node due to index maintenance, the client must retry the insertion at a later time. This binary search procedure benefits of the prefix property of GeoTrie which allows queries to start at any node, and thus prevents the root node from becoming a bottleneck.

Index maintenance. GeoTrie provides two index maintenance operations: *split* and *merge*. The *split* operation occurs when a *leaf node* stores B keys, where B is a system parameter. An overloaded *leaf node* scales out its load by creating eight new children nodes via recursive domain partitioning. The dispatch of data follows the prefix rule: a data entry with index T_k gets transferred to the new *leaf node* whose label l is prefix of T_k. After transferring all its data, the *split* node changes its state from *leaf node* to *internal node* and every child node becomes a new *leaf node*.

The *merge* operation is the opposite to the *split* operation. GeoTrie triggers the *merge* operation on a group of eight *leaf nodes* which share the same *internal node* as parent when the sum of their storage loads becomes less than $\lfloor B/8 \rfloor$ objects. When this happens the *internal node* sends a *merge message* to all its *leaf node* children, thus requesting they transfer back all their stored data. Upon transfer completion, all children *leaf nodes* detach from the prefix tree structure and the parent switches its state from *internal node* to *leaf node*. Typical applications generate far more insertions than deletions, so we expect a low proportion of *merge* operations compared to *split* operations.

Location-temporal range queries. A location-temporal range query is represented as a three dimensional range query

Input: $(\Delta Lat_b, \Delta Lon_b, \Delta t_b)$
Output: $Meta - data$
$prefixLat = commonPrefix(\Delta Lat_b)$;
$prefixLon = commonPrefix(\Delta Lon_b)$;
$prefixTime = commonPrefix(\Delta t_b)$;
// Common prefix of minimum size
$target$ = minComPrefix(PrefixLat,PrefixLon,PrefixTime);
// If there is not a common prefix
start from the root node
if $target = (,,,)$ **then**
  $target = (*, *, *)$;
end
$node = DHT - lookup(target)$;
if *node is a leaf node* **then**
  // process the request
  return LocalRequest($\Delta Lat, \Delta Lon, \Delta t$);
else
  if *node is an internal node* **then**
    // forward the request to the children nodes that covers the range
    $forwardRequest(children)$;
  else
    // Node is an external node
    // Locate the leaf node which covers all the interval using the target label as starting point
    $node = LocateLeaf(target)$ // Forward the request to this node
    $forwardRequest(node)$
  end
end

Algorithm 3: Location-temporal range query pseudocode

as presented in equation 1.

$$\begin{array}{rcccc} \Delta Lat & = & [lat1, lat_2], & lat_1, lat_2 \in [-90, 90] \\ \Delta Lon & = & [lon_1, lon_2], & lon_1, lon_2 \in [-180, 180] \\ \Delta t & = & [t_i, t_f], & t_i, t_f \in [0, 2^{32} - 1] \end{array} \quad (1)$$

This query is resolved as follows. First, the sender node uses the mapping function defined above in order to translate every coordinate constraint $(\Delta Lat, \Delta Lon, \Delta t)$ into binary strings belonging to the domain $\{0, 1\}^{32}$ as presented in equation 2.

$$\begin{array}{rcccc} \Delta Lat_b & = & [lat_1^{32}, lat_2^{32}], & lat_1^{32}, lat_2^{32} \in \{0, 1\}^{32} \\ \Delta Lon_b & = & [lon_1^{32}, lon_2^{32}], & lon_1^{32}, lon_2^{32} \in \{0, 1\}^{32} \\ \Delta t_b & = & [t_i^{32}, t_f^{32}], & t_i^{32}, t_f^{32} \in \{0, 1\}^{32} \end{array} \quad (2)$$

Then, it computes the common prefix label of minimum common size for every coordinate constraint and it forwards the query to the node which covers this label. For instance, query Q which combines constraints $\Delta Lat_b = [00..., 01...]$, $\Delta Lon_b = [10..., 11...]$, and $\Delta t_b = [110..., 110...]$ has a common prefix label of every coordinate $(0, 1, 11)$. This common prefix label has not a common size because the time coordinate is longer than the others. Thus, this common prefix

is converted to a label of minimum common size $l_Q = (0, 1, 1)$ which represents a label of GeoTrie. This label covers all data which is inside the interval of Q. If there is not common prefix, the query must start from the root node labeled $l = (*, *, *)$. Algorithm 3 presents a pseudocode for this procedure.

Depending on the state s of the node which receives the query, we identify three cases. (i) If the node is an *internal node*, it uses the *forwardRequest* function in order to recursively forward the query onward to the *leaf nodes* whose location-temporal range intersects that of the query interval. (ii) If the node is a *leaf node*, it is the only node that covers the required location-temporal range and returns the objects that satisfy the query. (iii) If the node is an *external node*, it follows that the common prefix is a label which does not yet exist in the prefix tree (i.e, the query arrived to a DHT node which does not hold this label). In this case, the query is necessarily covered by a single *leaf node*. in order to find this particular *leaf node*, the client node then starts the *LocateLeaf* algorithm from the label of the *external node*.

III. Evaluation

This section presents a theoretical and an experimental evaluation of GeoTrie. The theoretical evaluation assesses the scalability of the solution in terms of the message complexity for insertions and range queries. An extensive experimental evaluation studies the load balancing and the performance of GeoTrie in terms of the query response time. We chose to conduct our experimental evaluation with a real dataset: the Yahoo Flickr Creative Commons (YFCC) dataset [13] released by Yahoo! It comprises $48,469,177$ geotagged multimedia files (photos and videos).

A. Overview

This section presents a simulation-based assessment of GeoTrie to measure its performance, with a large scale real world dataset from Yahoo!

We implemented GeoTrie on top of FreePastry [14], an open-source DHT implementation. In our experiments, we deployed $N = 1,000$ DHT nodes. Every *leaf node* stores at most $B = 10,000$ keys. We assume that DHT nodes are distributed worldwide. We used a baseline latency file provided in the FreePastry distribution to generate latencies among DHT nodes. Table I gives the baseline values, while Figure 3a shows the observed distribution of latencies on the DHT nodes during our experiments.

1) Query set: In order to assess the performance of GeoTrie under different range query spans, we generated six query sets (QS) which cover different geospatial bounding box sizes and timespans as follows. First, we randomly picked $1,000$ tuples $t_k = (latitude, longitude, timestamp)$ from the dataset. Then, for every query set and for every tuple, we generated a query for a geospatial bounding box which encloses a $M \times M$ kilometres square with a time interval of H hours with centre in t_k.

Table II presents the size of M and H we chose for this evaluation and Table III presents the distribution of the data

Distribution	Latency [ms]
Min	2
Quartile 1	178
Quartile 2	225
Quartile 3	269
Max	350

TABLE I: Distribution of latencies among DHT nodes

Query set (QS)	B. Box $M \times M(km^2)$	Time interval (H)
1	2×2	1
2	2×2	48
3	200×200	1
4	200×200	48
5	ALL	1
6	2×2	ALL

TABLE II: Location-temporal range query sets

Dist.	QS1	QS2	QS3	QS4	QS5	QS6
Min.	1	1	1	1	60	1
Quartile 1	6	12	15	95	535	259
Quartile 2	21	37	36	256	771	1,393
Quartile 3	56	113	81	612	1,027	9,089
Max.	5,353	62,333	6,079	77,629	6,694	215,647
Avg.	50	169	68	555	825	14,008

TABLE III: Distribution of the number of data items that match every query set QS

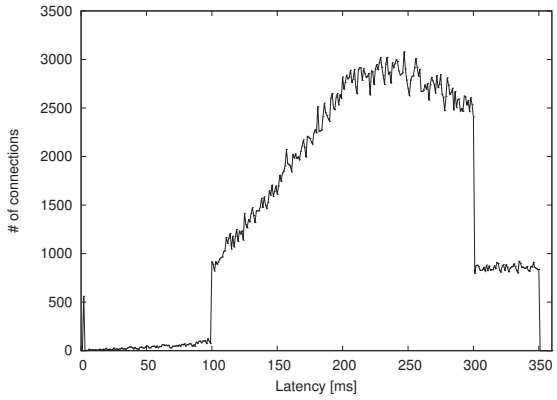
Dist.	QS1	QS2	QS3	QS4	QS5	QS6
Min	1	1	1	1	796	4
Quartile 1	1	1	5	5	1693	15
Quartile 2	1	1	9	9	2023	23
Quartile 3	1	1	14	14	2161	38
Max	31	82	54	108	2227	284

TABLE IV: Query cost distribution.

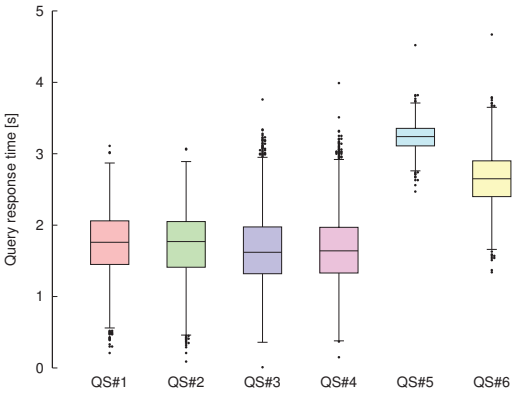
items that match every query set. These queries represent different bounding box sizes and timespans. For instance, query sets **QS1** and **QS2** represent highly selective queries. They target bounding boxes of 2×2 km^2 and timespans of 1 and 48 hours respectively. Query sets **QS3** and **QS4** target bigger bounding boxes of 200×200 km^2, with respective timespans of 1 and 48 hours. The query set **QS5** targets all the space domain with a timespan of one hour. Finally, the query set **QS6** targets all the time domain with a bounding box of two kilometres side size. Query sets **QS5** and **QS6** are specific in that the former covers the entire geospatial domain and the latter covers the entire temporal domain. Therefore, they both always start at the root node because of the absence of a common prefix. We believe that these combinations of queries represent a rich workload to assess the query load balance and performance of GeoTrie.

B. Data insertions

1) Message Complexity: Let N be the number of nodes in the DHT, n the number of keys to be indexed, and D the number of bits used to represent the data domain $\{0, 1\}^D$ of every coordinate of the tuple key T_k. The insertion/deletion



(a) Distribution of latencies among DHT nodes



(b) Location-temporal range query response time

Fig. 3: Latency among DHT nodes and GeoTrie Performance

Distribution	Time[s]
Min	1 x 10^{-3}
Quartile 1	1.80
Quartile 2	2.84
Quartile 3	3.33
Max	6.12

TABLE V: Insertion latency distribution

# DHT lookups	Percentage [%]	Min. Time[s]	Avg. Time [s]	Max Time [s]
1	0.05	1 x 10^{-3}	0.83	1.62
2	27.79	1 x 10^{-3}	1.47	3.20
3	2.01	0.387	2.12	3.90
4	20.27	0.49	2.71	5.05
5	49.84	0.82	3.34	6.12
6	0.04	2.32	3.88	5.76

TABLE VI: Number of DHT lookups per insertion and its latency distribution

of a data object identified by T_k involves a binary lookup in order to find the *leaf node* whose label is a prefix of T_k.

Equation 3 presents the message complexity in terms of the number of DHT lookup messages generated by a single insertion/deletion operation. Note that the message complexity of every DHT lookup is $O(\log(N)$. So the total number of messages generated by a single insertion/deletion operation is given by equation 4.

$$C_{ins/del}(D) = O(\log(D)) \tag{3}$$

$$C_{ins/del\_total} = O(log(D)) \times O(log(N)) \tag{4}$$

The message complexity of insertions and deletions on GeoTrie does not depend on the number of objects n. A balanced tree index built on top of a DHT requires $O(\log(n))$ DHT lookups. Via binary searches on a trie whose height cannot exceed 32, GeoTrie outperforms a balanced tree for large values of n. For instance, indexing a new entry among 50 million keys of size $D = 32$ bits requires about 25 DHT lookups with a tree, while GeoTrie does the job in about 5 DHT lookups.

2) Insertion time: In this experiment, we indexed all the meta-data of $48,469,177$ geotagged multimedia files extracted from the YFCC dataset [13] at a rate of $1,000$ items per second. At every insertion, a DHT node is chosen randomly in order to perform the operation. Then, we measured the insertion time as the time elapsed between the moment the client node starts the insertion process until the moment the leaf node successfully stores the item.

Table V presents the overall distribution of insertion latencies. Most of the latencies are concentrated between 1.8 and 3.3 seconds with a median of 2.84 seconds. The insertion time depends of the size of the prefix label of the target *leaf node*. Some insertions can directly arrive to the target *leaf node* through the binary search algorithm presented in section II. Others insertions require more messages to locate the target *leaf node*. In order to understand the behaviour of the binary search we measured the distribution of the number of DHT lookups in the insertion process. Table VI presents our results. Since the binary search algorithm is inherently sequential, the number of DHT lookups has a strong impact on the insertion latency. For instance, most insertions (49.84$\%$) require 5 DHT lookups. They present an average latency of 3.34 seconds. Instead, insertions that require 2 lookups (27.79 %) present an average insertion latency of 1.47 seconds (2.27 times faster).

C. Location-temporal range queries

1) Message Complexity: A location-temporal query on GeoTrie first reaches the node which maintains the common prefix label of minimum common size, and then branches out in parallel down the tree until it reaches all *leaf nodes*. Equation 5 computes the number of messages for this operation.

$$O(log(N)) \leq C_{range-query} \leq O(log(N)) + D + 1 \tag{5}$$

The lower bound occurs when the query arrives directly to a single *leaf node* which covers the interval, and the upper bound occurs when there is no common prefix of minimum size. In the worst case, the query arrives at the root node labeled $l = (*, *, *)$ and traverses the whole tree in parallel through a

(QS#)	Min	Quartile 1	Quartile 2	Quartile 3	Max	Avg
1	0.21	1.45	1.76	2.06	3.11	1.72
2	0.09	1.41	1.77	2.05	3.07	1.69
3	0.01	1.32	1.62	1.97	3.76	1.68
4	0.15	1.33	1.64	1.97	3.99	1.69
5	2.47	3.11	3.24	3.35	4.52	3.23
6	1.34	2.4	2.65	2.90	4.67	2.66

TABLE VII: Location-temporal range query response time in seconds

Distribution	QS1 SP	QS3 SP
Min	0	0
Quartile 1	5	3
Quartile 2	6	5
Quartile 3	6	5
Max	12	6

TABLE VIII: Distribution of the GeoTrie level starting point for **QS1** and **QS3**

maximum of $D + 1$ levels (the root node plus the maximum prefix size D=32). Note that the global implicit knowledge introduced by GeoTrie allows to reach the query subspace directly when it does not include the root node. Unlike queries that cover large geographic areas and timespans, queries which target small geographic areas and timespan share a larger common prefix of minimum size, and therefore directly arrive to higher levels of GeoTrie (i.e., close to the leaves). Indexes relying on a balanced tree usually start queries from the root node, and thus induces bottlenecks.

2) Range query performance: In this experiment, we study queries whose prefix maps to any other node than the root node. More specifically, we assess their impact on the query response time under a workload composed of concurrent range queries. In order to conduct this experiment, we generated a workload of $1,000$ queries per second for every set **QS1** to **QS5**. Every query originates from a DHT node chosen at random. It consists in counting the number of items inside the location-temporal range.

Figure 3b and table VII presents the query response time measured as the time elapsed between emission of the query and the latest reception of results from all the *leaf nodes* that store data relevant to this query. Only about 1% of the query response time corresponds to processing time (i.e, the time it takes a *leaf node* to filter-out all the data outside the query interval). The remaining 99% of the time corresponds to communication latency. Indeed, in our experiment the metadata is stored in memory, and therefore the local processing cost of a query over a single *leaf node* is much lower than the network latency cost. As expected, queries that span a shorter space present a lower query response time than queries that span a larger space. For instance, the average query response time is 1.9x slower for queries of **QS3** than for queries of **QS5**. Indeed, we tailored queries in this experiment so that they can avoid the root node. Our results show that Geotrie thus allows to alleviate a potential bottleneck on the root node, and this has a considerable impact on the query response time.

In order to understand how queries are distributed among the GeoTrie structure we measured the GeoTrie level starting point (SP) for **QS1** and **QS3**. We exclude both **QS5** and **QS6** because by construction they always start at the root node. Table VIII presents our results. Only 0.3% of queries of **QS1** (3 out of 1,000) starts at the root node, and 75% of the total amount of queries starts at a level greater or equal than 5. In the case of **QS3**, 8.3% of the queries (83 out of 1,000) started at the root node and the 75% of the queries started at a level greater or equal than 3. **QS3** queries target bigger areas, and therefore match shorter common prefixes in the trie structure.

IV. Related Work

Large scale architectures to store, query and analyse *big location data* constitute a fast-growing research topic. Current solutions fall into four groups: (i) Hadoop-based solutions; (ii) Resilient Distributed Dataset (RDD) based solutions; (iii) Key-value store-based solutions, and (iv) DHT-based solutions.

Hadoop-based solutions such as SpatialHadoop [4], Hadoop GIS [5], and ESRI Tools for Hadoop [6], extend the traditional Hadoop architecture [15] with spatial indexing structures such as R-Trees [16] or Quad-Trees [17] in order to avoid a scan of the whole dataset when performing spatial analysis. These approaches employ a two-layer architecture that combines a global index maintained on a central server with multiple local indexes. For instance, SpatialHadoop [4] builds spatial indexing structures over HDFS [5] in order to add spatial support for MapReduce tasks. However, these solutions are ill-suited for concurrent insertions and location-temporal queries because (i) the global index structure on a single node is prone to become a bottleneck, and (ii) they are designed for batch processing of large tasks. Unlike these solutions, GeoTrie constitutes a large scale global location-temporal index which provides random access and fault tolerance for concurrent insertions and location-temporal queries.

Resilient Distributed Dataset (RDD) based solutions such as Spatial Spark [18] and GeoTrellis[1] extend traditional RDD solutions such as Spark [19] in order to support big location data. Similarly to Hadoop-based solutions, these systems are designed for batch processing and do not target online processing.

Key-value store-based solutions such as MD-Hbase [8], MongoDB [20], ElasticSearch [21], and GeoMesa [9] require linearization techniques such as *space-filling curves* [7] in order to collapse several dimensions into a single one-dimensional index and to support multi-dimensional queries. Then, the multidimensional query can be reduce to a single dimensional space. However, *space-filling curves* [7] loosely preserve data locality and introduce several I/O overheads when the number of dimensions increase due to the *curse of dimensionality* [7]. These overheads impacts negatively the query response time. Unlike these solutions, GeoTrie follows a multidimensional approach which drastically reduces this overhead.

DHT-based solutions can be classified into two main groups: (i) extensions of traditional indexing structures such

[1] http://geotrellis.io

as B-Trees, Prefix Trees, R-Trees, QuadTrees, and KDtrees to DHTs [22]–[26], and (ii) overlay-dependent solutions [27]–[30]. Compared to the solutions of the first group, GeoTrie performs *domain partitioning* to prevent bottlenecks on the root node, and introduces global knowledge about the tree structure to balance the load of insertions and range queries. To the best of our knowledge, the only structure which performs *domain partitioning* over a DHT is the Prefix Hash Tree (PHT) [22]. PHT can handle multi-dimensional data using *linearisation* techniques such as *z-ordering*. As mentioned previously, however, dimensionality reduction with respect to space and time introduces query overheads due to the *curse of dimensionality*. GeoTrie reduces this overhead because it uses a multidimensional structure and, unlike the solutions of the second group, it is portable to any DHT.

V. Conclusion

This paper presents GeoTrie, a scalable architecture for massive geotagged data storage and retrieval built on top of a DHT. Our solution indexes the location and temporal dimensions of every meta-data on a multidimensional distributed structure which scales and balances the load of insertions and range queries. A theoretical analysis of the message complexity of every operation on GeoTrie demonstrates its scalability. An extensive experimental evaluation over a Yahoo! dataset comprising about 48 millions of multimedia files shows that our solution balances the load of insertions and range queries on a large scale. In a configuration involving $1,000$ nodes, queries which avoids the root node presents an average query response time up to 1.9x faster than queries which starts at the root level. This property of GeoTrie allows to alleviate a potential bottleneck on the root node. We are currently working on an extension of our proposal to handle n-dimensional range queries over massive data sets.

References

[1] N. Pelekis and Y. Theodoridis. The case of big mobility data. In *Mobility Data Management and Exploration*, pages 211–231. Springer, 2014.

[2] A. Eldawy and M.F. Mokbel. The era of big spatial data: Challenges and opportunities. In *Mobile Data Management (MDM), 2015 16th IEEE International Conference on*, volume 2, pages 7–10, June 2015.

[3] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231, 1998.

[4] A. Eldawy and M. F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1352–1363, 2015.

[5] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, 6(11):1009–1020, 2013.

[6] Randall T Whitman, Michael B Park, Sarah M Ambrose, and Erik G Hoel. Spatial indexing and analytics on hadoop. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 73–82. ACM, 2014.

[7] Hanan Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[8] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Md-hbase: a scalable multi-dimensional data infrastructure for location aware services. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, volume 1, pages 7–16. IEEE, 2011.

[9] Anthony Fox, Chris Eichelberger, John Hughes, and Skylar Lyon. Spatio-temporal indexing in non-relational distributed databases. In *Big Data, 2013 IEEE International Conference on*, pages 291–299. IEEE, 2013.

[10] Kisung Lee, Raghu K Ganti, Mudhakar Srivatsa, and Ling Liu. Efficient spatial query processing for big data. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 469–472. ACM, 2014.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[12] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[13] B Thomee, DA Shamma, G Friedland, B Elizalde, K Ni, D Poland, D Borth, and LJ Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 2015.

[14] Peter Druschel, Eric Engineer, Romer Gil, Y Charlie Hu, Sitaram Iyer, Andrew Ladd, et al. Freepastry. *Software available at http://www. cs. rice. edu/CS/Systems/Pastry/FreePastry*, 2001.

[15] T. White. *Hadoop: the definitive guide: the definitive guide*. " O'Reilly Media, Inc.", 2009.

[16] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[17] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[18] Simin You, Jianting Zhang, and L Gruenwald. Large-scale spatial join query processing in cloud. In *IEEE CloudDM workshop (To Appear) http://www-cs. ccny. cuny. edu/˜ jzhang/papers/spatial_cc_tr. pdf*, 2015.

[19] M. Zaharia, M. Chowdhury, M. J. Franklin, and S Shenker. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[20] Kristina Chodorow. *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[21] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide*. " O'Reilly Media, Inc.", 2015.

[22] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M Hellerstein, and Scott Shenker. Prefix hash tree: An indexing data structure over distributed hash tables. In *Proceedings of the 23rd ACM symposium on principles of distributed computing*, volume 37, 2004.

[23] Egemen Tanin, Aaron Harwood, and Hanan Samet. Using a distributed quadtree index in peer-to-peer networks. *The VLDB Journal*, 16(2):165–178, 2007.

[24] Cédric Du Mouza, Witold Litwin, and Philippe Rigaux. Large-scale indexing of spatial data in distributed repositories: the sd-rtree. *The VLDB Journal?The International Journal on Very Large Data Bases*, 18(4):933–958, 2009.

[25] Chi Zhang, Arvind Krishnamurthy, and Randolph Y Wang. Skipindex: Towards a scalable peer-to-peer index service for high dimensional data. *Department of Computer Science, Princeton University, New Jersey, USA, Tech. Rep*, pages 703–04, 2004.

[26] Rudyar Cortés, Olivier Marin, Xavier Bonnaire, Luciana Arantes, and Pierre Sens. A scalable architecture for spatio-temporal range queries over big location data. In *14th IEEE International Symposium on Network Computing and Applications-IEEE NCA'15*, 2015.

[27] Jinbao Wang, Sai Wu, Hong Gao, Jianzhong Li, and Beng Chin Ooi. Indexing multi-dimensional data in a cloud system. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 591–602. ACM, 2010.

[28] Rong Zhang, Weining Qian, Aoying Zhou, and Minqi Zhou. An efficient peer-to-peer indexing tree structure for multidimensional data. *Future Generation Computer Systems*, 25(1):77–88, 2009.

[29] Hosagrahar V Jagadish, Beng Chin Ooi, and Quang Hieu Vu. Baton: A balanced tree structure for peer-to-peer networks. In *Proceedings of the 31st international conference on Very large data bases*, pages 661–672. VLDB Endowment, 2005.

[30] Yuzhe Tang, Jianliang Xu, Shuigeng Zhou, and Wang-chien Lee. m-light: indexing multi-dimensional data over dhts. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 191–198. IEEE, 2009.

Characterizing GPS Outages: Geodesic Dead Reckoning Solution for VANETs and ITS

Pedro P. L. L. do Nascimento*†, Richard W. Pazzi†, Daniel L. Guidoni‡, and Leandro A. Villas*
*Institute of Computing, University of Campinas, Brazil
†University of Ontario Institute of Technology, Canada
‡Departament of Computer Science, Federal University of São João Del-Rei, Brazil
Email: liborio@lrc.ic.unicamp.br, richard.pazzi@uoit.ca, guidoni@ufsj.edu.br and leandro@ic.unicamp.br

***Abstract*—Several Intelligent Transportation Systems (ITS) rely on localization to enable services ranging from comfort to safety applications. Following this same idea, the use of Dedicated Short Range Communications (DSRC) devices based on the IEEE 802.11p standard, and the Vehicular Ad Hoc Networks (VANETs) paradigm, have resulted in the deployment of several protocols, services and applications that need different localization accuracy levels. The most common solution for localization is based on Global Navigation Satellite Systems (GNSS). GNSSs have problems of unavailability in dense urban areas, tunnels and multilevel roads. Moreover, GNSS have errors in the range of 5-15 meters. These unavailability and error problems are not acceptable for several VANET and ITS safety applications. In this work, we investigate and characterize the GNSS problems of error and unavailability based on datasets of real GNSS devices installed in vehicles and develop a Geodesic Dead Reckoning solution to overcome the GNSS unavailability issue.**

I. Introduction

The advent of Intelligent Transportation Systems (ITS) has opened to the development of a number of applications and services to enhance the quality of traffic and, consequently, improve the experience of people through intelligent vehicles. In this context, several studies have shown the potential of Dedicated Short Range Communications (DSRC) to establish inter-vehicle communications [1][2][3]. DSRC technology has been standardized as IEEE 802.11p. In this context, several protocols, services and applications have been developed for Vehicular Ad Hoc Networks (VANETs). These applications require different levels of position accuracy in order to provide services ranging from comfort and entertainment to safety and traffic monitoring.

Some of these applications require accurate localization in the range of 1-5 meters, e.g., cooperative cruise control, cooperative safe intersection and vehicle platooning. Similarly, critical safety applications require high accuracy localization in the range of centimetres. For instance, collision avoidance warning systems, lane change warning systems and driver assistance services[1]. These applications control the vehicle for a short period of time and are highly sensitive to delays[1][2].

The primary solution for localization is a GNSS System. The Global Positioning System (GPS). However, in dense urban areas, the reflection of GPS signals on high rise buildings, multilevel roads and overpasses bridges can cause the multipath propagation effect and increase GPS positioning error [4]. Moreover, GPS signals are completely blocked in tunnels caused by occlusion of the line of sight which often affects the provisioning of the GPS localization service.

In this paper we discuss two main contributions: i) The characterization of the GPS unavailability problems and error estimation based on real datasets. Several works only consider a random Gaussian distribution in their analysis and sometimes, these values do not correspond to real scenarios. ii) As a first step towards a complete localization system for VANETS and ITS, we present a Geodesic Dead Reckoning solution and discuss the advantages of the proposed solution to compute vehicle's position.

The related work is presented and discussed in Section II. Section III presents the characterization approach based on real datasets. Section IV details Geodesic Dead Reckoning Solution. The experimental results are discussed in Section V. Finally, the concluding remarks are discussed in Section VI.

978-1-5090-3216-7/16/$31.00

II. Related Work

Localization solutions in VANETS and ITS found in the literature can be divided into two categories: solutions using vehicular network technology to assist GPS, and solutions that do not use GPS. In the first category, the proposed solutions [2][3] merge the GPS information with the vehicle kinematics information and radio ranging techniques as Time of Arrival (TOA), Received Signal Strength Indicator (RSSI) or Round Trip Time (RTT). In this category, solutions often make use of Dead Reckoning in order to track the displacement and direction of the vehicle. These works have reported accuracy in the range of (3-5) meters.

In the second category one can find schemes that do not use GPS in the localization process. They rely on the use of radio technologies and Road Side Units (RSUs) [5] [6] to replace GPS. However, as reported in [2], the high speed of vehicles, network fragmentation, node density and short life-time of links impose several challenges to non-GPS localization approaches. Certain studies [3] [5] [6] have evaluated their solutions only on highway/urban scenarios where the topology of roads/lanes are straight lines, which minimizes the impact of errors inherent in proposed localization solutions. These works have reported accuracy in the range of (5-10) meters.

Nowadays, off-the-shelf GPS receivers are available and widely used in different services and applications. However, localization in VANEts and ITS remains an open challenge when safety applications are considered. Positioning errors in the range of 3-5 meters, as discussed earlier, have a negative impact on several safety applications. Moreover, a number of approaches do not take into account the GPS unavailability problem. Unlike the aforementioned schemes, we use real datasets to characterize the GPS outage problem in highway and urban scenarios. We use paths with different topologies in order to evaluate the impact of error and accuracy in the proposed Geodesic Dead Reckoning solution.

III. GPS Error and Unavailability Characterization

A. Real GPS DataSets and GPS Coordinate System

The Dublin dataset [7] contains GPS data from across Dublin City. This dataset contains the position of 978 buses over a total of 55 days in 2013 (an average of 75,000 positions per vehicle). The San Francisco dataset [8] contains GPS coordinates of approximately 500 taxis collected over 30 days in the San Francisco Bay Area in 2009 (an average of 20,930 positions per vehicle). The Rio de Janeiro dataset [9] contains GPS information of approximately 705 buses collected over 47 days in the Rio de Janeiro city in 2016 (an average of 35,000 positions per vehicle). Differently from the above datasets, it was necessary to develop a script to collect the GPS information in real-time from the online server.

The GPS system uses the World Geodetic System (WGS 84). This geodetic coordinate system representation is based on an approximation of the Earth's geoid by an ellipsoid that rotates around its minor axis. A location in the coordinate system is described in terms of the latitude and longitude, angles are measured according to the equatorial and meridional plan associated with the reference ellipsoid [10].

B. Characterizing GPS Outages

As mentioned previously, GPS systems presents their worst performance in tunnels. The signals provided by the satellites are completely blocked. In order to generate GPS unavailability problems, we perform a characterization from the data-sets described in Subsection III-A.

TABLE I
Tunnels Technical Features

Name	Type	Dist. (m)	Ways	Lanes	Vel. (km/h)	Min T. (s)	Max T.(s)
Rio 450	Urban	1,480	1	3	60.00	44.40	888.00
Dublin Port	Highway	4,600	2	2	80.00	103.50	2070.00
Yerba Buena	Highway	160	2	5	80.00	3.60	72.00

It is noteworthy that several variables affect the time at which the vehicle travels through the tunnel. Among the factors we can highlight traffic congestion and possible traffic lights at the entrance and exit areas of the tunnel. Table I shows recognized tunnel features from digital maps. Generally, the tunnels have two-way roads and it was necessary to establish a mechanism to differentiate the outages that occurs on each way. For this purpose the concept of bounding-box was used to differentiate outages. Given a set of points, one bounding-box is all points inside the considered area. Considering points of latitude and longitude, one bounding-box is defined by a pair of latitude and longitude points: $[(\phi_{min}, \lambda_{min}), (\phi_{max}, \lambda_{max})]$. In order to characterize the outages, we define two areas named Bounding-Box A and Bounding-Box B. This approach, jointly with the timestamp information provided by the GPS datasets, enables the implementation of the following Algorithm 1.

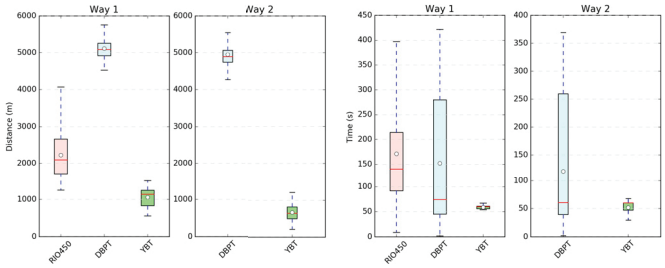
After the definition of the outage sets, we rebuild the vehicles routes by recovering all points inside the bounding-boxes. Figures 1a and 1b show, respectively, the times and distances of outages recognized considering the filter of outages. Considering only the second and third quartiles, a range of more converging values of all outages can be evaluated. For the RIO450, DBPT and YBT tunnels, the outage times for the 50% more representative central values are respectively $[98, 219]$, $[49, 275]$, and $[54, 61]$ seconds. The distance in outage is directly proportional to the time, and time and distance are proportional to the tunnel length.

Algorithm 1: GPS Outages Filter

Input : $boundingBoxA, boundingBoxB, dataSet$

Output: Two sets named $outagesWay1$, $outagesWay2$ with pairs of λ, ϕ points.

$minTime = tunnelDistance/(maxVelTunnel * 2)$;
$maxTime = tunnelDistance/(maxVelTunnel * 0.1)$;
$lengthDataSet \leftarrow length(dataSet)$;
$i \leftarrow 0$;
while $(i < lengthDataSet)$ **do**
  $\lambda_i \leftarrow dataSet[i].\lambda$;
  $\alpha_i \leftarrow dataSet[i].\phi$;
  $\lambda_j \leftarrow dataSet[i+1].\lambda$;
  $\alpha_j \leftarrow dataSet[i+1].\phi$;
  $timeOut = dataSet[i+1].timestamp - dataSet[i].timestamp$;
  if $((minTime < timeOut)$ ***and*** $(timeOut < maxTime))$ **then**
    if $(ValidateAOI(boundingBoxA, boundingBoxB))$ **then**
      $outagesWay1 \leftarrow (\lambda_i, \phi_i, \lambda_j, \phi_j)$;
    end
    else
      if $(ValidateAOI(boundingBoxB, boundingBoxA))$ **then**
        $outagesWay2 \leftarrow (\lambda_i, \phi_i, \lambda_j, \phi_j)$;
      end
    end
  end
end
return $outagesWay1$, $outagesWay2$



(a) GPS Outage Distance. (b) GPS Outage Time.

Fig. 1. GPS Outages.

C. Characterizing GPS Error

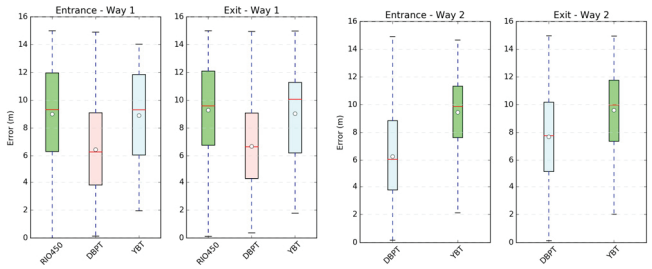
GPS error analysis illustrates the performance of GPS in real scenarios. As discussed earlier, several works have considered the GPS error in the range of 10 - 30 meters and these works consider only a zero mean Gaussian distribution as GPS error. It is important to note that we applied the GPS Outages Filter to remove outliers. We removed values greater than 15 meters from our analysis. Table II shows the total of outages recognized and the total of outages after the removal of outliers. The percentage of outages removed is less than 30 %, justifying the removal of these values. The YBT tunnel is a multilevel tunnel, for this reason the percentage of outliers is too high.

TABLE II
Outages without outliers.

Tunnel	DataSet		GPS error <15 m		Outages Removed (%)	
	way1	way2	way1	way2	way1	way2
RIO450	520	-	436	-	16.15	-
DBPT	960	1435	932	1227	2.91	14.49
YBT	202	391	53	145	73.76	62.91

In order to characterize GPS error from datasets, it is defined as the distance between the GPS point and the perpendicular point contained in the center lane. As shown in Figure 2 the error range of 10 to 30 meters considered by a number of studies [4][1] has been minimized. Even considering recent advances in GPS location estimation[11], the analysed error is still unacceptable for safety applications. In order to overcome

the GPS problems and improve localization accuracy, it is possible to use GPS with contextual information of the vehicle such as displacement, direction, and track information.



(a) GPS Error in Way1. (b) GPS Error in Way2.

Fig. 2. GPS Error.

IV. Dead Reckoning

The Dead Reckoning (DR) localization technique computes the current position of the vehicle based on the vehicle's last location estimation. Two devices provide the necessary information. The odometer is a device that provides information about the traveled distance. A digital compass is an electronic device built with magnetometers, that gives the angle (azimuth) of the vehicle in relation to the Earth's magnetic true north [1][4].

A. Classical Dead Reckoning

The classical Dead Reckoning Solution uses trigonometric functions in a local coordinate system. In this local coordinate system, the classical DR solution is given by $x_k = x_0 + \sum_{i=0}^{k-1} s_i \cos\theta_i$ and $y_k = y_0 + \sum_{i=0}^{k-1} s_i sin\theta_i$.

Here, (x_0, y_0) is the initial vehicle location at time t_0 and s_i, θ_i at time t_i are, respectively, the shortest path and the absolute heading of vehicle in relation to the earlier position (x_{i-1}, y_{i-1}) at time t_{i-1}. The relative heading (bearing) is defined as the difference between absolute headings at two consecutive instances and is denoted by δ_i. Given relative heading measurements δ_i in the range of times $t_1, t_2, ..., t_k$, the absolute heading θ_k of the vehicle at time t_k is computed by $\theta_k = \sum_{i=0}^{k} \delta_i$. The Figure 3 illustrates this approach.

B. Geodesic Dead Reckoning Solution

In Geodesy, the problem of defining the shortest path between two points on the Earth's surface is referred as *geodesic*. Solving a geodesic implies in solving the ellipsoidal triangle NAB. Here, N is the North Pole. NAF and NBH are meridians, and AB is a geodesic length s_{12}. The latitudes and longitudes of A and B are, respectively, ϕ_1, ϕ_2 and λ_1, λ_2. The longitude of B relative to A is $\lambda_{12} = \lambda_2 - \lambda_1$. EFH is the equator with E laying on the extension of the geodesic path AB; and α_0, α_1, and α_2 are the azimuths of the geodesic at E, A, and B [12]. There are two methods to achieve a solution to this problem: *Geodesic Direct* and *Geodesic Inverse*. The Geodesic Direct receive as input ϕ_1, α_1, s_{12} and aims to compute ϕ_2, λ_2 and α_2. The Geodesic Inverse receive ϕ_1, λ_1, ϕ_2, λ_2 aims to determine $s_{12}, \alpha_1, \alpha_2$, in this case $\lambda_{12} = \lambda_2 - \lambda_1$.

Recent advances in geodesy have solved some of the problems, allowing the computation of any geodesic in the Earth's surface. Before Karney's contributions [12], the best method to compute a geodesic was the Vincenty's method. However, Vincenty's method fails for some cases of geodesics

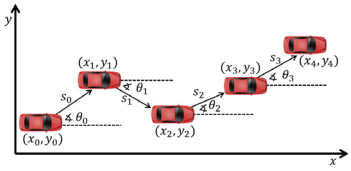


Fig. 3. Classical DR Solution.

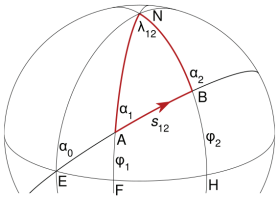


Fig. 4. The ellipsoidal triangle NAB [12].

[12][10]. Based on the aforementioned methods of geodesy, and the knowledge about coordinate systems, we propose a DR solution named Geodesic Dead Reckoning (GDR) to overcome the problem of GPS unavailability. Assuming that the set of GPS positions is given by $(\phi_0, \lambda_0), (\phi_1, \lambda_1)$, ..., (ϕ_i, λ_i), and ϕ_i, λ_i is the last know GPS position before a GPS outage, the Algorithm 2 is proposed.

Algorithm 2: Geodesic Dead Reckoning Algorithm

Input : Estimated GPS position ϕ_i,λ_i, the true north azimuth α_i and the traveled distance s_{ij}.

Output: Estimated GPS + GDR position: ϕ_j,λ_j

1 $gpsOutage \leftarrow GPSIsUnavailabe()$;
2 **while** $gpsOutage$ **do**
3 $\quad \alpha_i, s_{ij} \leftarrow VehicleKinematics()$;
$\quad \phi_j, \lambda_j \leftarrow GeodesicDirect(\phi_i, \lambda_i, \alpha_i, s_{ij})$;
4 **end**

The line 1 of the algorithm determines when the GPS is in outage stage. The loop in lines $2-4$ are executed while the GPS is unavailable. α_i is the azimuth in relation to the last know position and the true north s_{ij} is the traveled distance by the vehicle in a reading cycle. The last know position ϕ_i, λ_i and α_i, s_{ij} are parameters of the Geodesic Direct method. The new estimated position is given by ϕ_j, λ_j.

V. Results

A. Simulation Setup

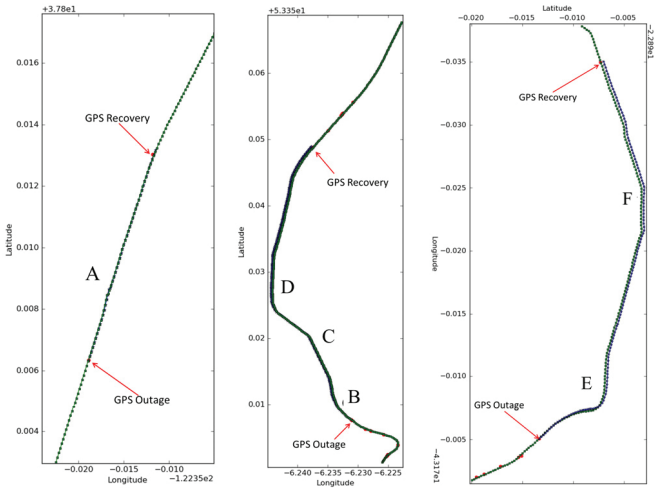
Our simulation experiments use SUMO version 0.25 in order to simulate mobility models using realistic traffic conditions. The Krauß Model has been configured with 50% of drivers imperfection [13]. For each outage recorded in the dataset, we generate a vehicle in the simulation environment. The map data with the information about the regions (number of lanes of a road or street, traffic lights, maximum speed, way direction, etc.) and the tunnels features were obtained from Open Street Maps [14]. All simulation results were performed using a T Student Distribution with confidence interval of 95%.

The implementation of our GDR solution was conducted using a set of scripts in Python3 and the library GeographicLib [12] that provides the geodesy methods described in Section IV-B. To estimate the error accumulation of the GDR, we evaluate the Error Per Cycle (EPC). Since the coordinates are in degrees, the distance between the true vehicle position and the estimated GDR is given by $EPC = GeodesicInverse(\phi_v, \lambda_v, \phi_{GDR}, \lambda_{GDR})$. Here (ϕ_v, λ_v) and $(\phi_{GDR}, \lambda_{GDR})$ are, respectively, the coordinates of the vehicle position and the estimated GDR in each reading cycle. In the last GDR read before GPS recovery, the EPC provides the amount of error accumulated in the GDR process.

B. DR Trajectory Impact and Error Analisys

As discussed in Section II, several existing works [3] [5] [6] use highways or straight streets. Consequently it ends up

favoring the performance of the proposed location solutions. In order to highlight this issue, we present the following analysis. In Figure 5, we present the topologies of streets, highways and tunnels used in the simulations.



(a) YBT (b) DPT (c) RIO450

Fig. 5. Geodesic Dead Reckoning Trajectories.

The green and blue traces represent respectively vehicles and GDR trajectories. It is important to note that Rio450 and DPT tunnels are curvilinear. Moreover, these scenarios contain sharp curves. On the other hand, YBT tunnel have smoother curves. We analyzed the GDR EPC exactly at the time which the vehicles are crossing the curves. In order to evaluate the GDR solution, we perform an aggregated analysis. We have marked the points A, B, C, D, E, F in the Figures 5 and 6 to show the effect of the tunnel path in the GDR tracking and, consequently, in the EPC.

At the point A (YBT tunnel), the slope in EPC occurs because the transposition between the vehicle path and the GDR path, the error reaches 7.35 meters in 47 seconds. At points B, C, D of the tunnel DPT, we can observe several curves with different shapes and directions. At the point D, there are a transposition between the vehicle position and GDR path causing the same effect observed at the point A. The points B and D are acute curves causing an increase in the error. In RIO450 tunnel, the effect of the curves is even more drastic. Only in 60 seconds the error doubles, jumping from 20 to 40 meters considering points E and F. This is due to the high angular variation between the point of outage and the point of recovery in a short time period.

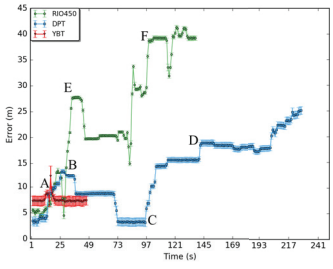


Fig. 6. Error per Cycle (EPC)

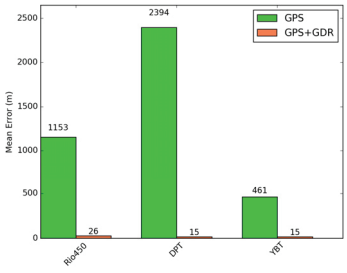


Fig. 7. GPS Overall Error.

Figure 7 shows the comparison of the mean error between standalone GPS and GPS+GDR solutions. In a system that only uses the GPS solution, it tends to keep the same position during the outage despite the vehicle being moving along the tunnel. For this reason, the error becomes quadratic while GPS unavailability time increases. However, GPS + GDR solution has a linear error increase over time.

VI. Conclusion and Future Works

This paper presented an evaluation of GPS error and outage problems from real datasets. We characterized time and distance that vehicles travel through areas of GPS unavailability such as tunnels, and contextualized the requirements of applications that rely on localization systems. From the considered datasets, we estimated GPS errors using digital maps. We have proposed a solution named Geodesic Dead Reckoning that uses the techniques and concepts of geodesy in order to overcome GPS error and unavailability problems. We have also performed evaluation experiments to assess the impact of errors in different scenarios on the proposed GDR scheme. In addition, an analysis has been conducted in order to show the impact of the unavailability of the stand alone GPS solution when compared to the proposed GDR solution. For future work, we will investigate a cooperative positioning solution using IEEE 802.11p technology to overcome the GDR limitations.

Acknowledgments

The authors would like to thank Sao Paulo Research Foundation (FAPESP) (grant 2015/07538-1), CNPq (agreement no. 132244/2016-0) and FAPEMIG (486332/2013-6) for the financial support. This work is also partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), Discovery Grant Program.

References

[1] A. Boukerche, H. A. Oliveira, E. F. Nakamura, and A. A. Loureiro, "Vehicular Ad Hoc Networks: A New Challenge for Localization-Based Systems," *Computer Comm.*, vol. 31, no. 12, pp. 2838–2849, 2008.

[2] N. Alam and A. G. Dempster, "Cooperative Positioning for Vehicular Networks: Facts and Future," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 4, pp. 1708–1717, dec 2013.

[3] N. M. Drawil and O. Basir, "Intervehicle-Communication-Assisted Localization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 678–691, sep 2010.

[4] I. Skog and P. Handel, "In-Car Positioning and Navigation Technologies - A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 1, pp. 4–21, mar 2009.

[5] A. A. Wahab, A. Khattab, and Y. a. Fahmy, "Two-way TOA with limited dead reckoning for GPS-free vehicle localization using single RSU," *13th Int. Conf. on ITS Telecommunications, ITST*, pp. 244–249, 2013.

[6] L. Sun, Y. Wu, J. Xu, and Y. Xu, "An RSU-assisted localization method in non-GPS highway traffic with dead reckoning and V2R communications," *2nd Int. Conf. on Consumer Electronics, Communications and Networks, CECNet*, pp. 149–152, 2012.

[7] "Dublin Bus DataSet," 2013. [Online]. Available: https://data.dublinked.ie/dataset/dublin-bus-gps-sample-data-from-dublin-city-council-insight-project

[8] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "A parsimonious model of mobile partitioned networks with clustering," in *2009 First International Communication Systems and Networks and Workshops*. IEEE, jan 2009, pp. 1–10.

[9] "Data Rio - BUS GPS," 2016. [Online]. Available: http://data.rio/dataset/gps-de-onibus

[10] J. M. Zogg, *GPS - Essentials of Satellite Navigation*. UBLOX, 2007.

[11] "International GNSS Service: Strategic Plan 2013-2016," Tech. Rep., 2012. [Online]. Available: http://kb.igs.org/hc/en-us/article_attachments/200543517/IGS_Strategic_Plan_2013.pdf

[12] C. F. F. Karney, "Algorithms for geodesics," *Journal of Geodesy*, vol. 87, no. 1, pp. 43–55, 2013.

[13] "Simulation of Urban MObility." [Online]. Available: http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

[14] "Open Street Maps." [Online]. Available: www.openstreetmap.org

Contextual Geotracking Service of Incident Markers in Disaster Search-and-Rescue Operations

Ev Cheng*[1], Kourtney Meiss*[2], Kendall Park[3], John Gillis[3], Dave Weber[4], Salman Ahmad[3], Prasad Calyam[3]
[1]*Department of Computer Science, Vassar College, NY*; [2]*Department of Computer Science, Wofford College, SC*
[3]*Department of Computer Science, University of Missouri-Columbia, MO*; [4]*Missouri Task Force 1*
amycheng@vassar.edu; meisskn@email.wofford.edu; {kfp3x5, ahmadsa}@health.missouri.edu;
{gillisj, calyamp}@missouri.edu; *dweber@allstateconsultants.net*

***Abstract*—Real-time geovisualization of disaster scenes provides visual situational awareness, which could decrease medical triage time, and also allows first responders to better allocate relief resources. In this paper, we describe a novel contextual geotracking service that provides spatiotemporal visualization of response history through the use of mobile devices and a wireless mesh network. During crisis response, with limited resources in a high-stress disaster relief environment, contextual data visualization of disaster incident scene status markers, and their presentation in a usable dashboard is crucial. We present novel visualization tools that we have developed to integrate custom map markers, tracking information collected through a wireless network, geovisualization over time through gradients, and spatiotemporal event filters. We evaluate our geotracking service in a field trial with a search-and-rescue task force comprising of professional first responders. We show effectiveness of our service in terms of data entry time, usability survey, and qualitative feedback within a disaster response simulation experiment.**

I. INTRODUCTION

Lack of usable technology due to the destruction of infrastructure during natural disasters can prevent first responders from responding as quickly and efficiently as possible. Lack of infrastructure significantly limits what types of technology can be implemented and standardized for national emergency response teams. Currently, responders are sent into the field with hand-held devices to record data for future analysis. This lack of real-time information contributes to an incoherent overview of the scene for incident commanders (ICs), which consequently leads to difficulties in filtering accurate and updated information, allocating resources, and prioritizing e.g., patient triage in medical relief efforts.

In our previous work, we have developed the Panacea's Cloud [1] that aims to serve as a real-time communication and coordination tool designed to provide situational awareness to incident commanders and responders in disaster scenarios. Panacea's Cloud uses an ad hoc network that is independent of existing infrastructure, such as an 802.11 wireless network or radio towers, which may be disrupted during a disaster. It enables first responders such as, firefighters and emergency paramedics, to work together to most effectively survey a scenario and provide treatment. Decreased triage time ultimately allows for efficient allocation of resources and could save lives.

In this paper, we extend our Panacea's Cloud by developing a novel contextual geotracking service of incident markers in disaster search-and-rescue Operations. To this end, we work collaboratively with Missouri Task Force 1 (MO-TF1), an organization of first-responders, based in 28 locations around the United States, who specialize in search-and-rescue and medical triage. Technological limitations are often encountered by MO-TF1 after natural disasters, such as Hurricane Katrina, and correspond closely to the problem Panacea's Cloud seeks to resolve.

Currently, MO-TF1's state-of-the-art technology is limited to hand-held GPS devices with a custom symbol set. These GPS devices do not allow for real-time data routing and are dependent on a central computer for synchronization. Immediately upon return from a search, responders must both upload and download data through a USB connection. A new search team is immediately dispatched after the return of another, which does not allow enough time for data analysis and integration. Therefore, no new information can be distributed until the *second* dispatch team returns and the *third* is dispatched. After synchronization, which is dependent on the number of responders, the data is analyzed through custom macros in Microsoft Excel to gain situational awareness [2].

In the absence of GPS devices, traditional handset radios and paper triage tags are used. However, communication can be disrupted (e.g., background noise during a disaster situation that interferes with radio units), and consequently, handset radios can be unreliable in quickly and accurately conveying and receiving information. Similarly, paper triage tags do not allow for real-time location tracking and information updating (e.g., the movement of patients over time or the number of patients classified per triage level). Existing works such as DIORAMA [3], under development by the University of Massachusetts Amherst, aim to solve the problem of medical triage during mass casualty incidents through active RFID readers and tags that transmit information, and support tools for IC communication and patient tracking user interfaces. However, they rely on infrastructures of cell and radio towers, and do not leverage ad hoc wireless networking at disaster incident scenes.

This material is based upon work supported by the National Science Foundation under Award Number: CNS-1359125 and Coulter Foundation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or Coulter Foundation.

978-1-5090-3216-7/16/$31.00

To address the above challenges, we enhance the real-time component of Panacea's Cloud for the processing and display of spatiotemporal information that is essential for ICs to gain an accurate depiction of the past and present events to best allocate resources and effectively respond. Due to the dynamic movement of responders in the field, our solution helps an IC to visually analyze coverage and organize relevant information through filters. Furthermore, to decrease cognitive burden associated with visual analysis, we develop novel spatiotemporal information visualization on the dashboard in an intuitive manner and in a way that requires minimal temporal processing and interaction with the interface.

The remainder of the paper is organized as follows: Section II outlines our geotracking and visualization objectives and novel visualization tools that we have developed. Section III presents experimental methods and results with MO-TF1. Section IV concludes the paper.

II. CONTEXTUAL GEOTRACKING AND VISUALIZATION

A. DESIGN OBJECTIVES

In the case of crisis response in a limited-resource and high-stress healthcare environment, data visualization and dashboard usability are crucial. Given the dynamic nature and high-stakes involved in mass causality incidents, the IC must be able to quickly interpret the available data of a scenario at any given moment and then use the dashboard effectively to allocate appropriate resources. Prior experience with similar systems can mitigate a user's error rates [4]. However, the assumption cannot be made that such experience exists. Assuming the user has no prior experience, it is essential for a dashboard to minimize cognitive burden and facilitate information processing. In this paper, we seek to address methods and tools in which Panacea's Cloud's user interface gives a spatiotemporal visualization of the patient story with limited visual-cognitive demand.

B. VISUALIZATION TOOLS

The following interface tools were implemented using JSX, which adds XML-syntax to Javascript; React, a Javascript library that renders UI components based on data change; and Redux, a state container changed through dispatched actions. These tools were chosen specifically for their efficient comparisons and re-rendering of current and incoming data. Visual component libraries that we used include: Leaflet, Leaflet plugins, Bootstrap, and React-adapted Bootstrap UI components.

1) Custom Markers: A common method of spatial indication on maps is the use of markers. However, for a disaster response interface, generic markers are severely limited in their ability to convey information as seen in Figure 2. Generic markers lack distinguishable features which contributes to users' inability to differentiate between markers. In addition, generic markers typically do not display meta-information that is vital in determining further action. Finally, generic markers are not interactive and do not provide an option for actionable updates such as updating the status of a marker. Due to these limitations, generic markers are not suitable for Panacea Cloud's dashboard.

Fig. 1. Generic markers lack sufficient information for disaster response

Custom markers are added to the dashboard using a combination of Javascript libraries as shown in Figure 2. The different markers include features, such as icons and colors, which allows users to differentiate between them. To suit any user's particular needs, the marker icon, color, and description are customizable through a centralized JSON configuration file. The freedom to edit, add, or remove the custom markers confers an element of extensibility to Panacea Cloud's dashboard, eases transition from previous systems, and lessens the cognitive demand in interpreting data. To expand on the visual information provided by the markers, a pop-up of plain text and/or pictures is available through user interaction to allow for visual geotagging of static points (e.g., a road block) and dynamic points (e.g., a resource in motion).

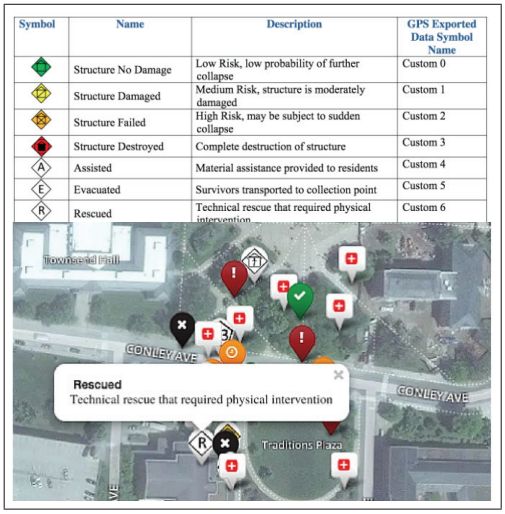


Fig. 2. Custom markers are easily distinguishable and display a pop-up with information from MO-TF1's legend

In our MO-TF1 use case, twenty-four custom markers, currently used on the MO-TF1 GPS systems, are configured into a JSON file. When clicked, the name and description of the custom symbol is displayed as shown in Figure 2. To differentiate MO-TF1 responders from incident markers, a different icon is used. The responder markers can also display information in pop-ups.

2) Tracking Information: A significant deficit of markers is the lack of time integration. Markers handle the indication of spatial location well, but can only display current location without reference to past locations. In the case of disaster response, the loss of past information is highly critical. Thus, creating tracks between markers to visualize past locations is a means of adding a temporal component to maps.

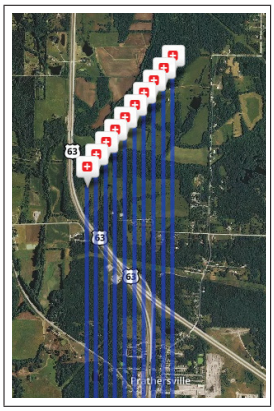


Fig. 3. Solid paths behind responder markers indicate past spatial locations

As shown in Figure 3, paths visualize a survey history in an area and differentiate incoming data, including but not limited to responder-to-search-area ratios, responder search efficiency, specific device failure, total areas searched and needing to be searched. Incident markers can be directly linked to a responder, allowing for tests of accuracy (e.g., a marker an IC may expect to see during training and the actual marker entered). A progression of the scene can then be traced as an expanding coverage of an area of disaster. The logging of the data allows for future analysis of composite search-and-rescue data as well as future analysis of specific responder data. The former can lead to improved search-and-rescue methods that minimizes loss of lives, and the latter can lead to improved training methods.

We used a model to relate the rendered components to each responder or custom marker, which in turn refers to a geotag's type ID for filtering. Each responder, hardware device, and custom marker has a unique ID. This differentiates between responder and device and creates flexibility for a case where a responder may have to use a different device, due to issues such as hardware. This also allows for easy testing of devices, as device failure while tracking can be viewed on the dashboard in real-time. Within the custom marker, the responder unique ID is also recorded, which connects the responder to their markers. On the dashboard itself, each responder has their own path, marker, and pop-up, while each marker has its own marker and pop-up.

3) Geovisualization Over Time Through Gradients: The use of paths behind markers, to indicate past locations of responders and incidents, is a geovisualization over a temporal frame. However, in the implementation of these paths, specificity of information is lost, and relative times cannot be determined. For example, a marker indicates a responder's current location at a *specific* time, while a path indicates a responder's past location at a *nonspecific* time.

One proposed solution is a dynamic timeline playback. However, due to its nature, this has a high temporal cost, forcing a user to view a dynamic image over a period of time in order to determine the location of markers at a specific time within the timeline. In the case of disaster response, this is precious time that could be better allocated. To create a static image that both displays specific spatial and temporal information, gradient paths are implemented. Figure 4 displays the mapping of a gradient path to a timeline key with timestamp labels. This allows a user to process a single static image and extract spatial and temporal information about a scenario without continuous playback. Thus, a user can use the color of a gradient to determine the spatial record of resources and responders at a specific time or within a time frame.

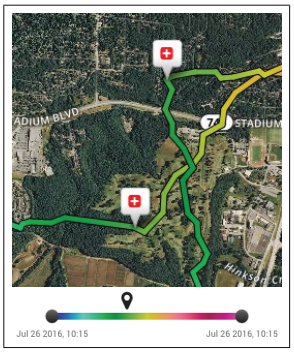


Fig. 4. Close-up of gradient paths and the corresponding gradient key with labeled timestamps generated with randomized data

The use of gradients to map information confers all the advantages that a solid-colored line does, including information related to specific responders and progression of coverage. In this case, gradients provide a clearer view of temporal progression as well as an overall view. Regarding real-time data upload and display, as in the Panacea's Cloud dashboard, the use of gradients visually organizes incoming information.

One criticism of gradients is the reliance on color and a lack of accessiblity to colorblind users. To address this issue, a colorblind palette shown in Figure 5 is used to address the two common forms of colorblindness: protanomaly and deuteranomaly. Colorblindness is estimated to effect 8% of men and 0.5% of women worldwide. Of total colorblind population, 75% of colorblind men have either protanomaly or deuteranomaly [5]. Consequently, a palette adjusted for red-green colorblind users increases accessibility. The option to configure with specific hex triplets is also available through a JSON file. Like with the custom markers, the centralized characteristic of configuration files allows this dashboard to extend to multiple use cases beyond search-and-rescue.

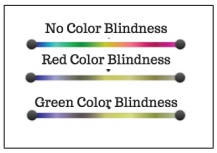


Fig. 5. Slider with the path gradient as perceived by users without color-blindness and users with red-green colorblindness

The implementation of gradients for the Panacea's Cloud dashboard was received positively by MO-TF1, especially given that gradients are not a feature available on their current system. The usability, accessibility, and immediate applicability of the gradients to search-and-rescue operations makes it a vital component to the Panacea Cloud's dashboard for extracting spatial and temporal information without time resource allocation.

4) Spatiotemporal and Event Filter: In the case of vast amounts of incoming data or an incident with multiple events (e.g., an earthquake with multiple aftershocks), analysis of an event rather than the incident may be preferable. Accomplishing this necessitates the ability to select a time frame in which data can be analyzed. By changing the display view, a spatiotemporal filter decreases the visual-cognitive burden of the IC by removing distracting or currently irrelevant information and allows for more accurate analysis.

To focus on an event in an incident, a spatiotemporal filter is available on the Panacea's Cloud dashboard. A user is able to select the timespan for the desired view. By default, if the whole view is selected or a view including the most recent timestamp is selected, incoming data is always included. This removes the task of re-adjusting the slider constantly. Dragging the display slider through the filtered times changes the view dynamically. An example is shown in Figure 6. A play/pause button is also available for users who prefer to play back an event at a consistent rate, starting at either a specific time within the filter or at the start of the filter; the button allows for user interruption on the detection of change events within the slider. The filter determines the gradient paths displayed, while the marker above the filter indicates how far along the filtered path that the resource has traveled. The display therefore shows past, present, and future progression, and allows for greater precision in the playback of scenes.

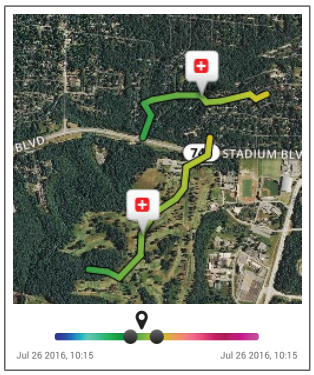


Fig. 6. Filtered gradient paths and corresponding filtered gradient key

For further data organization, a reset button is placed on the dashboard to separate incidents. Upon user interaction, the reset button makes a call to the API for a new incident, which clears the current data from the dashboard without changing its storage within the database. This enables the IC to access and display cleared data later as a separate incident. To keep users from accidentally double-clicking and sending two incident requests, the button is disabled during fetching.

The separation of data into multiple incidents and events within an incident filters the data in a user-controlled way. This creates dashboard-displayed information that is more usable; gradient paths and markers become more effective through filtering. Additionally, incident creation and incident-event filtering makes Panacea's Cloud's dashboard extensible and applicable to other use cases, such as device tracking or more efficient ambulatory aid.

III. PERFORMANCE EVALUATION

A. EXPERIMENTAL METHODOLOGY

To compare MO-TF1's current Garmin GPSMAP 64 system to Panacea's Cloud, we ran three trials: Trial One tested Panacea's Cloud with Recon Jets, Trial Two tested MO-TF1's current hand-held Garmin system, and Trial Three tested Panacea's Cloud with a Mobile View on Android devices.

Standard training procedure for MO-TF1 involves the placement of laminated placards symbolizing "incidents" with characteristics corresponding to the custom MO-TF1 markers (e.g., number of victims, detection of human remains, destroyed structure, etc.) onto wooden stakes along a road to simulate a neighborhood for search-and-rescue. The placards are traversed by responders, and the characteristics are entered as quickly and accurately as possible.

Trial One was performed a week before Trials Two and Three and without MO-TF1 participants present. The preliminary results of the trial determined that Recon Jets, do not provide suitable hardware for the MO-TF1's use case, which most likely can be contributed to the lack of a competitive wearable technology market at this time. Transmission of data though the Recon Jets relied on an internal camera, to photograph QR codes corresponding to the custom markers, which produced overexposed images in both high and low contrast environments. The internal GPS displayed high rates of GPS scatter and inaccuracies which greatly impacted the coherence of an incident. The user interface was unusable for data entry and data verification due to the application closing at inappropriate times and its inability to confirm the data was received by the incident commander.

Thus, to suitably and more rigorously test Panacea's Cloud, a mobile application was created for data entry; Figure 7 displays the screen as seen by users. The application was accessed locally on Android devices, given to MO-TF1 participants in Trial Two, with instructions on application use. Our focus in this paper will be on the results of Trials Two and Three.

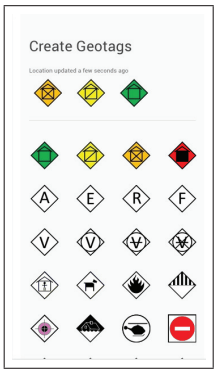


Fig. 7. Panacea's Cloud's Mobile View as seen by interface users

For Trials Two and Three, twenty-nine incidents, marked as events within a single incident on Panacea's Cloud's dashboard, were placed along a designated road in the training area. In both trials, two MO-TF1 participants entered all characteristics of each incident and verbally indicated the start of data entry and the end of data entry for each incident. The travel time from one marker to the next was recorded

but filtered out. For the Garmin system, a separate time was recorded for the upload of data from the handheld devices through the USB. For Panacea's Cloud, the time between the last marker recorded and its appearance on the dashboard was taken as a rough indicator of update time.

To determine MO-TF1's subjective perception of the current Garmin system and Panacea's Cloud's Mobile View for data entry, the participants completed a usability survey containing 10 questions, rated on a Likert scale of 1 to 5.

- I think that I would like to use this system frequently.
- I found this system unnecessarily complex.
- I thought the system was easy to use.
- I think that I would need the support of a technical person to be able to use this system.
- I found the functions in this system well integrated.
- I thought there was inconsistency in this system.
- I would imagine that most people would learn to use this system quickly.
- I found this system very cumbersome to use.
- I felt very confident using this system.
- I need to learn a lot of things before I could get going with this system.

Additionally, the same usability survey was completed by the acting IC for both Iron Sights, the current Garmin dashboard [2], and Panacea's Cloud dashboard. A qualitative analysis was also conducted to gather feedback regarding suggested improvements to Panacea's Cloud dashboard.

B. EXPERIMENTAL RESULTS

As shown in Figure 8, the average data entry time per incident for Panacea's Cloud was 3.6x faster than the data entry time per incident for the Garmin system. For the current Garmin system, the total time taken by the participants in entering data for the 29 incidents was 850 seconds (≈14.2 minutes) and 1193 seconds (≈19.9 minutes). This averages to 35.2 seconds per incident. For Panacea's Cloud, the total time taken for the same 29 incidents was 329 seconds (≈5.5 minutes) and 242 seconds (≈4.0 minutes). This averages to 9.8 seconds per incident.

Fig. 8. Total time taken by each user to enter 29 incidents for each tested

In terms of overall usability as shown in Figure 9, Panacea's Cloud's Mobile View and dashboard scored the same or better, as rated by both participants and the incident commander, in 9 out of the 10 categories. The participants Panacea's Cloud was easy to use and learn, a characteristic especially important in disaster scenarios where users may not have the time to undergo extensive training before entering the field. One notable difference is the Garmin system's higher score in confidence of use. This outcome was not surprising because the Garmin is the system they are comfortable and currently using.

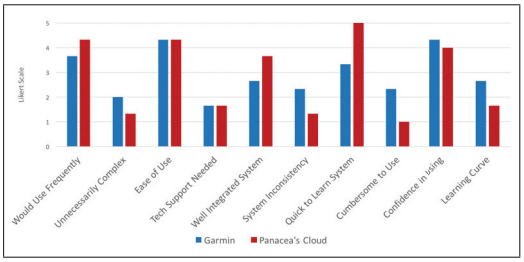


Fig. 9. Usability questionnaire results indicate Panacea's Cloud outperforms the current Garmin GPS system in 9 out of the 10 categories

Qualitative analysis of the data reveals that gradients were received positively by the IC. Suggestions for the dashboard included an ability to enter a bulk number of markers, a tally for each marker, data verification for the user that a marker is successfully sent, and dates of incidents under a separate tab on the dashboard.

IV. CONCLUSION AND FUTURE WORK

Our experimental trials display a 3.6x decrease in data entry time using the Panacea's Cloud contextual geotracking service instead of the handheld Garmin system. This decrease of time in sending and retrieving data is indicative of a reduction in time of search-and-rescue missions as well decreased triage time. We found that the Garmin system upload time is dependent on the number of users, while Panacea's Cloud acts in real-time data display on the dashboard. This creates a scalability for Panacea's Cloud that does not apply to the current Garmin system used in search-and-rescue efforts of groups such as MO-TF1. The usability survey results and positive feedback also indicate greater system usability. Further trials with a larger sample size and greater control of variables are needed for more conclusive results.

Outside of search-and-rescue operations, the Panacea's Cloud dashboard is extensible to other use cases where geovisual information must be displayed and filtered over a timespan, such as medical triage, device tracking, or environment monitoring. Because of these many use cases, future work on Panacea's Cloud dashboard is focused on increasing usability and adding to the available features.

REFERENCES

[1] J. Gillis, P. Calyam, A. Bartels, M. Popescu, S. Barnes, J. Doty, D. Higbee, S. Ahmad, "Panacea's Glass: Mobile Cloud Framework for Communication in Mass Casualty Disaster Triage", *Proc. of IEEE Mobile Cloud*, pp. 128-134, 2015.

[2] F. Endrikat, "US&R Program Directive 2014-013 - Search Operation Data Collection and Reporting Standards", *National Urban Search & Rescue Force Representatives Report*, 2014.

[3] A. Ganz, J. Schafer, Z. Yang, J. Yi, G.Lord, G. Ciottone, "Mobile DIORAMA-II: Infrastructure less Information Collection System for Mass Casualty Incidents", *Proc. of IEEE EMBC*, pp. 2682-2685, 2014.

[4] H. Tuzun, E. Telli, A. Alir, "Usability testing of a 3D touch screen kiosk system for way-finding", *Computers in Human Behavior*, Vol. 61, pp. 73-79, 2016.

[5] "Types of Colour Blindness", in Colour Blind Awareness. [Online]. Available: http://www.colourblindawareness.org/colour-blindness/types-of-colour-blindness. [Accessed: Jul. 25, 2016].

Using NAS Parallel Benchmarks to Evaluate HPC Performance in Clouds

Thiago Kenji Okada
and Alfredo Goldman
Instituto de Matemática e Estatística (IME)
Universidade de São Paulo (USP)
São Paulo – SP, Brazil – 05508-090
Email: {thiagoko,gold}@ime.usp.br

Gerson Geraldo H. Cavalheiro
Centro de Desenvolvimento Tecnológico – Computação
Universidade Federal de Pelotas (UFPel)
Pelotas – RS, Brazil – 96010-610
Email: gerson.cavalheiro@inf.ufpel.edu.br

***Abstract*—Cloud computing is a reality nowadays, however there are few studies trying to understand what happens in the actual cloud infrastructures for HPC applications. The focus of this study is the evaluation of NAS Parallel Benchmarks on cloud computing environments. We analyze the execution of applications from NAS Parallel Benchmarks (LU and SP), comparing the execution behavior in different infrastructures: a public cloud, a private cloud and a NUMA multiprocessor system. Our broad goal is to estimate the performance of actual HPC applications on cloud, based on its communication characteristics. We conclude that HPC users should be careful with Virtual Machines with higher virtual CPU count, thanks to Google usage of Hyper-Threading technology and Virtual Machine instances scheduling.**

***Keywords*—cloud computing, benchmark, NAS Parallel Benchmarks**

I. Introduction

With the advent of cloud computing, it is not necessary anymore to invest large amounts of money on computing resources for private use. Instead, it is possible to obtain processing or storage resources, and even complete systems, on demand, using one of the several available services from cloud providers like Amazon EC2, Microsoft Azure and Google Compute Engine.

In public cloud environments, the user has minimal knowledge about the infrastructure of hardware. However, knowledge of the network topology can be used to improve performance in HPC applications [1]. Moreover this information is not available for cloud computing users, HPC users might have non-optimal performance when executing applications in public clouds. This may incur an extra cost for the user, since cloud computing is paid per-usage. There also might be an important performance loss.

In this work, we analyze the behavior of NAS Parallel Benchmarks (NPB) in clouds, including Google Compute Engine (GCE), a public cloud, and a private cloud implemented with OpenStack. We compare the performance of NPB in GCE and OpenStack with a NUMA multiprocessor system, to evaluate the performance of multiple NPB benchmarks on different systems. Our broad goal is to provide estimates on HPC applications performance in clouds, based on its characteristics. In this paper, we present and analyze the results of two different NPB applications, LU and SP.

978-1-5090-3216-7/16/$31.00

The remainder of this paper is structured as follows: in Section II, we review the literature about the area. In Section III, we describe the NAS Parallel Benchmarks. In Section IV, we describe how we did our experiments and methodology used to analyse the results. In Section V, we present the results of the experiments, including our analysis about the results. Finally, in Section VI, we present the conclusions and future work.

II. Related work

Ma *et al.* [2] developed an approach for matching communication patterns in scientific parallel applications. They analyze four main properties, namely, *temporal, spatial, volume* and *communication graphs*. After that, they applied a *rank transformation* on them, allowing to compare similarities and contrast differences between applications, independent of size. They tested their approach using four applications from NAS Parallel Benchmarks version 3.0: BT, SP, MG and LU. After applying their methodology on these benchmarks, the authors show that these problems have similar communication patterns independent from input size (A, B or C), and even on the number of processors (they document tests with 16 and 64 *logical process*).

One limitation from [2] is that it only considers two MPI operations, `MPI_Send` and `MPI_Recv`, to obtain θ and ϵ, although the benchmarks include calls to collective communication primitives. However in [3], Faraj *et al.* analyzed the communication characteristics of the MPI implementation of NPB, to study the effectiveness of *compiled communication optimization* for MPI programs. Their tests show that *collective* communications in NPB (that includes `MPI_Allreduce`, `MPI_Bcast`, `MPI_Barrier`, etc.) are just a small part of the communication in NPB applications, while *point-to-point* communications (`MPI_Send`/`MPI_Isend` and `MPI_Recv`/`MPI_Irecv`) are the bulk of the communication in NPB, validating the results from [2] for the whole benchmark.

Evaluation of HPC applications in cloud environments is a recurring theme in the literature, so different authors tested the feasibility of running HPC application in clouds [4]–[7]. They generally analyse metrics such as network bandwidth/latency

and/or processing speed. However, our approach is different: it is based on the previous knowledge from Ma *et al.*, Faraj *et al.* [2], [3] of communication attributes from NPB applications, and it focus in the relation between two communication approaches used in HPC applications, instead of raw performance. Those two communication approaches are: **intra-node**, that is, communication between process in the same node (in our case, a virtual machine), via either multithreading or message passing; and **inter-nodes**, that is, communication between process in different nodes (VMs), via message passing.

III. NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) are a small set of programs designed to help evaluate the performance of parallel supercomputers, developed by NASA. These benchmarks are derived from computational fluid dynamics (CFD) applications. They consist of five parallel kernels and three simulated application benchmarks.In this work we used two of the three simulated application benchmarks, LU and SP.

We chose LU and SP benchmarks from NPB, because both had a MPI *Single-Zone* version (called **SZ** from now) and MPI+OpenMP *Multi-Zone* version (called **MZ** from now). It is important for this work to analyze both SZ and MZ versions, since this allows evaluation of both intra-node and inter-node communication performance. BT also has both SZ and MZ implementations, however BT is similar to SP benchmark considering communication patterns [2]. Thanks to this, BT analysis was omitted from this work for lack of space.

In this work we focused on three different parameters to evaluate what is happening during the benchmarks: *total execution time*, *total communication volume* and the *number of communication calls*. The *total execution time* is the execution time measured by the application itself, since each NPB application reports its execution time after finishing execution. The *total communication volume* is measured by running one benchmark instance on 32 containers with 1 CPU each in our NUMA system, and running `sysstat`[1] in the background, to measure the bandwidth used in the network interface. The *number of communication calls* is the number of communication function calls from MPI in each benchmark, measured by TAU[2], a portable profiling and tracing toolkit for parallel programs.

Both *total communication volume* and the *number of communication calls* are constant for the same application configuration. It means that, for example, running a LU benchmark in class D with 32 logical processors will always result the same volume and number of communication calls, independent of the machine used to run those benchmarks. However, *total execution time* should be different on each virtual machine configuration. So in Table I, we show *total volume communication* and *number of communication calls*, since they are constants.

IV. Methodology of the Experiments

LU-SZ benchmark uses MPI, and in our experiments LU-SZ is executed with 32 MPI logical processors. SP-SZ benchmark uses MPI too, however SP-SZ needs a square number of logical processors to run, so we run it with 25 MPI logical processors. LU-MZ and SP-MZ versions are executed with 1 MPI logical processor for each VM (e.g.: if we have 4 VMs we use 4 logical processors). The number of OpenMP threads used (set by `OMP_NUM_THREADS` environmental variable) correspond to the number of vCPUs in each VM (e.g.: if we have VMs with 8 vCPUs we use 8 threads for each MPI process). We limited the number of vCPUs to 32 since this is the size of the biggest available VM in our case of study (Google Compute Engine).

TABLE I. Total volume communication and number of communication calls in each NPB benchmarks.

	Total Volume Communication (GB)	Number of Communication Calls
lu.D.32	121.73	12,741,100
sp.D.25	407.60	375,900

We modified the number of VMs and the VM size in each experiment, always maintaining 32 vCPUs. So, if we use only one VM, we use a 32 vCPUs VM; if we use two VMs we use 16 vCPUs in each one, and so on. The following VMs sizes are used in the experiments executed in GCE: $1\times$*n1-highcpu-32*, $2\times$*n1-highmem-16*, $4\times$*n1-highmem-8*, $8\times$*n1-highmem-4*, $16\times$*n1-highmem-2*.

The change from *n1-highcpu* to *n1-highmem* is because NPB needs a *master* process with a relatively higher memory compared to the other nodes. So with smaller VMs, the benchmarks started to fail thanks to the lack of memory. However, this should not impact this evaluation, considering that in terms of CPU power all VMs should be equivalent. According to GCE dashboard all VMs used Intel® Haswell CPUs.

For the private OpenStack cluster we had access to an OpenStack Liberty setup, called *revoada*. The host system is running Ubuntu 14.04 (trusty), Linux kernel 3.19, KVM virtualization. There are 7 physical nodes in *revoada*, connected in a star topology in a 1Gbps Ethernet network. Each node have $2\times$Intel® Xeon® CPU E5645 @ 2.40GHz and 8x4GB (32GB in total) of RAM.

Finally, our NUMA system is composed of $4\times$AMD Opteron™ 6276 @ 1,4GHz, called *hydra*. Each CPU has 16 cores (resulting in 64 cores in total) and each CPU has 32GB of RAM directly connected to it, however the total memory is available to all processors with different memory latency access. This results in a total of 128GB available to the operational system. The operational system is Ubuntu 14.04 (trusty), running Linux kernel 3.13 and the software LXC 1.0.8. LXC is a container system, used to limit the number of CPUs in each experiment and simulate separate VMs.

All VMs and containers are running Ubuntu 14.04, GCC 4.8.4 and OpenMPI 1.6.5.

V. Experiment and Results

The experiments are executed in *GCE*, *hydra* and *revoada* infrastructures. For the experiments with large sizes (class D), we only did five executions for each setting. This was motivated by two main reasons. On one hand, for the experiments with smaller sizes, we did a careful statistical analysis

[1] http://sebastien.godard.pagesperso-orange.fr/

[2] https://www.cs.uoregon.edu/research/tau/home.php

in the experiment, analyzing the variation in execution for 50 repetitions. In those experiments, we always got results concentrated near the average, with small standard deviations, as exemplified in Figure 1. This shows that the experiments results are stable. On the other hand, each execution of class D size was very time consuming (around 1 hour for each run).

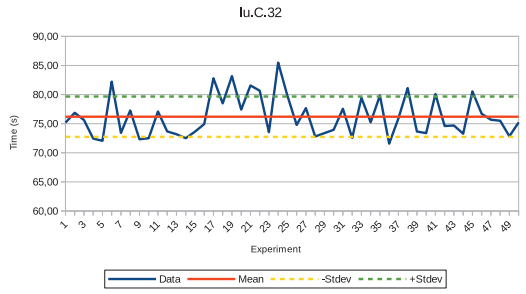


Fig. 1. Graph of execution times between 50 repetitions in LU-SZ running in *hydra*, including mean and standard deviation.

For the graphs in the next subsections, the vertical axis is the execution time in seconds (s), while the horizontal axis is the different infrastructures tested. *GCE* represents Google Compute Engine, *hydra* represents our NUMA architecture, *revoada* represents our private OpenStack cloud; **C** represents the number of vCPUs in each VM, while **V** represents the number of VMs used in the experiment. So, GCE-2**C**x16**V** represents the execution time in Google Compute Engine, using 16 VMs with 2 vCPUs each, GCE-4**C**x8**V** uses 4 vCPUs and 8 VMs and so on. Notice that, for any x**C**$\times y$**V**, we always have $x \times y = 32$ vCPUs. To allow reproducibility, the scripts used in this work are available under an open-source license (MIT), in GitHub[3].

We highlight that we are not comparing the difference between execution times in different infrastructures. Our actual objective is to analyse the difference in behavior between different architectures.

A. LU

Figure 2 shows the results of LU-SZ benchmark, which exploits intra-node parallelism using MPI. A first observation, it stands out that the execution time shows a low variation, except in GCE-32Cx1V and revoada-16Cx2V, which are the lowest number of possible VMs in each cloud infrastructure used. Our hypothesis is that in those cases, LU-SZ is affected by Intel® Hyper-Threading technology, so in both cases there are not enough physical cores to support the number of requested vCPUs per VM.

Our analysis of the results in *hydra* allows us to conclude that in a infrastructure with low communication costs, there is no significant variance in performance (the biggest difference between means is ± 188.6 seconds). Thanks to this, we conclude that GCE-16Cx2V are probably running in different physical machines, otherwise the performance would be better (i.e. similar to the finer-grained cases).

When there is a non-negligible cost of communication between VMs in physical network (like in *revoada*), using finer-grained VMs allows the VM scheduler to allocate some VMs in the same machine, reducing communication costs and, consequently, reducing the execution time. This is the case in both revoada-2Cx16V and revoada-4Cx8V, and likewise, GCE-2Cx16V, GCE-4Cx8V and GCE-8Cx4V.

We can conclude that the scheduling of fine-grained VMs, in other words, VMs with low vCPU count, is more efficiently made in this infrastructure.

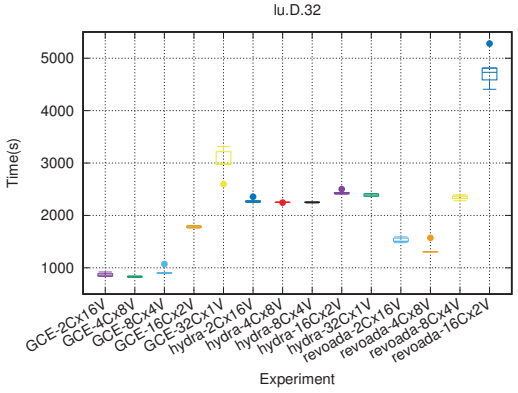


Fig. 2. Boxplot of LU-SZ total execution time in our tested infrastructures.

Figure 3 shows the results of LU-MZ benchmark. In those Multi-Zone benchmarks it is except, at least partially, that communication related costs are overlapped by effectively processing when using multithread [8], since threads blocks while accessing critical region, while other threads occupy the processor evolving the calculation.

With coarse-grained VMs, the number of communication channels between VMs is small. So the competition between threads may occur during the access to the MPI communication channel. In 2Cx16V cases, we have 1 MPI communication channel shared between 16 threads in each VM. When we decrease VM granularity, by reducing the number of vCPUs and increasing the number of VMs, we have more MPI communication channels to be explored by each thread. This reduces the competition in each MPI channel, consequently increasing the total performance. Looking at *GCE* results, 4 VMs seems to be the minimum number to avoid MPI channel congestion.

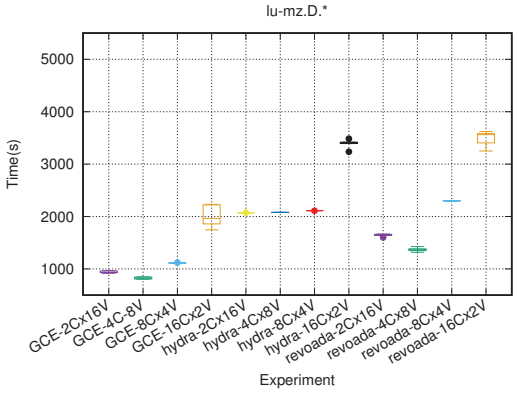


Fig. 3. Boxplot of LU-MZ total execution time in our tested infrastructures.

[3]https://github.com/m45t3r/naspb-tests

B. SP

Figure 4 shows the result of SP-SZ benchmark. SP-SZ have a superior total execution time in general compared to LU. The *total volume communication*, as shown in Table I, is likewise superior, however there is a smaller *number of communication calls*. Likely happened to LU-SZ, we had low variation in execution times in *hydra* infrastructure (the biggest difference between means is ± 212.15 seconds). And we observe in SP-SZ too, the coarse-grained and the impact of Hyper-Threading in GCE-32Cx1V, and the Hyper-Threading and communication impact in revoada-16Cx2V. There is a high variation in GCE-16Cx2V, that may be caused by the load balancing in *GCE*, causing a migration of the VMs used during the experiment. In *GCE* cases with finer-grained VMs, that is 8 vCPUs or less, it is possible to observe the scheduling of VMs in the same physical node or physical nodes nearby.

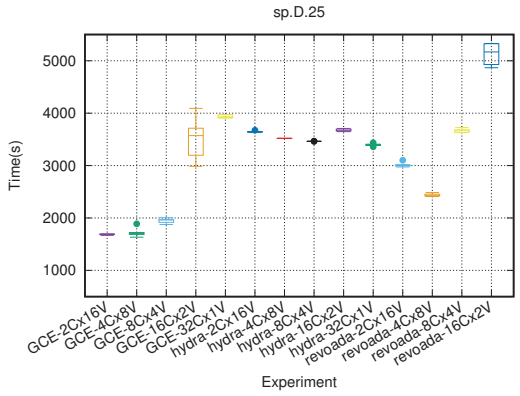


Fig. 4. Boxplot of SP-SZ total execution time in our tested infrastructures.

Figure 5 shows the results of SP-MZ benchmark. The highest execution times are observed with a smaller number of VMs, similar to LU-MZ. Similar to LU-MZ too, we found that communication costs do not affect performance in configurations with fewer number of vCPUs.

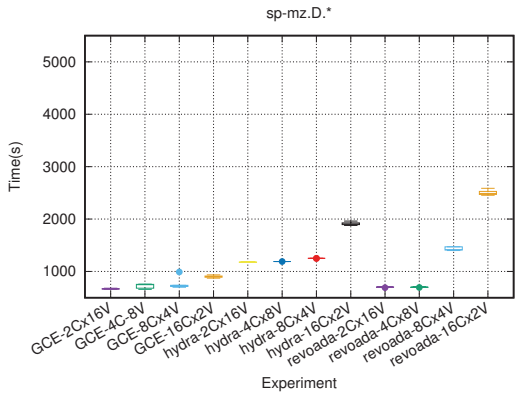


Fig. 5. Boxplot of SP-MZ total execution time in our tested infrastructures.

VI. Conclusion and Future Work

In this work we show the impact of intra-node and inter-node communication in different infrastructures. The results show that the performance of HPC application can be affected, positively, using the appropriate number of vCPUs in each VM used during the execution of the application. Coarse-grained VMs was penalized, thanks to the cloud's VM scheduler and Hyper-Threading technology. Considering those factors, it is possible to optimize performance of HPC applications in clouds.

Our analysis is focused on Google Compute Engine. GCE is a valuable public cloud to do our experiments since the pricing of utilization in GCE is charged by minute, after the first 10 minutes. Since the VMs are billed per minute, a reduction in *total execution time* may result in important savings in total cost. For example, in sp.D.25, we can cut up to 2300 seconds ($\sim$38 minutes) in execution time, comparing GCE-2Cx16V with GCE-32Cx1V. In terms of money savings, it represents a savings of \$1.31 per run[4].

In the future we will apply the same analysis to other public clouds, like Amazon EC2 and Microsoft Azure, and compare them to Google Compute Engine.

Acknowledgment

This work was supported by CAPES/Brasil (*Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal do Nível Superior*).

And we would also like to thank Google for the academic credits to access their public clouds.

References

[1] M. Diener, E. H. M. Cruz, M. A. Z. Alves, M. S. Alhakeem, P. O. A. Navaux, and H.-U. Heiß, "Locality and balance for communication-aware thread mapping in multicore systems," in *21st Int. Conf. on Parallel and Distributed Computing, Proc.* L. J. Träff, S. Hunold, and F. Versaci, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 196–208.

[2] C. Ma, Y. M. Teo, V. March, N. Xiongy, I. R. Pop, Y. X. He, and S. See, "An approach for matching communication patterns in parallel applications," *Proc. of the 2009 IEEE Int. Parallel and Distributed Processing Symp.*, no. December 2008, 2009.

[3] A. Faraj and X. Yuan, "Communication characteristics in the NAS parallel benchmarks," *Proc. of the PDCS '02*, no. January 2002, pp. 729–734, 2002.

[4] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," in *Proc. of the 19th ACM Int. Symp. on High Performance Distributed Computing - HPDC '10*, New York, New York, USA: ACM Press, Jun. 2010, p. 395.

[5] A. Gupta and D. Milojicic, "Evaluation of HPC applications on cloud," *Proc. - 2011 6th Open Cirrus Summit, OCS 2011*, pp. 22–26, 2012.

[6] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas, "Performance evaluation of Amazon EC2 for NASA HPC applications," in *Proc. of the 3rd workshop on Scientific Cloud Computing Date*, ACM, 2012, pp. 41–50.

[7] Z. Li, L. OBrien, R. Ranjan, and M. Zhang, "Early observations on performance of Google Compute Engine for scientific computing," in *2013 IEEE 5th Int. Conf. on Cloud Computing Technology and Science*, IEEE, vol. 1, 2013, pp. 1–8.

[8] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990.

[4] https://cloud.google.com/products/calculator/#id=fcac81db-0018-41f8-ac3b-b41619157ef8

A User Level Approach to Schedule BoT Applications on Private Clouds

Maicon Ança dos Santos André R. Du Bois Gerson Geraldo H. Cavalheiro
Graduate Program in Computer Science, UFPel, Brazil
Email: {madsantos,dubois,gerson.cavalheiro}@inf.ufpel.edu.br

***Abstract*—This paper presents a user level approach to schedule tasks generated by bag of tasks (BoT) applications on a private cloud. At this level, the scheduler consolidates the load of the tasks in a given number of virtual machines providing the estimated makespan. We present the proposed algorithm as well as a model for BoT applications and a performance assessment in a OpenStack based IaaS infrastructure. The results show that the makespan can be reduced by grouping tasks in coarse units of loads.**

I. Introduction

Bag of Tasks (BoT) is a very popular pattern of applications in cloud computing. This class of application exploits close to 3/4 of total processing capacity provided by cloud infrastructures [1]. A BoT defines a set of independent tasks requiring processing resources to be completed. The amount of resources required by each task can differ as well as the number of tasks in the *bag* during the execution. In this paper we consider a scenario where a client of a private cloud infrastructure must identify the processing power requirements to run a BoT application. This client is supposed to know the application to predict the evolution of the BoT during time. We propose an application level scheduler to distribute the load of the tasks over a limited number of processors furnishing the expected execution time. In this scenario, each processor represents a virtual machine (VM) owned by the client and responsible for executing the load it accumulates. The set of VMs is submitted to run the application in a private cloud infrastructure. Since the number of host in the infrastructure can be lesser than the number of VMs, the problem is to identify how far the effective execution time was from the expected one.

The main contribution of this paper is the *task consolidation* strategy, achieved at application level, responsible for distributing the load of tasks among VMs. We propose a general online strategy to consolidate tasks in coarser workload units. Different policies specialize this general strategy taking into account different task attributes, such as arrival time, computational cost and remaining time to finish, to ordering ready tasks in a priority list.

The remaining of this paper is structured as follows. In Section II we present some related work, beholding those considering the execution of BoTs in clouds and cloud scheduling strategies. Section III presents the abstraction we propose to describe BoTs. The task consolidation strategy is presented in Section IV. Section VI presents the a case study conducted on a OpenStack based infrastructure. Section VII concludes the paper and presents the final remarks.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

II. Related Work

Different traces of grids usage are analyzed by Iosup and Epema in [1]. They have observed that tasks generated by BoT applications consume near to 75% of total available CPU time, represent 75% of total load submission, and frequently this kind of applications are responsible for over 90% of total CPU time consumption.

In [2], the scheduling problem is to maximize the usage of available processing resources to execute BoT tasks in a heterogeneous cloud infrastructure. In this work, the authors propose a scheduling framework composed by 14 heuristics. This framework implements the heuristics in a two phase scheduling: in the first phase tasks are ordered prior to execution; in the second, tasks are mapped to available resources. In this schema, tasks have two attributes, one is the arrival time, and the other is the number of instructions (used to compute the expected execution time). The results of this work were validate by simulation.

In [3] we found another scheduling strategy for heterogeneous clouds taking into account the resources cost. The scheduling goal is minimizing completion time of an application while respecting an upper bound for the budget to be spent by the user. The scheduling operates dynamically while the applications evolve. To make scheduling decisions, the strategy is to profile the execution of tasks to infer their computational costs. The paper documents a performance assessment achieved in a local cloud infrastructure managed by the Ibis platform [4].

In [5] another scheduling strategy for heterogeneous GRIDs is presented. No information about the infrastructure, is used. The authors propose three policies for composing resource offers to schedule deadline constrained BoT applications. Among other conclusions, the authors state that offer-based scheduling produces less delay for jobs that cannot meet deadlines in comparison to scheduling based on load availability, and that more jobs can meet deadlines if the total computing power is known.

Another important work is presented by Iosup an his group ([1], [6]). After analyzing a large number of traces of executions on grids, they propose a workload model for BoT applications. Some components of this model are:

- A large number of users is able to submit BoTs. The probability of a given user submitting a new BoT is determined by a Zipf distribution.
- The inter-arrival time between consecutive BoT arrivals in a provider is described by a Weibull distribution.
- The number of tasks (BoT size) is also described by Weibull distribution.
- The computational requirements of tasks in a BoT can be described also by a Weibull distribution of probability.
- The runtime required to finish each task (task length) in a BoT better fit a Normal distribution.
- The variability of tasks lengths in a BoT can be described by a Weibull distribution.

We consider that cloud schedulers can provide better result if the user is able to schedule himself his BoT in a bounded number of processing resources. This hypothesis is based in the conclusion presented in [7]. Different schedulers were analyzed in this work and authors state that the makespan can be reduced by regrouping tasks in coarser execution units due to the high degree of unpredictability with respect to resource availability of cloud environments.

Our approach to prove this hypothesis is experimental. We propose a high level abstraction to describe BoT applications and a basic framework to regrouping tasks in virtual machines (VMs). A BoT description is translated to a random BoT application with the proprieties identified by Iosup and group. The VMs represent the coarser units of work. The prototype was validated in a cloud infrastructure managed by OpenStack.

III. The BoT Abstract Application

A BoT is described by a set A of n 4-tuples $A = \{q_1, \ldots, q_n\}$. Each 4-tuple $q_i = [a_i, d_i, b_i, c_i]$ defines a set of b_i identical tasks ready at the time a_i. The duration and the processing cost of those tasks are giving by d_i and c_i. In our model, the time evolves discretely. Thus, the attributes related to time, a e d, are natural positive numbers. The cost c is informed as a percentage of utilization of a CPU. We consider a homogeneous architecture.

A. Step-by-step execution

Time evolves discreetly, step-by-step. The attributes a and d define the granularity of tasks in a BoT. A task τ_j^i belonging to q_i can be seen, in consequence, as a sequence $\{w_0, \ldots, w_{d_i-1}\}$ of jobs, each job having a computational cost c_i. Since each job represents a time step at execution time, the following rules are valid:

1) The $w_1 \in \tau^i$ is ready to execute at time a_i;
2) Given two jobs w_{k-1} and w_k belonging to a τ^i, with $1 \leq k < d_i$, the precedence order $w_{k-1} \prec w_k$ must be respected.

The number of jobs in a BoT A is the sum of the number of time steps required to execute each tasks defined by A.

IV. Task Consolidation

Task consolidation is a scheduling effort accomplished by the user to map tasks described by a BoT in a finite number of processing resources. Different scheduling policies can be obtained combining information about tasks attributes (arrival time, duration, computational cost) and the ongoing execution step (to compute the amount of completed or remaining number of jobs of each task).

A. Task decomposition

Although tasks in a BoT are independent, a given task is able to create new tasks on the same BoT while it executes. Consider two tasks τ_k^i and τ_l^j from groups q_i and q_j where $a_i < a_j$ and $\delta = a_j - a_i$ is true. Since tasks have no individual identification, it is valid to consider that any task in q_i creates the group q_j at the end of the job $w_\delta^{\tau^i}$. In this way, the tasks in the group q_j cannot be pushed into the bag before all tasks τ^i finish the step w_δ, even in a situation where there is a limited number of processing resources. To prevent incorrect execution order, we introduce a *global step* as an external clock to synchronize the creation times.

A job $w_j^{\tau_i^k} \in \tau_i^k$ is described by $w_j^{\tau_i^k} = [g_j^i, r_j^i, f_j^i, c_j^i]$. At this level of decomposition, the abstraction of tasks is no more necessary. In this 4-tuple g_j^i represents the global step when the job becomes ready. The job execution cost $c_j^i = c_i$. The number of jobs remaining and finished in the original task jobs sequence are represented by r_j^i and f_j^i, respectively. Thus, we have $b_i = r_j^i + f_j^i$ and $g_j^i = a_i + f_j^i$. For a given task τ_i, a sequence of d_i jobs $\tau_i = \{w_1^i, w_2^i, \ldots w_{d_i}^i\}$ is instantiated. To proceed the task consolidation, the *job* is the unit of work considered, taking into account that jobs with the same global step are independent and those belonging to a same task must be executed in the strict order of their global steps.

B. Processing contention

A cloud infrastructure M defines a limited amount of processing resources. We consider an homogeneous cluster with unbounded memory capacity and no delays for communication with processors. We also consider that the scheduling operations introduce no overhead at execution time. Thus, only the BoTs are the resource consumer in the system. M is composed by a set of m identical processors $M = \{p_1, p_2, \ldots p_m\}$. In a given global step s, this infrastructure provides $|C^M| = m \times 100\%$ processing capacity. The processing requirement L_s^A of a BoT A^n in a given step s is the sum of the processing requirements of all ready jobs. Processing contention is observed when $L_s^A > |C|$. In this case, a set of ready jobs must be choose to be executed at the step s and the others delayed. Since the load of job is atomic, *i.e.*, cannot be split, the effective load processed in s is $|L_s^A| \leq |C^M|$.

The set of delayed jobs will be executed, at least, at the global step $s + 1$. Since there are creation precedence constraints, in the form $w_\delta^{\tau^i} \longmapsto q_j$, $\forall q_i$ and $q_j \in A$, when a job is delayed, creation of new tasks must be delayed as a consequence.

C. Normalizing the global step

As shown previously, the attribute a_i of a 4-tuple q_i represents the arrival time in an unbounded architecture of

all tasks defined by q_i. This time a_i represents also the amount of processing already finished by a given BoT. If processing contention is occurs, all remaining jobs will suffer the consequences of the delay.

The processing contention results on the preemption of some executing tasks, pushing them back to the bag. In this case, the global step of all jobs in the bag, belonging to preempted or ready tasks, is incremented in one step. The normalization is required to preserve both the temporal order between two jobs $w_j^i, w_{j+1}^i \in \tau_i$ and the creation dependency between $w_j^i \longmapsto q_k, \forall i < k \leq n, \in A^n$ when $g_j^i + 1 = a_k$.

V. User Level Scheduling

The user level scheduling provides the mapping of jobs produced by a BoT over a cloud infrastructure with limited number of resources.

A. Basic operation

The user level scheduler has as input a bag description and the number of available processors. This scheduler is correct if all dependencies between jobs are respected and it must to guarantee that a ready job will be waiting if there is at least one processor with enough processing capacity available. The scheduler runs at the end of each global step and produces the load distribution to the following global step. The following operations may be executed:

Job selection: Only ready jobs (*i.e.*, $g_i = g$), are eligible. A job is selected from a list, this list may be ordered by a priority criteria.

Processor selection: Selects a processor to execute the selected job. Since the scheduler operates on a homogeneous architecture and there are no communication costs and memory limits, this selection takes into account the amount of processing power available on each processor.

Task preemption: A task may be preempted when new tasks are pushed into the bag and there is not enough processing power available. Due to some priority criteria, or even by random choice, a running task can have the execution of its jobs delayed. Jobs cannot be preempted, since they execute atomically in one time step.

Task migration: Since there is no communication cost, task migration occurs without time constraints. As jobs cannot be preempted, they cannot be migrated.

We define two map policies combining job and processor selection operations: batch and cyclic. Both policies have in common the input (the set of processors and a list with ready jobs) and the output (a list of jobs for each processor). If the ready list is not empty at the end of a mapping, the remaining jobs are pushed back to the bag and a normalization of time steps is triggered. When a mapping processing starts, the processing capacity of each processor is assumed to be 100% (the processors are idle).

The batch policy first selects a processor and then move through the list of jobs while the load mapped to the processor selected is below 100%. If the processing cost of the job in the head of the list does not exceeds the current capacity of the processor, the job is removed from the list and its processing cost added to the processor load. Otherwise, the first job is skipped and the same check is repeated for the second job in the list and so on. The mapping process for a processor ends when it attains 100% of processing capacity or when there are no more jobs to be considered. The process for all processors ends when all processors where selected or when the ready queue becomes empty.

The cyclic policy maps a job to a processor at once. Processors are numbered from 1 to m. The mapping process visits all processors cyclically and selects, starting from the head of the ready list, a job that does not exceed 100% of the current processor processing capacity. The process ends when a cycle was completed without any placement or when the ready list becomes empty.

B. The list scheduling

The model of BoTs allows to build a list of jobs statically. This list is, first, ordered by the global step of each job, considering a contention free architecture, then, the jobs belonging to the same global step are sorted by a priority criteria.

$$A^{m'} = \mathbb{S}' < F > (A, m')$$

The scheduler is a function $\mathbb{S}'$ that maps the tasks of an application described by a list of jobs A over the m' processors of a homogeneous parallel machine. The scheduler $\mathbb{S}'$ applies interactively a consolidation algorithm until it finishes the mapping of all jobs in A. Each iteration represents a global step evolved. The result of each iteration is the mapping of jobs in a given global step, removing placed jobs from the list and updating the load of each processor.

C. Defining a priority criteria

A job in our model is described by a 4-tuple $w_i = [g_i, r_i, f_i, c_i]$. The g_i attribute, the global step, defines a temporal order among jobs. The consolidation algorithm assures that this temporal order is respected. Thus, the priority criteria must be applied in jobs in the same global step. Any attribute r_i, f_i e c_i, or attributes combination, can be used to define a priority order.

The sorting algorithm has as input a list of jobs and provides as output the same list, ordered by a function F. This function can implement different criteria, e.g., the computational cost of a job, the arrival time of the tasks (the older a task is, the higher is the priority of its jobs), etc.

D. Scheduling result

The application level scheduling, named task consolidation algorithm, implements an online list strategy statically. It provides the computational load of each available processor at each global step. It is assumed there is no overhead at execution time, $e.g.$ scheduling overheads and communication latencies, and that a processor executes its load independently from the others. $|S^A|$ represents the expected length, in steps, of the execution.

Table I
PRODUCTION BOTS CASE STUDY (TIME IN MINUTES).

BoT	# tasks	(L_μ,L_σ)	Arrival	# VMs	$\lvert S^{UP-*}\rvert$	Finish	$\lvert S^{UP-*}\rvert$-Finish (%)
UP-1	98	(50,10)	0	10	355	394	10.99%
UP-2	41	(50,10)	65	8	240	262	9.17%
UP-3	56	(10,2)	110	10	215	225	4.65%
UP-4	23	(10,2)	165	8	225	231	2.67%
UP-5	10	(50,10)	200	4	245	252	2.86%
UP-6	39	(50,10)	230	8	345	359	4.06%
UP-7	8	(10,2)	270	4	275	297	8%
UP-8	26	(25,5)	360	8	460	472	2.61%
UP-9	16	(50,10)	400	4	480	489	1.86%
UP-10	10	(25,5)	530	4	570	576	1.05%

VI. PERFORMANCE ASSESSMENT

To illustrate a performance assessment of our scheduling schema, we investigate the behavior at execution time of ten independent BoTs (UP-1 to UP-10) submitted to our private OpenStack (Kilo release) based cloud infrastructure with ten nodes (quadcore, 16 Gb, Gigabit Ethernet). These BoTs have different attributes, such as arrival time, number of tasks, processing loads and ideal number of VMs. Table I provides some information about the (randomly) generated BoTs as well as the performance obtained. For example, UP-1 begins the experiment at minute 0 and its $|S^{UP-1}|$ is 355, but it was effectively finished at minute 394. The load of the tasks of each BoT was generated by a normal distribution ([1]) were the average and the standard deviation was given by L_μ and L_σ. In this table, $|S^{UP-10}|$ and the **Finish** time correspond also to the expected and effective time to finish all executions.

In our experiment, we start OpenStack with the default parameters and each new VM launched is placed employing the Filter strategy. Then, during the execution, every 5 minutes (i.e, the time required to execute a global step in our experiments) a naïve system scheduling strategy detects all nodes overloaded (CPU occupation above 80%) then migrates VMs from these nodes in order to balance the load.

Even considering that the UP BoTs have not saturated the infrastructure, the **Difference %** between the expected and the real time execution decreases with the number of active VMs. We ascribe this behavior to the load balancing among nodes provided by the system scheduler. In average we have observed 136 VM migrations on each run of this set of BoTs. To support this assumption we have submitted all UP-* to be consolidated as a single BoT with 16, 20 and 36 VMs. The $|S^{big}|$ obtained was 770, 640 and 490 minutes and the effective running time was 883, 733 and 553 minutes, respectively, representing roughly a difference of 12% on all cases.

VII. CONCLUSION

Bag-of-tasks is a very common pattern of parallelism in cloud computing and represents the main source of load in these infrastructures. In this paper, we proposed an application level scheduling, named task consolidation, to distribute the load generated by a BoT in a bounded number of virtual machines. A performance assessment in a real cloud infrastructure is also presented. Next steps will include a simulation tool to evaluate the task consolidation strategy in a large number of scenarios. Another contribution of this paper is the BoT description model.

In this work we were able to show that the effect of unpredictability with respect to resource allocation in a cloud infrastructure can be constrict by a tasks grouping policy, as expected by [7].

ACKNOWLEDGEMENT

This work was supported by CAPES/Brasil (Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

REFERENCES

[1] A. Iosup and D. Epema, "Grid computing workloads," *IEEE Internet Computing*, vol. 15, no. 2, pp. 19–26, 2011.

[2] J. O. Gutierrez-Garcia and K. M. Sim, "A family of heuristics for agent-based elastic cloud bag-of-tasks concurrent scheduling," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1682–1699, Sep. 2013.

[3] A.-M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in *CLOUDCOM 2010*, 2010, pp. 351–359.

[4] N. Palmer, T. van Kessel, R. Kemp, N. Drost, R. V. van Nieuwpoort, J. Maassen, H. E. Bal, G. Wrzesinska, T. Kielmann, K. van Reeuwijk, F. J. Seinstra, C. J. H. Jacobs, and K. Verstoep, "Real-world distributed computer with ibis," *Computer*, vol. 43, no. 8, pp. 54–62, 2010.

[5] M. A. S. Netto and R. Buyya, "Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems," in *IPDPS 2009*, 2009, pp. 1–11.

[6] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, "The performance of bags-of-tasks in large-scale distributed systems," *Proc. of HPDC 2008*, p. 97, 2008.

[7] M. Gokilavani, S. Selvi, and Udhayakumar, "A survey on resource allocation and task scheduling algorithms in cloud environment," *IJEIT*, vol. 3, no. 4, Oct. 2013.

Heterogeneous Resource Allocation in Cloud Management

Serdar Kadioglu, Mike Colena, Samir Sebbah
Advanced Constraint Technology
Oracle Corporation, Burlington, MA 01803, USA
{firstname.lastname@oracle.com}

***Abstract*—This paper introduces a combinatorial problem arising from real-world business requirements as part of resource allocation in Cloud Management. In particular, we focus on the allocation of a set of heterogeneous resources serving multiple tenants with different service level agreements. There exist certain business rules that govern the application stemming from privacy, performance, and capacity requirements. We show how to formulate the problem as constrained optimization and then solve it efficiently using Artificial Intelligence based constraint propagation. Our approach stands out as a high-level, declarative solution that is efficient and easy to maintain and update.**

I. Introduction

In the Cloud Computing era, the number of applications that requires continuous processing of high-throughput data is ever growing: consider, for example, financial market analysis, internet traffic, various IoT applications, or Big Data analytics [1]. Varying service level agreements and different applications characteristics make static cloud resource allocation inefficient. This calls for dynamic, easy-to-modify and maintain resource allocation strategies.

The particular resource allocation problem studied in this paper emerges as a part of cloud management and is aimed at managing physical resources such as CPU cores and disk space. These resources must be sliced and shared between virtual machines running potentially heterogeneous workloads. The ultimate goal is to provide support for the indexing capability of search platforms, such as the Apache Solr [2] in our case; a popular open-source search engine designed to index and search multiple sites in near real-time. Apache Solr, and other similar engines, continuously generate ever growing quantities of heterogeneous data. The issue is that, this unbounded data flow must be handled effectively and efficiently using limited resources, respecting various business rules. This is exactly the problem we studied.

Due to their intrinsic complexity, such problems are frequently dealt with custom-designed heuristics. The issue with heuristic solutions is that they are often rigid, hard to extend, and even harder to maintain as requirements evolve, which frequently happens in cloud environments. To circumvent that, we propose an Artificial Intelligence (AI) based constraint formulation to tackle resource allocation. The main contribution of this paper is a high-level, declarative AI approach that is easy to update and maintain, and performs well in practice as demonstrated in our experiments.

978-1-5090-3216-7/16/$31.00

II. Problem Description

Consider a setting where we are given a cluster of host machines. These are physical systems that host one or more Solr *nodes*, i.e., Java Virtual Machines (JVMs). The machines can be heterogeneous, e.g., with different specifications from storage to process capacity, bandwidth, etc. Each Solr node (JVM) can host one or more *cores*, which are used to index the incoming data. A logical set of one or more cores represents a *core group*. In this context, a group refers to a related set of data records, and within a single group, the records are distributed/balanced among multiple cores. This is similar to the concept of *sharding* in terms of database architecture. The system is aimed at serving a set of tenants that own one or more core groups. The tenants can be of different types (e.g., trial, standard or enterprise customers) which limits the number of cores that they are entitled to in core groups. Similarly, the tenant type also defines the application growth limit (e.g., trial customers are allowed up to one core group increment for storage per week). The goal is to assign all the cores to JVMs achieving a balanced system overall. While each particular application deployed in the cloud has its own peculiarities and might be subject to other specifications, the following rules capture the main structure of our problem.

I. Each tenant should claim separate JVMs.
II. Cores of each core group should be placed on different JVMs but on the same host machine.
III. The storage capacity of machines and JVMs should be respected.
IV. The maximum load among all JVMs should be minimized.

We refer to this problem as the Core Group Placement Problem (CGPP).

Definition 1 (CGPP instance) *An instance I of the* CGPP *can be defined as I (M, T, numJVM, CapacityM, CapacityJ, numGroups, numCores), where M is the number of host machines, T is the number of tenants to serve. Then, for each machine* $m \in M$, $numJVM_m$ *denotes the number of JVMs in m,* $CapacityM_m$ *denotes the maximum number of cores allowed in m,* $CapacityJ_m$ *denotes the maximum number of cores allowed per JVM in m. Also, for each tenant* $t \in T$, $numGroups_t$ *denotes the number of core groups of t,* $numCores_{gt}$ *denotes the number of cores in each group of t.*

III. Solution Approach

We now propose a novel AI-based solution to CGPP leveraging Constraint Programming (CP) [3]. In principle, a constraint formulation consists of three elements:

1) A set of *decision variables* to represent the unknowns of a problem, whereby
2) Each decision variable can take a value from a set of possible values, called *domain*, and
3) A set of restriction on the possible combinations of values referred to as *constraints*.

A. The Decision Variables

The CGPP lends itself naturally to a high-level model thanks to global constraints and the declarative nature of CP.

We only need a single set of decision variables to target the heart of the problem: where to assign each core?

JVM of Each Core: There is a one-to-one mapping between these variables and the cores. Each core is treated as an integer decision variable with a domain that consists of all JVMs.

Let $J = \Sigma_{m \in M} numJVM_m$ be the total number of JVMs and $G_t = \{1, \ldots, numGroups_t\}$ to refer to all groups of tenant t. Then,

$$jvm_{tgc}\{1, \ldots, J\} \quad \begin{array}{l} \forall t \in T, \forall g \in G_t \\ \forall c \in numCores_{tg} \end{array} \tag{1}$$

The other set of decision variables are as follows.

Machine of Each Core Group: These variables denote the machine assigned to each group and their domain is from 1 to the number of machines.

$$machine_{tg}\{1, \ldots, M\} \quad \forall t \in T, \forall g \in G_t \tag{2}$$

Tenant of JVM: These variables denote the tenant of each JVM and their domain is from 0 to the number of tenants. The value 0 means that the JVM does not belong to any tenant.

$$tenant_j\{0, \ldots, T\} \quad \forall j \in J \tag{3}$$

Load of JVM: These variables store the load of each JVM and their domain is from 0 to the core limit on that JVM. Let $m(j)$ return the machine index corresponding to a given JVM j. Then,

$$loadJ_j\{0, \ldots, CapacityJ_{m(j)}\} \quad \forall j \in J \tag{4}$$

Load of Machine: These variables store the load of each machine and their domain is from 0 to the core limit on that machine.

$$loadM_m\{0, \ldots, CapacityM_m\} \quad \forall m \in M \tag{5}$$

Notice how in both (4) and (5) the capacity constraints for JVMs and machines are implicitly enforced by the CP formalism via the domain information.

B. The Constraints

Now, we need to link these variables together via constraints to express the desired properties of a solution to CGPP.

The first constraint in our model links the assignment of cores to JVMs with the tenant of that JVM. This ensures Rule–I which states that each tenant should claim seperate JVMs.

$$tenant[jvm_{tgc}] = t \quad \begin{array}{l} \forall t \in T, \forall g \in G_t \\ \forall c \in numCores_{tg} \end{array} \tag{1}$$

Here we are indexing into a set of decision variables, $tenant$, using a *decision variable*, jvm_{tgc}, as opposed to a simple static array indexing operation. In CP, this is referred to as the `Element` global constraint [4] and it neatly captures this relation. Compare it for example with a low-level binary representation as in boolean formulations or linear programs that enumerates each possible assignment and then try to force or forbid some assignments.

The second constraint links the tenant of a JVM to the machine assigned to a particular group of a tenant. That means, the cores in that group can only be placed on the JVMs of that machine. For example, if a group belongs to the first machine, then the available values in the domain of the cores in that group can only consist of JVMs in the first machine. This ensures the same machine requirement in Rule–II.

Let *numJVMBefore* be an integer array of size M where each index corresponds to a machine and returns the number of JVMs before that machine. Then,

$$\begin{array}{l} lowerBound = numJVMBefore[machine_{tg}] \\ upperBound = lowerBound + numJVM[machine_{tg}] \\ lowerBound < jvm_{tgc} \leq upperBound \\ \qquad \forall t \in T, \forall g \in G_t \\ \qquad \forall c \in numCores_{tg} \end{array} \tag{2}$$

In order to express this constraint, we again take advantage of the powerful `Element` constraint and first calculate how much we should shift from the left, the $lowerBound$, and then, how far we can go towards right, the $upperBound$.

The third constraint states that the cores in the same group have to be in different JVMs. This ensures the unique JVM requirement in Rule–II. For simplicity we omit the c subscript of the jvm variable to refer to the collection of all cores. We take advantage of another expressive global constraint; `AllDifferent` [5]

$$AllDifferent(jvm_{tg}) \quad \begin{array}{l} \forall t \in T \\ \forall g \in G \end{array} \tag{3}$$

The next constraint links assignment of JVMs to cores with the load on that JVM using the Global Cardinality Constraint (`GCC`) [6], [7]. Remember that the load variables were created with appropriate upper bounds which implicitly enforces the capacity limits.

$$GCC(jvm, \{1, \ldots, J\}, loadJ) \tag{4}$$

$$
\begin{aligned}
m \in M &= \{1..6\} \\
t \in T &= \{1..5\} \\
g \in G_t &= \{\{1..10\}, \{1..15\}, \\
&\quad \{1..12\}, \{1..8\}, \{1..10\}\} \\
c \in C_t &= \{\{1..4\}, \{1..4\}, \{1..2\}, \\
&\quad \{1..4\}, \{1..2\}\} \\
maxCoreJVM &= 10 \\
maxCoreMachine &= \{145, 226, 145, 145, 145, 145\} \\
numJVM &= \{32, 48, 32, 32, 32\} \\
numJVMBefore &= \{0, 32, 80, 112, 144, 176, 208\} \\
J &= \sum_{m=0}^{|M|} numJVM_m = 208
\end{aligned}
$$

$$
\begin{aligned}
\forall t, g, c \quad & jvm_{tgc} \in \{1..J\} \\
\forall t, g \quad & machine_{tg} \in \{1..|M|\} \\
\forall j \quad & tenant_j \in \{0..|T|\} \\
\forall j \quad & load_j \in \{0..maxCoreJVM\} \\
\forall m \quad & loadM_m \in \{0..maxCoreMachine_m\} \\
\forall t, g, c \quad & tenant[jvm_{tgc}] = t \\
\forall t, g, c \quad & lb = numJVMBefore[machine_{tg}] \\
& lb < jvm_{tgc} \leq lb + numJVM[machine_{tg}] \\
\forall t, g \quad & AllDifferent(jvm_{tg}) \\
& GCC(jvm, \{1..J\}, loadJ) \\
\forall m \quad & loadM_m = \sum_{j=0}^{numJVM_m} loadJ[numJVMBefore_m + j] \\
& minimize \quad max(loadJ)
\end{aligned}
$$

Fig. 1. The instance data (on the left) and the corresponding constraint formulation with the decision variables, constraints, and the objective (on the right).

Our last constraint accumulates the load of every JVM in each machine yielding the machine load. Again, the capacity limit is enforced via the upper bound of the machine load variables.

$$
loadM_m = \sum_{j=0}^{j=numJVM_m} loadJ[numJVMBefore_m + j] \quad \forall m \in M \tag{5}
$$

C. The Objective

Finally, our objective can simply be stated as;

$$\min \qquad max(loadJ)$$

It is possible to infer simple bounds for the objective. Let $Cores$ be the set of all cores among all the groups of all tenants. Then the objective can not be less than the ratio of total number of cores divided by the total number of JVMs and cannot be greater than the maximum capacity among all JVMs.

$$\frac{|Cores|}{|J|} \leq max(loadJ) \leq max(CapacityJ)$$

D. The Search Directive

A key observation is that the value of all decision variables can be extracted once jvm variables are assigned to feasible values. Given the assignment of all cores, we can find out the load of each JVM, which tenant they belong to, the machine of each group, and so on. Hence, the critical decision variable in this model is jvm. We can provide this information to the constraint solver to focus the search on those variables first.

$$branch(jvm)$$

The relative branching order among the jvm variables can be static such as lexicographic order, or dynamic/adaptive such as weighted-degree [8] or impact-based search [9]. These heuristic are commonly available in most constraint solvers.

IV. Related Work

Resource allocation problems are commonly solved via (Meta)-Heuristics. As in [10], we also believe that the addition of side constraints such as privacy, reliability, and sustainability make declarative AI approaches a prime candidate to provide robust solutions that can easily adapt to changes.

Strikingly, researchers from Microsoft share our concern on evolving requirements in cloud systems and that heuristics might not eventually suffice [11]. While they propose to use a domain specific language for resource allocation as a declarative approach, we turn to constraint programming as a well-established generic AI paradigm for which several open-source and commercial solvers are available.

V. Numerical Results

Benchmarks: Based on the information and business requirements gathered from application teams, we created instances of varying sizes and complexity as listed in Table I. Our data set consists of instances generated using a combination of the following parameters: {25, 50, 100} machines, half of which was selected uniformly at random to have 32 JVMs and the other half has 48 JVMs, {25, 50} tenants with {10, 20} average group size per tenant. The physical machines considered have roughly 16 and 22 TBs storage limits, which result in 145 or 226 cores respectively on each machine when JVMs are limited to 600MBs. Each JVM is limited to 10 cores. Notice that, not all JVMs can be loaded to maximum core capacity due to storage limits on the machines that restrict the total number of cores. These instances result in 20% to 40% density in the machines, at which point we hit the storage limits. Overall, these benchmark instances allow us to work with thousands of JVMs in our experiments whereby the resource provisioning spans a time period of 2 months.

Implementation: The constraint model is implemented using an in-house constraint solver at Oracle. Since the operators we used are commonly available, our solution can also be implemented with ease using other commercial or open-source constraint solvers such as IBM Ilog CP Optimizer [12] or Gecode [13]. All experiments were run on a Dell laptop with Intel Core i5 CPU @2.5 GHz and 8.00 Gb RAM.

Runtime Experiments: We remind that our results are for solving *all* cores and *all* groups at once. In that sense, we are solving larger instances compared to a system that is progressively growing as needed. There is a strict runtime limit of 2 minutes based on the needs of the deployed application.

As shown in Table I, any experiment with 25 or 50 machines took on average less than ∼5 seconds to find and *prove* the optimal solution. Going forward, any experiment with 100 machines were solved to optimality on average less than ∼2 minutes. At this point, we hit the time limit dictated by application preferences as longer runtimes are not considered viable in practice. Within this time frame, our solution finds the best allocation involving thousands of JVMs which met the business requirements successfully.

Parallelization: It is reasonable to assume that larger instances, e.g., thousands of machines might be of interest in some applications. In that case, for further scalability, our monolithic solution can be parallelized by decomposing the instances into smaller groups of machines and tenants, and running the constraint solver on the resulting models simultaneously. A similar approach is successfully employed in BtrPlace, a flexible virtual machine scheduler based on constraint models [14].

Development Time: We welcome ideas from the community on how to apply other techniques and improve upon our solution for CGPP. However, let us emphasis again the importance of simplicity. Our constraint formulation is less than 50 lines of source code and there is almost no gap between the application and the implementation. This is highly desired in a production environment and reduces the time to solution. Our development time was only a few days to build the constraint model followed by testing, integration, and deployment.

MACHINES	
Number of machines	Avg. Runtime
25	$<$ 2 seconds
50	$<$ 5 seconds
100	$<$ 2 minutes
% of 32 JVMs	∼50
% of 48 JVMs	∼50
Machine storage limit	{16 TB, 22 TB}
Machine core limit	{145, 226}
JVM storage limit	600 MB
JVM core limit	10
TENANTS	
Number of tenants	{25, 50}
Avg. group size	{10, 20}
Group core limit	{2, 4}
% enterprise tenants	∼50
% standard tenants	∼50
RESOURCE PROVISIONING	2 months

TABLE I
CGPP INSTANCES AND THE AVERAGE RUNTIME OVER 30 INSTANCES GROUPED BY THE NUMBER OF MACHINES FOR ALL PARAMETERS.

VI. CONCLUSION

We introduced a new combinatorial problem, namely CORE GROUP PLACEMENT, targeting the allocation of heterogeneous resources in cloud centers. We presented a number of business requirements motivated by a real application. We showed a succinct and efficient constraint formulation is possible. The formulation was able to address multiple requirements; virtualization, capacity limits, privacy and performance issues, and load balancing. The use of a declarative AI-based approach allowed flexibility and extensibility; much needed properties in cloud settings that are evolving permanently. Finally, experimental results demonstrated that the constraint propagation is highly effective in finding optimal solutions. Overall, our solution excelled in meeting application requirements and we believe that AI-based approaches stand out as an attractive alternative to be considered for industrial applications in cloud management.

REFERENCES

[1] K. Kant, "Data center evolution: A tutorial on state of the art, issues, and challenges," *Computer Networks*, vol. 53, no. 17, pp. 2939–2965, 2009.

[2] D. Smiley, E. Pugh, and K. Parisa, *Apache Solr 4 Enterprise Search Server*. Packt Publishing, 2014.

[3] K. Apt, *Principles of Constraint Programming*. Cambridge Univ. Press, 2003.

[4] P. Van Hentenryck and J. P. Carillon, "Generality versus specificity: An experience with AI and OR techniques," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988, pp. 660–664.

[5] J.-C. Régin, "A filtering algorithm for constraints of difference in csps," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, ser. AAAI '94. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 362–367. [Online]. Available: http://dl.acm.org/citation.cfm?id=199288.178024

[6] Y. T. A. Oplobedu, J. Marcovitch, "Charme: Un langage industriel de programmation par contraintes, illustré par une application chez renault," in *Proceedings of the Ninth International Workshop on Expert Systems and their Applications: General Conferencehnical*, 1989, pp. 55–70.

[7] I. Katriel and S. Thiel, "Complete bound consistency for the global cardinality constraint," *Constraints*, vol. 10, no. 3, pp. 191–217, 2005.

[8] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, "Boosting systematic search by weighting constraints," in *ECAI*, vol. 16, 2004, p. 146.

[9] P. Refalo, "Impact-based search strategies for constraint programming," in *Principles and Practice of Constraint Programming–CP 2004*. Springer, 2004, pp. 557–571.

[10] J.-C. Régin and M. Rezgui, "Discussion about constraint programming bin packing models." *AI for data center management and cloud computing*, vol. 11, p. 08, 2011.

[11] A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in *proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 15.

[12] IBM, *IBM ILOG CPLEX Optimization Studio 12.5*, 2015.

[13] Gecode Team, "Gecode: Generic constraint development environment," 2016, available from `http://www.gecode.org`.

[14] F. Hermenier, J. Lawall, and G. Muller, "Btrplace: A flexible consolidation manager for highly available applications," *Dependable and Secure Computing, IEEE Transactions on*, vol. 10, no. 5, pp. 273–286, 2013.

Leveraging an Homomorphic Encryption Library to Implement a Coordination Service

Eugenio A. Silva Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

***Abstract*— The paper presents MorphicLib, a new partial homomorphic cryptography library written in Java that can be used to implement a wide-range of applications. The paper shows the use of the library with the HomomorphicSpace coordination service. This service is a tuple space that stores encrypted tuples but still supports operations like returning tuples with values within a certain range.**

I. Introduction

Researchers and practitioners for decades assume that encrypted data cannot be processed. Generally, it is necessary to decrypt the data before performing any operations over that data. This problem becomes specially important when large data resides in a public cloud. In this case, there is a dilemma between two alternatives: (1) either the data is decrypted in the server-side (in the cloud), which poses security issues, namely the need to pass the key to the server and have the information exposed to insider threats in the cloud [1], [2] at least during the operation; or (2) the data is decrypted in the client-side, which involves downloading the data from the cloud (typically expensive and slow) and prevents using the computation power of the cloud. A good solution to this dilemma would be to perform the desired operations directly on the encrypted data, at the server-side, where it is stored. This would avoid the cost of moving the data, would allow leveraging the cloud's computation power, and would solve the security issues.

The term *homomorphic encryption* designates forms of encryption that allow some operations to be performed over encrypted data, without decrypting it. With those forms of encryption, it is possible to perform resource-intensive computing tasks at server-side without having to decrypt it first. Homomorphic encryption became popular with Gentry's work [3], which was coincident with the emergence of cloud computing. Gentry's scheme provides fully homomorphic encryption (FHE), so it allows performing arbitrary computation on encrypted data. Other FHE schemes were presented in the following years [4], [5].

Although in theory FHE solves the problem of computing encrypted data outsourced to a cloud, the performance of these schemes is too poor for practical applications [6]. For that reason, much effort has been placed in developing and using *partial homomorphic encryption* (PHE) schemes [7], [8], [9], [10], [11]. PHE schemes allow performing some computation over encryption data, but not arbitrary computation like FHE. CryptDB is an important step towards the deployment of PHE in real systems [11]. CryptDB is a relational database management system that stores encrypted data and allows doing SQL queries. The system combines a set of PHE schemes and has enough performance for many applications.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

This paper presents *MorphicLib*, a new partial homomorphic cryptography library that can be used to implement a wide-range of applications. The library contains functions (normally) executed at the client-side and functions for the server-side. For the client-side there is the encryption scheme, i.e., functions for encryption, decryption, and key generation. For the server-side there are homomorphic equivalent operations (addition, multiplication, comparison, etc.). The library was programmed in Java in order to ensure portability, i.e., that it can be executed in different platforms, both client and server-side. Moreover, Java is arguably the most popular general purpose programming language today, with a large set of APIs, and a strong programming community.

The paper shows the usefulness of the library with a service that is interesting in its own right, the *HomomorphicSpace coordination service*. Coordination services like Google Chubby [12], Google Megastore [13], Apache Zookeeper [14], and GigaSpace's tuple space (now part of XAP) [15] are important components in current cloud systems. They are used for tasks such as synchronization, locking, orchestration, metadata storage, leader election, and replica failure detection. *DepSpace* [16], [17] is a tuple space, i.e., a coordination service that follows Linda's associative memory paradigm [18], similarly to GigaSpace's tuple space. DepSpace is replicated, so it can tolerate arbitrary (Byzantine) faults in some of its replicas.

HomomorphicSpace is an extension of DepSpace with homomorphic encryption (MorphicLib), so that data (tuples) can be stored encrypted at the servers. DepSpace's commands to read and retrieve tuples were extended with operators for inequality, less/greater relations, and keyword search, all over encrypted data. Moreover, HomomorphicSpace supports addition and multiplication of tuples in the server. Data is never decrypted at the server, only at the client after retrieval. HomomorphicSpace is Byzantine fault-tolerant like DepSpace.

II. MorphicLib Library

As already mentioned, MorphicLib is a novel library of partial homomorphic cryptographic functions written in Java and providing a Java API. MorphicLib was not developed from scratch, but based on existing source code whenever possible. The objective was both to simplify the task and to avoid introducing bugs, which tend to appear due to the complexity of cryptographic code. This library can be used both at the

TABLE I
MORPHICLIB'S MAIN CLASSES

Property	Homomorphic Operations	Class	Input Data Types
Random	None (strong cryptanalisys resistance)	`HomoRand`	Strings, Byte Arrays
Deterministic	Equality an inequality comparisons	`HomoDet`	Strings, Byte Arrays
Searchable	Keyword search in text	`HomoSearch`	Strings
Order preserving	Less, greater, equality comparisons	`HomoOpeInt`	32 bit Integers
Sum	Add encrypted values	`HomoAdd`	BigInteger, String
Multiplication	Multiply encrypted values	`HomoMult`	BigInteger, String

client-side to encrypt and decrypt data, and at the server-side to do operations over encrypted data.

The code of the library is organized in classes, one per *homomorphic property*. One crucial different between PHE and FHE is that in the former data has to be encrypted taking into account the kind of operation that will be supported over the encrypted data. With FHE, on the contrary, arbitrary computation is possible over encrypted data (at a cost, in terms of performance). As we opted for PHE, for each homomorphic operation we have four kinds of functions (or methods):

- key generation function, typically used at client-side;
- encryption function, typically used at client-side;
- decryption function, typically used at client-side;
- homomorphic operation functions, which do operations over encrypted data, typically used at the server-side.

Information about the properties of the PHE algorithm, the operations supported, and the classes is in Table I.

III. HOMOMORPHICSPACE COORDINATION SERVICE

This section presents HomomorphicSpace, a coordination service that leverages MorphicLib to handle encrypted data at the server. HomomorphicSpace is an extension of DepSpace, so we start by presenting the latter.

A. DepSpace

DepSpace (Dependable Tuple Space) is a fault- and intrusion-tolerant *tuple space* [16]. Architecturally it is client-server system implemented in Java. The server-side is replicated in order to tolerate arbitrary faults. The client-side is a library that can be called by applications that use the service. Clients communicate with the servers using a Byzantine fault-tolerant total order broadcast protocol called BFT-Smart. The most recent version supports extensions to the service [17].

The service provides the abstraction of tuple spaces. A tuple space can be understood as a shared memory that stores *tuples*, i.e., sequences of *fields* (data items) such as (1, 2, a, hi). Tuples are accessed using *templates*. Templates are special tuples in which some fields have values and others have undefined values, e.g., wildcards meaning any value ("*"). A template *matches* any tuple of the space that has the same number of fields, in which the values in the same position are identical, and the undefined values match in some sense. For example, the template (1, *, a, *), matches the tuples (1, 2, a, hi) and (1, 7, a, 14), but neither (1, 2, b, 4) nor (1, 2, a, hi, 5). DepSpace supports a set of commands, issued by clients and executed by the servers. Here we consider the following commands:

- `out` *tuple* – inserts a tuple in the space;
- `inp` *template* – reads and removes from the space a tuple that matches the template;
- `rdp` *template* – reads but does not remove from the space a tuple that matches the template;
- `inAll` *template* – reads and removes from the space all tuples that match the template;
- `rdAll` *template* – reads but does not remove from the space all tuples that match the template.

DepSpace does not support homomorphic operations. However, it allows fields to be encrypted and basic equality matching by storing a hash jointly with the encrypted field. This solution however is vulnerable to trivial brute force and dictionary attacks. It does support the definition of access control policies using its policy-enforcement mechanism.

B. Threat Model

The threat model we consider for HomomorphicSpace is similar to the threat model for DepSpace except for one crucial difference: we consider that any server may be adversarial and try to read the content of the tuples it stores. We consider that all tuples of their fields for which confidentiality has to be preserved are encrypted using homomorphic encryption, preventing malicious servers from doing such an attack. Similarly to DepSpace, adversaries may compromise up to f out of $3f + 1$ servers and stop them or modify their behavior arbitrarily. This is tolerated using replication and the BFT-Smart protocol. Network messages may also be tampered with by the adversary, but the system uses this using secure channels.

C. Commands

HomomorphicSpace extends DepSpace to allow commands over tuples with encrypted data items. More precisely in comparison with DepSpace, HomomorphicSpace: (1) supports the original match operations over encrypted data; (2) extend matching beyond the equality and wildcards with more complex matches, i.e., inequality, order comparisons (lower, greater), and keyword presence in a text, all over encrypted data; (3) allow addition and multiplication of encrypted fields.

Besides values and wildcards ("*"), HomomorphicSpace's *templates* can include the following fields:

- $\%\ word_1 \ldots word_n$ – matches a textual field containing all the words indicated;
- $> val$ – matches a numeric field containing a value greater than *val*;
- $>= val$ – matches a numeric field containing a value greater or equal to *val*;
- $< val$ – matches a numeric field containing a value lower than *val*;
- $<= val$ – matches a numeric field containing a value lower or equal to *val*.

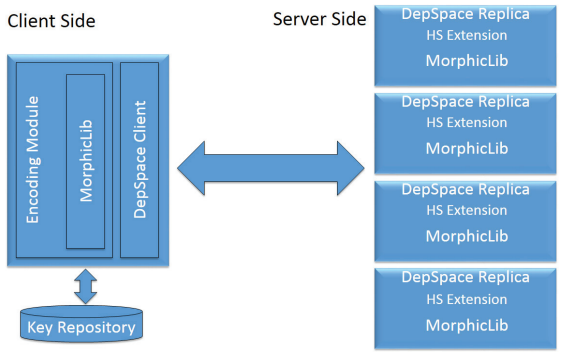


Fig. 1. HomomorphicSpace architecture

HomomorphicSpace adds three commands to those provided by DepSpace (Section III-A). The first is `crypt` *id template* and aims to define a *tuple encryption type*. The command takes as input an identifier (id) for the type it will create, and a template with the homomorphic operation desired for each of the fields, which will determine the homomorphic property. For example, if the template contains for a given field the operation "=", the system infers that the encryption to be used for that field is deterministic, which is the strongest that allows that operation. If no operation is indicated, the field will not be encrypted. The complete list of interpreted operations is:

- $=, <>$ – determinist encryption
- $>, >=, <, <=$ – order preserving encryption
- $\%$ – searchable encryption
- $+$ – Paillier
- $\&$ – RSA
- . – random encryption
- other value – no encryption

The second command is `rdSum` *template*. This command starts by collecting all the tuples that match the template similarly to `rdAll`, then sums the (encrypted) fields with $+$ in the template. The function returns a single tuple with the result. The third command is `rdProd` *template*, which works similarly to `rdSum` but does multiplication instead of sum.

This scheme allows a single type of encryption per field (unlike CryptDB). However, with the tuple data structure this is not a restriction. For instance, for tuples with a single numeric field, two operations like equality and sum can be supported by transforming that field in two and using the tuple encryption type $(=, +)$.

D. Architecture and Functioning

Architecturally the HomomorphicSpace is similar to DepSpace, with a client-side and a server-side. Figure 1 represents the system with 4 replicas, i.e., with $f = 1$. From the confidentiality point of view, the server-side is untrusted and the client-side trusted.

The server-side of the system is mostly DepSpace code with the server-side of the MorphicLib and with extensions to process the homomorphic operations. The client-side includes MorphicLib's and DepSpace's client-side libraries. The main functions of the client is to encrypt tuples and send them to the tuple space, and to decrypt them before they are delivered to the application. When a tuple is encrypted, the encryption keys are stored in a *key repository* (a folder with one file per key). Next we describe both sides in more detail.

Client side – When the `crypt` command is issued (i.e., that method is called), the library generates keys for every field of the tuple for which homomorphic properties are desired. These keys are stored jointly with the tuple encryption type (id and template) in the key repository. All the other commands (`out`, `inp`, etc.) include an id that the library uses to retrieve the corresponding tuple encryption type and keys from the repository. If the operation indicated in a field is not compatible with the encryption defined with the `crypt` command, the command returns an error.

The library uses the DepSpace client library to send to the servers the command and the fields. If the command is an `out`, the fields are encrypted with the scheme defined in the tuple encryption type and the keys previously stored. If the command involves reading tuples, it contains the operation and encrypted values. Note that each field of each id has its own key (or key pair for RSA), but the same field for the same id is always encrypted with the same key. When the library receives a reply from the servers, it does the opposite, i.e., it decrypts the encrypted fields using the corresponding schemes and keys.

Server side – The server-side handles different commands in different ways. The `out` command is executed the same way as in DepSpace. The fields may be encrypted but they come encrypted from the client so the tuple is stored unmodified. The `inp` and `rdp` commands were modified using DepSpace's extension mechanism in order to support the $=$, $<>$, $>$, $>=$, $<$, $<=$, and text search operations over encrypted data, returning one of the matching tuples. The `rdall` and `inall` commands work similarly, as `rdp` and `inp`, but return all matching tuples. The `rdSum` and `rdProd` commands are implemented as a modification of the original `rdAll` command that returns a single tuple with the relevant fields respectively added or multiplied.

IV. Experimental Evaluation

We did a set of experiments to evaluate the performance of HomomorphicSpace. The experiments were executed in two personal computers. The first had an Intel(R) Core(TM) i7-3537U CPU @ 2.00 GHz, 4 GB RAM, and Windows 8.1 (64 bits). The second had an Intel(R) Core(TM)2 Duo CPU U9400 @ 1.40 GHz, 3,5 GB RAM, and Ubuntu 15.10 (64 bits). The software was executed using Java 1.8 with Oracle JDK in the Windows Machine and OpenJDK in the Linux Machine. The 2 machines were connected by an IEEE 802.11b/g/n switch (up to 54 Mbps).

We used the Linux machine to run the client, and the Windows machine to run the servers. Although we used a single machine for the server-side, it contained 4 server replicas. The client-side application had a *command line interface* that allows writing commands to be executed by the tuple space. The performance of tuple space operations depends on the

TABLE II
EXACT MATCH EXECUTION TIMES (MS)

Encryption used	out 100 tuples	rdp 1 tuple	inAll
No encryption	3659 ± 465	30 ± 5	235 ± 39
Deterministic	3747 ± 724	35 ± 5	342 ± 62
Order Preserving	3771 ± 580	32 ± 8	312 ± 75

load of the space, so we started all the experiments with an empty tuple space.

1) Performance of tuple exact matching with encrypted fields: In order to evaluate the performance of exact matching (equality) with encrypted fields for the relevant encryption schemes (and no encryption), we made the following test: (1) insert (`out`) 100 tuples with a single field in the tuple space, which are encrypted in the cases of Determinist and Order Preserving encryptions; (2) execute an exact match with `rdp value` and decrypt the tuple retrieved (if encrypted); (3) retrieve all tuples from the space with `inAll` $\star$ and decrypt the 100 tuples (if encrypted).

The tests made were all exact match (equality), independently of the encryption scheme or no encryption used (see Step 2 above). However, the performance for inequalities (different, greater than, greater or equal to, ...) would be very similar as all of them are simple byte comparisons. Each test was executed 30 times and the times for the three steps were measured. The results are in Table II. A first conclusion from the table is that encryption has no impact in the match operations, as the comparisons without encryption (2nd row) and with encryption (3rd and 4th rows) take very similar times (column for command `rdp`). A second conclusion is that the use of encryption (3rd/4th rows versus 2nd row) did not cause observable delay in the experiments (2nd and 4th columns). This result is consistent with the values obtained for the library, with encryption/decryption times that are fractions of a millisecond. Furthermore, the encryption/decryption load is *at the client*, not at the server side, so it has no impact in the capacity of the servers to process requests.

2) Performance of the ordered operations: In order to evaluate the performance of the ordered operations we made the following test: (1) insert (`out`) 100 tuples with a single field in the tuple space, with values from 0 to 99, encrypted with the Order Preserving scheme; (2) execute an `rdp` (read one matched tuple), with the parameters indicated in the first column of Table III and decrypt it; (3) execute an `inAll` (read and delete all matched tuples), with the parameters indicated in the first column of Table III and decrypt. The test was repeated 30 times.

We can observe in the table that the execution times of the `rdp` command are all inside the deviation intervals of each other, meaning that the type of match does not affect the execution times. For the `inAll` operation we can see that the slower operation in the one that reads all the tuples ($\star$), the second slower is the one that reads all minus one ($<>$), and the faster operation is the one that reads just one tuple (=). The other operations have execution times somewhere in the middle. This allow us to conclude that the execution time of the `inAll` operation depends not on the type of comparison, but on the number of tuples retrieved. This is caused by the communication delay caused by more data.

TABLE III
ORDERED OPERATIONS EXECUTION TIMES

Condition	rdp (ms)	inAll (ms)	Tuples selected
$*$ (match all)	35 ± 8	257 ± 27	100
$=$ (match)	29 ± 9	33 ± 22	1
$<> 50$	28 ± 6	209 ± 7	99
< 50	31 ± 7	195 ± 46	50
$<= 50$	30 ± 8	174 ± 21	51
> 50	29 ± 7	181 ± 26	49
$>= 50$	34 ± 8	204 ± 53	50

Acknowledgements This work was supported by the European Commission through project H2020-653884 (SafeCloud) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

REFERENCES

[1] Cloud Security Alliance, "The notorious nine: Cloud computing top threats in 2013," Feb. 2013.
[2] F. Rocha and M. Correia, "Lucy in the sky without diamonds: Stealing confidential data in the cloud," in *Proc. 1st DCDW Workshop*, 2011.
[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annual ACM Symposium on Theory of Computing*, 2009.
[4] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology – EUROCRYPT 2010*. Springer, 2010.
[5] M. Yagisawa, "Fully homomorphic encryption without bootstrapping," Cryptology ePrint Archive, Report 2015/474, 2015.
[6] K. Lauter, M. Naehrig, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proc. 3rd ACM Workshop on Cloud Computing Security*, 2011.
[7] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Proc. 28th Annual International Conference on Advances in Cryptology*, 2009.
[8] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, Jan 2014.
[9] B. Ferreira, J. Rodrigues, J. Leitão, and H. Domingos, "Privacy-preserving content-based image retrieval in the cloud," *CoRR*, vol. abs/1411.4862, 2014.
[10] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *Computer Security - ESORICS*, 2015.
[11] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. 23rd ACM Symposium on Operating Systems Principles*, 2011.
[12] M. Burrows, "The Chubby lock service for loosely-coupled distributed systems," *Proc. 7th Symposium on Operating Systems Design and Implementation*, 2006.
[13] J. Baker, C. Bond, J. Corbett, and J. Furman, "Megastore: Providing scalable, highly available storage for interactive services." in *Proc. 5th Biennal Conference on Innovative Data Systems Research*, 2011.
[14] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free coordination for Internet-scale systems," in *Proc. 2010 USENIX Annual Technical Conference*, 2010.
[15] GigaSpaces, "XAP 9.0 documentation – product overview – concepts," http://wiki.gigaspaces.com/wiki/display/XAP9/Concepts, 2011.
[16] A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga, "DepSpace: a Byzantine fault-tolerant coordination service," in *Proc. 3rd ACM SIGOPS/EuroSys European Systems Conference*, Apr. 2008.
[17] T. Distler, C. Bahn, A. Bessani, F. Fischer, and F. Junqueira, "Extensible distributed coordination," in *Proc. 10th ACM SIGOPS/EuroSys European Systems Conference*, 2015.
[18] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programing Languages and Systems*, vol. 7, no. 1, Jan. 1985.

Multi-Phase Proactive Cloud Scheduling Framework Based on High Level Workflow and Resource Characterization

Nelson Mimura Gonzalez
Tereza Cristina Melo de Brito Carvalho
Escola Politécnica, Universidade de São Paulo (USP), Brazil
e-mail: {nmimura, carvalho}@larc.usp.br

Charles Christian Miers
Santa Catarina State University (UDESC), Joinville, Brazil
e-mail: charles.miers@udesc.br

***Abstract*—Workflows are used to represent applications in terms of the computational cost and the interdependencies of tasks. In parallel, clouds are a viable solution to execute complex applications in terms of performance and cost. This paper presents a cloud scheduling framework composed by multiple proactive phases that continuously compute and improve resource allocation and load distribution for workflow execution in cloud environments. The framework relies on a high-level characterization of resources and workflows to describe the capabilities provided by the infrastructure and the performance requirements to be met. Implementation and tests are based on the optimization of workflows executed on three scenarios: a private cloud, a hybrid cloud (private and public), and a multi-cloud setup. Results show improvement of run time performance compared to greedy approaches. Moreover, the framework is able to handle performance fluctuations, especially for long duration workflows.**

1. Introduction

Scientific workflows represent a class of complex applications whose execution requires the utilization of the compute capacity of a large number of nodes [1]. In parallel, cloud computing is used to deliver on demand storage and compute capabilities, as the resources can be leased and utilized on demand [2]. The conjunction of these technologies is successful, with several existing solutions for execution of e-Science applications on commercial clouds [3].

However, with more resources available and more complex infrastructures, even a small performance imbalance among nodes might lead to monetary costs [3]. Existing proposals for cloud scheduling [3], [4] adopt a simple resource model for nodes that do not fully address the complexities of each workflow. Furthermore, for data-intensive workflows such as the APEX workflows [5] the data transfers represent a substantial fraction of the total time. Nevertheless, for most scheduling algorithms data movement is considered part of the execution, which is not the case for modern applications [1]. In fact, data movement might even dominate execution time and cost for these cases, especially for hybrid and multi-cloud setups.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

This paper presents MPSF (Multi-Phase Proactive Scheduling Framework), a cloud scheduling framework based on multiple scheduling phases that continuously assess system resources in order to dynamically optimize resource distribution. MPSF is based on a rich high-level characterization of resources and workflows that describes the interactions between workflow phases and the computational cost for each case. Moreover, the framework considers both compute and transfer times to distribute the input to the nodes and also to retrieve the results. The workflow characterization is used to determine the time to process and transfer the input. The scheduling framework computes the optimal data distribution by determining how much data each node must receive in order to balance the execution and transfers across all nodes. In addition, MPSF is able to adequately distribute data in scenarios with multiple clouds, considering the cost of sending and receiving data over slower links. The framework also provides rescheduling capabilities in order to redistribute the workload, addressing the performance fluctuations on the system.

The remainder of the document is organized as follows: Section 2 presents the problem addressed in this paper. Section 3 presents the scheduling framework, the performance model, the resource and workflow model, and some of its features. Section 4 presents the evaluation methodology used in the experiments. Section 5 presents the experimental results in three different scenarios (private, hybrid, and multi-cloud). Section 6 presents the analysis of the results and of the framework. Section 7 presents the related work and compares MPSF to some other similar approaches. Section 8 presents our considerations and future work.

2. Problem Definition

For workflows with a large computational component and associated I/O (e.g., modern scientific workflows [5]), the fundamental problem to be solved is how to distribute input data to the cloud nodes. The usual problem definition starts by considering a set of nodes $n_i \in \mathcal{N}$, which is the resource model for the problem. Moreover, consider also the workflow represented as a weighted Directed Acyclic Graph (DAG) $\mathcal{G} = (\mathcal{T}, \mathcal{D})$, where $\mathcal{T}$ is the set of tasks and $\mathcal{D}$ is the dependencies between tasks.

Another approach to interpret this problem is to consider the total input to be processed, d, the set of nodes $\mathcal{N}$ and the workflow description $\mathcal{W}$. In this case, the nodes are described not as a function of their peak hardware capabilities (e.g., number of instructions per second), but as a function of the workflow being executed. Compute and transfer capacities are measured as a data rate (i.e., a multiple of bytes per second). Each node receives a fraction d_i of the total input d, thus the time t_i of sending data to a node, processing the data, and receiving the result, is calculated based on the amount of data d_i sent to node i, the node processing capacity p_i, and its transfer capacity r_i.

3. MPSF

3.1. Performance Model

Based on the problem definition, this is a fundamental optimization problem and the makespan (time for completion of all tasks in all nodes) is obtained by balancing the execution of all nodes in a way that the times t_i are all the same. Considering that the input is concentrated in a particular node, the transfers occur sequentially and the time to transfer to one node must be taken into account when optimizing the t_i values. The process continues until all nodes have received their data fragments. Thus, the time expression for each node is:

$$t_i = \sum_{j=1}^{i} \frac{d_j}{r_j} + \frac{d_i}{p_i} \tag{1}$$

The system to be solved is composed by the expressions:

$$\begin{cases} t_i = t_j, \forall i, j \in \mathcal{N} \\ \sum_{i=1}^{\mathcal{N}} d_i = d \end{cases} \tag{2}$$

Solving the system, the final expression for the amount of data to send to a node in order to optimize the execution of tasks across the system is:

$$d_m = \frac{\prod_{j=m}^{n-1} p_j (\frac{1}{r_{j+1}} + \frac{1}{p_{j+1}})}{1 + \sum_{i=1}^{n-1} \prod_{j=i}^{n-1} p_j (\frac{1}{r_{j+1}} + \frac{1}{p_{j+1}})} \cdot d \tag{3}$$

This final expression is similar to a weighted distribution, but the main difference is that the expressions also consider the transfer time as a relevant portion of the whole execution.

3.2. Resource Model & Workflow Model

One of the main features of MPSF is the utilization of a high-level characterization of resources and workflows. MPSF defines the computational capacity as a function of the workflow to be executed, as the amount of data that can be processed at a time varies according to the program being executed. The high-level description can be manually generated based on expert knowledge of the workflow, or it can be automatically generated by MPSF automated characterization capabilities.

The workflow model in MPSF is defined using five operators, as observed in the example in Figure 1. These operators are similar to the ones described in prior references, such as the work from Bharathi et al. [6]. In the Figure 1, from left to right, the first operator defines the **input** of the workflow, usually mapped to one or more nodes that contain the inputs for the execution of the workflow; the second operator is a **splitter**, responsible for dividing the input into several smaller pieces, referred as *fragments*, which are distributed to the compute nodes; the third operator is a **compute** operation, which could be as simple as adding numbers or as complex as an actual workflow phase; the fourth operator is a **merger**, responsible for receiving several result fragments (obtained by processing the input fragments) and merging them into a single result; finally, the fifth operator indicates an **output**, representing the final result of the execution of the workflow.

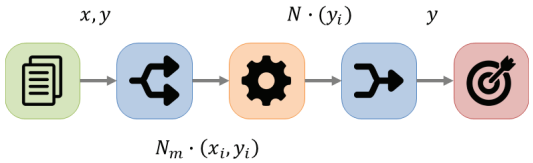


Figure 1: Example of workflow description.

In this example (Figure 1) the input is represented by two variables, x and y, which could vectors for instance. These inputs are sent to the splitter which generates N fragments. Each compute node m receives N_m fragments, and the products generated altogether are N result fragments containing a new value for y.

3.3. Multi-Phase Proactive Scheduling

The scheduling decision is computed several times before and during the execution of the workflow. First the scheduling is computed before starting the workflow in order to calculate how much input data each node should receive. Then, the scheduling decision is continuously reevaluated to handle performance fluctuations in the systems. If the fluctuation surpasses a certain threshold configured in the framework, a new data distribution is calculated considering that now each compute node is a potential input node. On the other hand, even if this threshold is surpassed, if the performance benefit of redistributing the load is not higher than another threshold, nothing is modified.

4. Evaluation Methodology

Three systems were used in the experiments to evaluate the performance of MPSF. Two systems are located in the United States: **Gauss**, with ten nodes based on the Intel Xeon X5687 processor (16 VCPUs and 32 GB RAM per node), and **Vulcan**, with sixteen nodes based on the IBM POWER8 processor (160 VCPUs and 512 GB RAM per

node). The third system is located in Brazil: **Mamona**, with seven nodes based on the Xeon E3-1230 processor (8 VCPUs and 16 GB RAM per node). All machines were configured to run Ubuntu Trusty 14.04. The cloud environments were set up using **Docker**[1], enabling fast deployment of containers compared to virtual machines. The resources were partitioned in advance by allocating one container per VCPU due to the similarity in the amount of memory per VCPU among the three systems.

Three scenarios were built for the execution of experiments. **Scenario A** is a private cloud built using nodes from system Gauss. **Scenario B** is a hybrid cloud combining the private cloud from Scenario A and a public cloud built using nodes from system Vulcan. Finally, **Scenario C** is a multi-cloud scenario combining the resources from scenario A to a remote private cloud.

Three benchmarks were selected for the experiments. The first is AXPY, the scaled vector addition in the form $y = \alpha x + y$, where x and y are vectors with N elements each and α is a constant. Figure 2 shows the visual representation of this benchhmark. Vectors x and y are sent to the splitter, generating N fragments each containing an index and the x_i and y_i values to compute the result. Each compute component receives N_m fragments and the value of α, generating N_m tuples containing the index and the final value of y_i. The merger receives all N tuples generated by the compute components, each tuple containing an index and the final y_i value. These values are combined and the merger creates the final y vector.

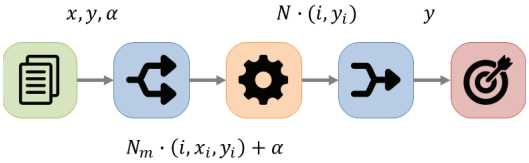


Figure 2: Workflow description for AXPY.

The second benchmark is GEMM, the general matrix to matrix multiplication in the form $C = \alpha AB + \beta C$, where A, B, and C are matrices and α and β are constants. Matrices were set square matrices $N \times N$, and α and β were set to 1.0. The visual representation of this benchmark is depicted in Figure 3. Matrices A, B, and C are sent to the splitter, producing N^2 tuples with enough information to compute a single cell of the resulting matrix. Finally, the merger receives N^2 result fragments, creating the final matrix C.

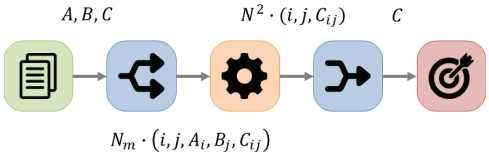


Figure 3: Workflow description for GEMM.

The third benchmark is WAVG, the implementation of a weighted average algorithm given a vector of N values v_i and their associated weights w_i. The splitter receives both vectors and generates N fragments composed by the tuple (v_i, w_i). Each compute component m receives N_m fragments and calculates a local weighted average. The visual representation of this workflow is depicted in Figure 4.

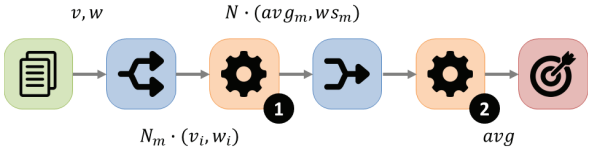


Figure 4: Workflow description for WAVG.

4.1. Compared Methods

The methods compared in the experiments were:

1) Single/Serial: Execution on a single thread;
2) Single/Parallel: Single node using all cores;
3) Naive/VCPU: Naive distribution strictly based on processing power and greedy over VCPUs (i.e., tries to use all VCPUs available);
4) Naive/Core: Naive distribution greedy over cores;
5) Naive/Node: Naive distribution greedy over nodes;
6) MPSF no-resc.: Partial implementation of MPSF without the rescheduling components; and
7) MPSF: The full implementation of MPSF.

5. Experimental Results

Experiments were repeated ten times and the results represent the average for each case.

5.1. Scenario A

The results for AXPY on Scenario A (private cloud) are summarized in Table 1. Results show the importance of the I/O component during resource allocation phase. For 50G the memory footprint (800 GB) is larger than the total memory available in the system (320 GB), thus requiring intense disk I/O. Because the naive algorithms do not consider the I/O components in the initial resource allocation the performance is severely compromised. MPSF is able to gradually split the input based on the time to transfer

Table 1: Results (run time in seconds) for AXPY on Scenario A with vector size N of 500 million elements (8 GB of total footprint), 5 billion elements (80 GB), and 50 billion elements (800 GB).

#	Algorithm	500M	5G	50G
1	Single/Serial	4.96	68.15	-
2	Single/Parallel	0.67	22.78	-
3	Naive/VCPU	0.22	3.09	62.05
4	Naive/Core	0.21	3.01	49.87
5	Naive/Node	0.32	4.12	54.10
6	MPSF no-resc.	0.14	1.19	28.15
7	MPSF	0.15	1.21	26.82

1. https://www.docker.com/

data from the input node to the other nodes. Compared to the best naive approach, for 50G vectors MPSF provides a speedup of 1.86x. Moreover, the rescheduling mechanism provides an improvement of 4.7% by handling performance fluctuations in the system.

Table 2: Results (run time in seconds) for GEMM on Scenario A with matrix side sizes N of 10k (total memory footprint of $3 \cdot (10k)^2 \cdot 8\text{B} = 2.4$ GB), 20k (9.6 GB), and 40k (38.4 GB).

#	Algorithm	10k	20k	40k
1	Single/Serial	82.34	667.31	-
2	Single/Parallel	12.10	188.40	-
3	Naive/VCPU	1.49	13.12	252.13
4	Naive/Core	1.52	14.39	128.93
5	Naive/Node	8.48	102.26	979.90
6	MPSF no-resc.	1.38	13.17	122.35
7	MPSF	1.41	13.25	117.41

The results for GEMM on scenario A are summarized in Table 2. Matrix sizes are presented as the number of elements in each dimension (N). For the 20k matrix the Naive/VCPU distribution provides a slightly better result over MPSF, as the I/O contention effects are not expressive and due to the management overhead generated by the framework. However, for the 40k matrix the larger amount of data to be transferred favors MPSF, and the rescheduling mechanism also leads to some improvement by handling the performance fluctuations.

Table 3: Results (run time in seconds) for WAVG on Scenario A with vector size N of 500 million elements (8 GB of total footprint), 5 billion elements (80 GB), and 50 billion elements (800 GB).

#	Algorithm	500M	5G	50G
1	Single/Serial	28.51	898.07	-
2	Single/Parallel	4.12	124.84	-
3	Naive/VCPU	0.37	11.10	497.28
4	Naive/Core	0.58	17.40	716.88
5	Naive/Node	2.91	97.78	3,920.20
6	MPSF no-resc.	0.40	12.01	362.81
7	MPSF	0.41	12.59	384.97

Finally, the results for WAVG on Scenario A are summarized in Table 3. For vectors of size 500M and 5G the core-based and VCPU-based naive approaches even outperforms MPSF. For larger data sizes the I/O component becomes relevant and only MPSF is able to attain to the natural increase in number of operations. Compared to the best result obtained via naive approaches, MPSF provides a performance gain of 27.0%, with a speedup factor of 1.37x.

5.2. Scenario B

Scenario B combines the resources from Gauss (private cloud) and Vulcan (public cloud). MPSF was configured to either allocate one container per VCPU or one container per core, depending on the benefit of running tasks on memory, without having to use disk I/O. The benchmark selected for the experiments was GEMM. The input is located in the Gauss cluster, and the results must be stored there too. Table 4 shows the results comparing GEMM for scenario A and B.

Table 4: Results (run time in seconds) for GEMM. Table compares results for 40k matrices running on Scenarios A and B.

#	Algorithm	A	B
4	Naive/Core	128.93	88.26
6	MPSF no-resc.	122.35	70.68
7	MPSF	117.41	71.12

The average compute rate is 153.6 MB/s for Gauss and 1,474.8 MB/s for Vulcan. Moreover, the effective bandwidth (also collected by the framework during execution and considering fluctuations) between the clusters is 128.12 MB/s. The data distribution reported by the framework is around 56% for Gauss and 44% for Vulcan. The improvement over the Naive/Core approach is of 19.9%, with a 1.25x speedup.

5.3. Scenario C

Scenario C combines a private cloud built on top of the Gauss nodes to another private cloud in a different geographical location using the Mamona nodes. To test this setup the data source was configured in one of the Mamona nodes and the GEMM benchmark was executed. Moreover, a restriction was set in the experiment to force the result of the operation to be sent to a Gauss node.

Table 5: Results (run time in seconds) for GEMM, 40k matrices running the workflow only on Mamona nodes *vs.* also using the Gauss nodes.

#	Algorithm	A/Gauss	A/Mamona	C/+Gauss
4	Naive/Core	128.93	7,098.21	5,443.31
6	MPSF no-resc.	122.35	6,172.35	3,601.54
7	MPSF	117.41	5,075.12	3,330.68

Table 5 shows the results of running GEMM with 40k matrices. The new run times are much longer as the results must be sent over the network from Mamona to Gauss. To simplify the tests, only the Naive/Core approach was considered in the comparisons. Executing the workflow using Mamona nodes only and using the Naive/Core approach leads to a total run time of around 2h – 1h20min is spent just to send the results over the network to Gauss. For Scenario C Naive/Core simply distributes the load across all containers (one container per core) without considering the I/O costs. This leads to a total run time of around 1h30min, an improvement of 23.3% over the A/Mamona run time. However, using MPSF with the rescheduling capabilities leads to a run time of around 55min, 38.8% of improvement over the Naive/Core approach and 7.5% over MPSF without rescheduling.

6. Analysis

MPSF consistently delivers better performance over naive approaches that try to greedily explore the available resources. Internally, MPSF also provides rescheduling capabilities that are particularly useful for workflows with

long duration (pipeline longer than one hour). Based on high-level workflow descriptions such as the APEX Workflows [5], this is usually the case for scientific applications. Figure 5 shows a summary of the experimental results showing the relative run time of MPSF compared to Naive/Core.

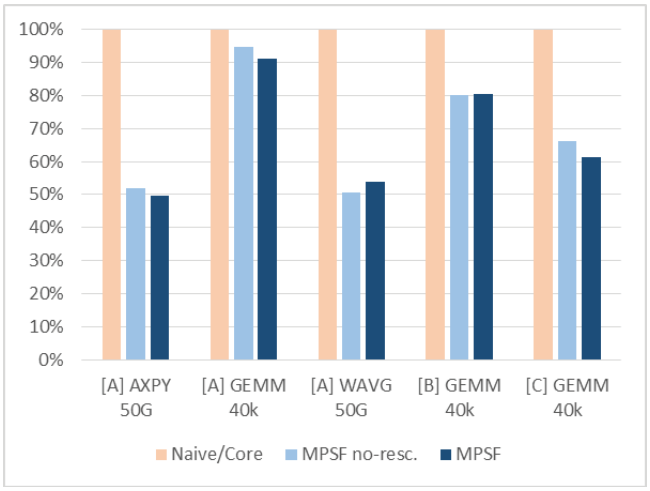


Figure 5: Summary of experimental results showing relative run time of MPSF compared to the Naive/Core approach. Figure shows scenarios (A, B, C), benchmark (*AXPY*, GEMM, WAVG), and problem size.

MPSF performs well especially for workflows wherein the data distribution constitutes an important component in the total time. For AXPY and WAVG MPSF shows a relative run time of around 50% compared to Naive/Core. Moreover, the experiments for scenarios B and C show that a scheduling algorithm that considers the time to transfer data, whether input or results, leads to much better performance.

7. Related Work

Topcuoglu et al. [7] proposed Heterogeneous Earliest-Finish-Time (HEFT) algorithm designed for makespan minimization. Computation cost is modeled by using estimates of the execution time to complete the task on a processor. Compared to HEFT, MPSF encompasses a more detailed view of resources and the performance characteristics of each workflow phase.

Proportional Deadline Constrained (PDC) [3] is the algorithm which selects cloud instances to meet deadlines while minimizing cost. Cluster Combining Algorithm (CCA) [4] is the algorithm which addresses multicore clouds by using a scoring approach. MPSF describes computational capacity as a function of the application, rendering more accurate predictions and a better initial load distribution. On the other hand, MPSF does not consider cost nor deadline constraints in its scheduling decisions.

For hybrid clouds Bittencourt et al. [2] proposed the Hybrid Cloud Optimized Cost (HCOC) scheduling algorithm, which reduces makespan by leasing resources from the public cloud to fit a deadline while maintaining a reasonable cost. MPSF, in contrast, focuses on maximizing performance without initially considering cost, although the allocation threshold avoids MPSF to become greedy.

Avresky et al. [8] presented a framework based on machine learning techniques to predict failures cause by anomalies during the execution of applications. Their work focuses on the reliability part of scheduling, while MPSF focuses on balancing the load sent to each node.

8. Considerations and Future Work

MPSF, Multi-Phase Proactive Scheduling Framework, defines a set of components to read and continuously monitor the resources of systems, and then distribute the workload based on a high-level description of the systems and the characterization of the workflow. The framework is able to predict this workload by calculating the optimal run time based on the distribution of the input data to the containers configured across the systems. The framework provides components to reschedule a workflow based on the performance fluctuations of the system. Finally, the framework is able to consider the I/O time component, such as data transfers among containers and between different clouds (Scenarios B and C). Experimental results indicate run time improvement of over 2x compared to greedy approaches, especially for larger input data sets and longer computations. Regarding limitations, the automatic detection and analysis of different algorithm implementations (or binaries) and the mechanism to choose the best alternative require these alternatives to be represented in the workflow description. Future work includes the definition of a reliability metric based on the Mean Time Between Failures (MTBF) of nodes and containers, then comparing these values to the makespan of a workload and to the remaining time to failure [8].

References

[1] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3373–3418, 2015.

[2] L. F. Bittencourt and E. R. M. Madeira, "HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011.

[3] V. Arabnejad and K. Bubendorfer, "Cost Effective and Deadline Constrained Scientific Workflow Scheduling for Commercial Clouds," in *Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on*, Sept 2015, pp. 106–113.

[4] A. Deldari, M. Naghibzadeh, and S. Abrishami, "CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud," *The Journal of Supercomputing*, pp. 1–26, 2016.

[5] "APEX Workflows," http://www.nersc.gov/assets/apex-workflows-v2.pdf, March 2016, accessed: 2016-07-25.

[6] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *3rd Workshop on Workflows in Support of Large-Scale Science*. IEEE, 2008.

[7] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, Mar 2002.

[8] D. R. Avresky, P. Di Sanzo, A. Pellegrini, B. Ciciani, and L. Forte, "Proactive scalability and management of resources in hybrid clouds via machine learning," in *Network Computing and Applications (NCA), IEEE 14th International Symposium on*. IEEE, 2015, pp. 114–119.

Task Based Load Balancing for Cloud Aware Massively Multiplayer Online Games

André Pessoa Negrão, Luís Veiga and Paulo Ferreira
Distributed Systems Group, INESC-ID
Instituto Superior Técnico, Universidade de Lisboa
andre.pessoa@tecnico.ulisboa.pt,{luis.veiga,paulo.ferreira}@inesc-id.pt

***Abstract*—In this paper we propose a task based load distribution framework for Massively Multiplayer Online Games running in hybrid cloud environments. Our solution breaks down high level tasks into subtasks in such a way that i) *core subtasks* (those with strong timing constraints) are executed at private resources owned by game operators; while ii) *background subtasks* (those with looser timing/reliability constraints) can be offloaded to temporary resources acquired from a public cloud. Our approach is lightweight and allows for faster deployment of newly acquired servers, making it more suitable for temporary overload situations. We present evaluation results confirming our solution as a viable alternative to traditional strategies.**

I. Introduction

The popularity of a Massively Multiplayer Online Game (MMOG) is hard to predict at deploy time. In addition, MMOGs have a highly dynamic active user population, which experiences variations in number even over short periods of time. These two factors combined make it difficult for MMOG operators to anticipate the exact amount of resources necessary to efficiently provision the game. As a result, operators tend to adopt pessimistic measures by deploying static and large infrastructures in which the number of resources is based in worst case predictions of load. The result is an *over-provisioned* environment in which a number of resources are idle for most of the time [1].

In light of this, several authors [1], [2], [3], [4], [5], [6] have proposed alternatives to over-provisioning based on Cloud Computing (CC). By taking advantage of the elasticity of CC [7], MMOG operators can deploy more conservative infrastructures and resort to the cloud to acquire resources according to the actual load experienced at runtime. As a result, the available resources are used more efficiently and the costs of deploying and supporting the game are reduced, benefiting every stakeholder in the MMOG environment.

While *cloud aware* resource provisioning promises to improve the cost-effectiveness of MMOGs, existing solutions employ the traditional technique of partitioning/replicating the virtual world on a fully operational game server deployed at the newly acquired resource. This approach has two main drawbacks. First, adding a new replica/partition to the system entails reconfiguration costs (e.g., data migration) at deploy time and heavy synchronization costs during the time the acquired server is up. Second, given the unreliable nature of the cloud, deploying fully operational servers might not be an acceptable solution for some MMOG companies.

978-1-5090-3216-7/16/$31.00

In this work, we propose a workload distribution solution for cloud-aware MMOGs inspired by task based computing. In our solution, when an overload event is identified, regular high level tasks that are executed by the overloaded server are broken down into subtasks that can offloaded to different resources. Tasks are partitioned so that i) subtasks with strong timing constraints or that require reliable game data are kept at the original server or offloaded to reliable resources, while ii) subtasks with less stringent timing requirements (*background tasks*) can be offloaded to temporary resources acquired from the cloud or to lightly loaded active resources.

The main advantage of this approach is that background tasks can be designed to operate over abstract data and execute low priority operations, while critical tasks/data are concentrated on reliable, long term resources. These characteristics make task partitioning more suitable for temporary overload situations (it introduces lower overhead at deploy time) and for execution in unreliable environments (it promotes the use of unreliable resources only for less critical operation).

The paper is structured as follows. Section II describes our task based load distribution framework. Section III presents implementation information. Section IV discusses the experimental results obtained. Section V overviews the related work. Finally, Section VI concludes the paper.

II. Task Partitioning

In this section we describe our task partitioning framework in detail. Without loss of generality, we discuss the process as involving two servers, the *main* (or *master*) server and the *task* (or *worker*) server. The master is the server that requests load distribution in order to have its workload lightened. The worker is the server that absorbs the extra workload coming from the master (in the form of tasks).

A. Definition and Offloading

The task partitioning process consists in breaking down a high level task that is executed as a whole at a single server into multiple tasks: one *core* subtask and one or more *background* subtasks. By default, the core task continues its execution at the main server. Background tasks, on the other hand, are meant to be offloaded to cloud resources.

The core subtask should be designed to keep responsibility for executing the most time critical operations, as well as those that require application specific information. Background tasks, on the other hand, are designed to execute operations with weaker timing and reliability requirements. In addition, background tasks should be designed to be *game-independent* – their execution should not require the installation of a copy of the game, it should depend only on abstract data (e.g., standard geometric data, such as positions and areas).

This separation of concerns between core and background tasks allows a system facing an overload situation to avoid/minimize the need for expensive code and/or data transfer. The main server retains the main responsibilities for user and data management; background tasks execute less critical operations over abstract, non-sensitive data, making them more suitable to be offloaded to unreliable resources, such as those obtained from the cloud. They can, however, also be offloaded to private resources, both new and active: because task partitioning allows the high level task to be partitioned into subtasks with arbitrarily small sizes, applications can more easily take advantage of lightly loaded active resources.

While having independent background tasks is ideal for the scenarios we envision, forcing this requirement on their design would be too restrictive. This way, application designers can also design *dependent background tasks* – background tasks that need a full copy of the game data (or a reliable partial copy). This solution gives application programmers more freedom and still allows applications to benefit from the advantages task partitioning provides in terms of separation of concerns between reliable and unreliable resources. On the downside, it has the potential of preventing the fast server loading that independent background tasks allow. A workaround to this is to design dependent subtasks with an independent *loading phase*. This way, when they are offloaded to a new server they can be executed while the application loads onto the new machine. When loading is complete, the subtask can be upgraded to its dependent phase.

B. Task Characterization

In state partitioning and state replication, the effects of adding a partition/replica are general and mostly independent of scenario: it results in a distribution of load and server↔client bandwidth at the expense of additional server↔server communication. When it comes to task partitioning, the arbitrary and application specific nature of the process means that each task partitioning strategy may have a different impact on each different resource. For example, one task partitioning policy may only reduce computational load; a different policy may have a higher impact on bandwidth, while still influencing CPU usage.

For the load distribution process to efficiently make use of task partitioning, each task partitioning strategy must have a *task descriptor* that characterizes its effects on the relevant resources. In particular, the task descriptor should describe which resources will experience a load reduction at the master and which resources will have a load increase at the worker(s). The task descriptor can also contain a quantification of these increases/reductions, although the reliability of the quantifications is difficult to ensure due to the dynamics of the execution environment.

In addition to task characterization, servers also need to be defined as *core* or *background*, in order for the system to appropriately offload tasks. By default, private resources are considered core and take precedence for the execution of core tasks. Core resources can, however, also execute background tasks if necessary.

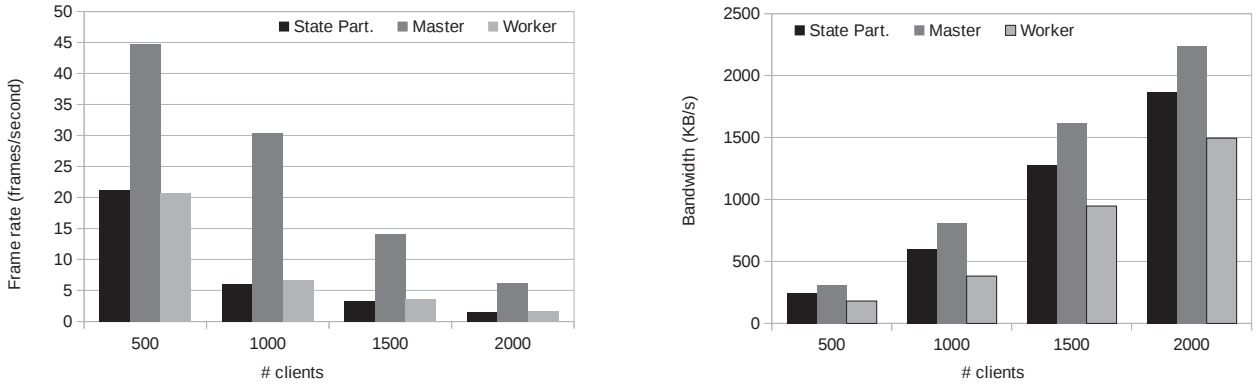
Public resources, on the other hand, are considered background by default. Individual public resources can, however, be promoted to the core group if application operators so decide. Promotion can be done temporarily *on-the-fly* (e.g., if core resources are necessary, but the system no longer has private resources available) or permanently, if operators see no reason to distinguish between public and private resources.

III. Implementation

Our task based solution is an extension to the load balancing component of CloudDReAM [8], our middleware for cost-effective execution of MMOGs in the cloud. CloudDReAM follows a hybrid cloud architecture in which virtualized servers can be acquired from public cloud providers, as well as from a private cloud run by the operators of the MMOG. To manage server→client traffic CloudDReAM uses Interest Management [9], which consists in sending to each user only updates that refer to relevant objects – typically those within an *Area of Interest* (AoI) surrounding the player's *avatar*. In particular, we use our own Vector-field Consistency (VFC) model [10], which divides the AoI into multiple zones such that: updates to objects in the inner zone are sent to the user immediately; as the distance increases, updates are propagated at increasingly lower frequencies.

Within CloudDReAM, we implemented a few default task partitioning strategies. Due to space constraints, in this paper we focus on the two main strategies: matching and update propagation partitioning. *Matching* is one of the main tasks executed by a game server. It consists in identifying, for each user, which objects are within the user's AoI and, of those, which ones have been modified (and, consequently, need to be propagated to the user). The matching process naturally lends itself to partitioning into two subtasks: the first subtask identifies which objects are within each client's AoI; the second task identifies which of those objects have been updated and need to be sent to the user.

Of these two tasks, the first one can safely be executed at a task server, which becomes responsible for identifying and informing the main server of modifications to each user's AoI. To make sure that the natural synchronization delays between the master and the worker do not result in missed interactions, the worker monitors and informs the master about each user's *extended* AoI – the actual AoI, plus an additional area surrounding it. The master can, this way, keep track of the objects in the nearby area and timely identify new additions to any AoI.



(a) Frame rate.

(b) Upload bandwidth.

Figure 1. Performance comparison between state and task partitioning.

To partition update propagation, we take advantage of the multi layered characteristics of VFC by having the task server propagating updates that refer to the less critical consistency zones. Updates regarding high priority zones continue to be managed by the master.

IV. Evaluation

In this section we analyse how our solution fares against state partitioning. To evaluate our solution, we deployed a 9 machine cloud from which new servers (VMs) are acquired as necessary. We compare the performance of the system when all servers are state partitioned against a version of the system mixing task servers and state partitioned servers. For clarity, we focus our evaluation on partitions that in one case are state partitioned and the other are task partitioned. Thus, our results show a pairwise comparison between the two strategies: a pair of partitioned servers *versus* a main and a task server.

Throughout the analysis, we show the results obtained with a varying number of clients. Clients were simulated by bots, each controlling a single avatar that explores the game map. We deployed clients evenly across the game map, resulting in approximately the same number of clients in each partition; as a result, the performance of the state partitioned server is very similar. For this reason the figures show the averages of the values obtained by each pair of state partitioned servers. By contrast, due to their inherently asymmetrical relation, we show the results of the master and worker servers separately. The number of clients presented in each figure corresponds not to the total number of clients in the system, but to the number of clients shared by each pair of servers.

Figure 1(a) shows the performance differences between state and task partitioning in terms of frame rate. *Frame rate* measures how frequently the server is able to execute the matching algorithm and send updates to clients. It is a crucial high level metric that determines if the application is able to provide its required quality of service.

As shown in the figure, the performance improvements of offloading the matching task enable the system to ease the natural degradation in frame rate that occurs as the number of clients increases. Our solution is able to achieve frame rates higher than 10 fps for up to 1500 clients, delivering optimal frame rates for slow paced games and suitable for most types of games. At 2000 clients, frame rates drop to approximately 6 fps, which, while not generally suitable, are still enough for many types of games, especially under high load. The partitioned system, on the other hand, experiences a clear sharp decrease with more than 500 clients.

The frame rates of the worker, on the other hand, are considerably lower than the master. This is due to the fact that the worker has to execute the two parts of the matching algorithm: 1) identifying if an object is within the AoI of another and, if so, 2) identifying the specific VFC consistency zone of such object. The worker, however, is responsible for propagating updates that refer to the lower priority zones, which have weaker requirements in terms of propagation frequency. Considering the values achieved, it is unlikely that the lower frame rates of the worker prevent the timely propagation of such lower priority updates.

As Figure 1(a) shows, the worker is, nevertheless, able to obtain higher frame rates than the state partitioned servers. The main reason for this is that as the number of users increases, update processing takes on a larger role at those servers, due to conflict detection and synchronization with neighbour servers, among other application specific issues. The worker, on the other hand, does not handle such issues, because it only deals with lower priority updates and does not participate in neighbour synchronization.

Figure 1(b) shows the upload bandwidth results obtained. The results show that the master is able to offload between 35%-40% of its bandwidth requirements to the worker. These values put the maximum per-server upload bandwidth requirements of the task based solution, at most, 15% above those of the state partitioned system, which has an approximately even ratio between each pair of servers.

Note that the balanced distribution between state partitioned servers only occurs because players have been evenly dis-

tributed across partitions. In practice, however, this situation is highly unlikely and differences in load between partitioned servers occur naturally. In fact, if the difference increases past a certain threshold (due to hotspots), one of the servers becomes overloaded. Overcoming this situation requires acquiring a new server or, at least, redistributing the load among active servers. In any case, this it involves expensive server reconfigurations, adding more complexity to the system.

Task partitioning, on the other hand, is less exposed to hotspots because workload partitioning between the master and the server is mainly determined by *AoI density* (i.e., the average number of objects inside a user's AoI), whereas in partitioned systems the main factor is *partition density*. Hotspots impact both of these factors, but have a stronger influence on partition density, which depends exclusively on the number of players in the partition. AoI density, on the other hand, additionally depends on user behaviour and increases at a slower rate.

V. Related Work

Nae *et al.* developed some of the most seminal work on cloud aware resource provisioning for MMOGs [11], [1]. Among their main contributions is a neural-network load prediction algorithm that identifies the resources necessary to provision a game environment over time. Marzolla *et al.* [5]propose using a Queueing Network model to balance load among servers in a multi-tier cloud infrastructure. Najaran *et al.* [2] propose renting game servers from the cloud and organize them in a P2P network. Glinka *et al.* [4] also proposed a cloud aware infrastructure, but organize servers as a replicated cluster.

The common factor between these solutions is that they are based on traditional state partitioning/replication workload distribution strategies, which have some important limitations. First, they require full game servers, which take time to become fully operational in the cloud. Load prediction may minimize this problem, but makes the system too dependent on its accuracy. Second, they involve server reconfiguration and synchronization costs that are not suitable for temporary (and, potentially, short-term) overload situations. Finally, they expose application data to the unreliable resources of the cloud, which might not be acceptable for some game companies.

Our solution tackles these issues by allowing fast deployment of servers executing background tasks, which, by design, are also more appropriate for execution in unreliable resources. In addition, the task partitioning process is more stable, as it maintains the most critical operations and data at the original server.

Lim and Lee [12] have previously proposed a task based load distribution scheme for MMOG-like environments. Their solution allocates fine-grained, message-level tasks to servers on a cluster at runtime according to CPU and network load. However, their solution assumes a static infrastructure and requires servers to be connected through high-speed networks, which contrasts with the cloud environments our work targets.

VI. Conclusions

In this paper we proposed a task-based load balancing architecture for Massively Multiplayer Online Games running in cloud environments. Our system breaks down high level server tasks into subtasks of arbitrary size and priority that can be offloaded to other servers. Critical tasks are meant to keep executing at the original server, while lower priority tasks are designed to be offloaded to public cloud resources. In comparison with the load distribution strategies typically used in MMOGs, our solution is faster to deploy (making it more suitable for temporary overload situations) and more appropriate for execution in unreliable cloud resources. In addition, it is general enough to allow applications to freely define how and which tasks should be partitioned, as well as the conditions for their offloading.

Acknowledgments: This work was partially supported by national funds through FCT – Fundação para a Ciência e Tecnologia with reference UID/CEC/50021/2013.

References

[1] V. Nae, A. Iosup, and R. Prodan, "Dynamic resource provisioning in massively multiplayer online games," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 3, pp. 380 –395, march 2011.

[2] M. T. Najaran and C. Krasic, "Scaling online games with adaptive interest management in the cloud," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, ser. NetGames '10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 9:1–9:6.

[3] E. Carlini, M. Coppola, and L. Ricci, "Integration of p2p and clouds to support massively multiuser virtual environments," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, ser. NetGames '10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 17:1–17:6.

[4] S. Gorlatch, D. Meilaender, A. Ploss, and F. Glinka, "Towards bringing real-time online applications on clouds," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*, 2012, pp. 57–61.

[5] M. Marzolla, S. Ferretti, and G. D'Angelo, "Dynamic resource provisioning for cloud-based gaming infrastructures," *Comput. Entertain.*, vol. 10, no. 3, pp. 4:1–4:20, Dec. 2012.

[6] C.-F. Weng and K. Wang, "Dynamic resource allocation for mmogs in cloud computing environments," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, 2012, pp. 142–146.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[8] A. P. Negrão, M. Adaixo, L. Veiga, and P. Ferreira, "On-demand resource allocation middleware for massively multiplayer online games," in *Proceedings of the 2014 IEEE 13th International Symposium on Network Computing and Applications*, ser. NCA '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 71–74.

[9] K. L. Morse, "Interest management in large-scale distributed simulations," University of California, Irvine, Department of Information and Computer Science, Technical Report ICS-TR-96-27, Jul. 1996.

[10] L. Veiga, A. Negrão, N. Santos, and P. Ferreira, "Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency," *Journal of Internet Services and Applications*, vol. 1, pp. 95–115, 2010.

[11] V. Nae, R. Prodan, and T. Fahringer, "Cost-efficient hosting and load balancing of massively multiplayer online games," in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, 2010, pp. 9–16.

[12] M. Lim and D. Lee, "A task-based load distribution scheme for multi-server-based distributed virtual environment systems," *Presence*, vol. 18, no. 1, pp. 16–38, 2009. [Online]. Available: http://dx.doi.org/10.1162/pres.18.1.16

DARSHANA: Detecting Route Hijacking for Communication Confidentiality

Karan Balu Miguel L. Pardal Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Lisbon, Portugal
{karan.balu, miguel.pardal, miguel.p.correia}@tecnico.ulisboa.pt

***Abstract*—The Border Gateway Protocol (BGP) plays a critical role in the Internet providing connectivity to hosts across the world. Unfortunately, due to its limited security, attackers can hijack traffic by generating invalid routes. Some detection systems for route hijacking have been presented, but they require non-public information, high resources, or can easily be circumvented by attackers. We propose DARSHANA, a monitoring solution that detects route hijacking based solely on data-plane information, and has enough redundancy to prevent attacker countermeasures such as dropping of traceroute probes. DARSHANA uses active probing techniques that enable detection in near real-time. By using diverse methods, DARSHANA can still detect attacks even if the adversary manages to counter some techniques. We show that our solution allows effective detection of many hijacking attacks by emulating them using PlanetLab and Amazon AWS.**

I. Introduction

The Internet is a network composed by many interconnected networks. Administrative network domains are called *Autonomous Systems (AS)*, and the routing between these autonomous systems is handled by the *Border Gateway Protocol (BGPv4)* [1]. Each AS contains one or more *Internet Protocol (IP) prefixes*, whereas each prefix is an identifier for a subnetwork. If some AS wants to provide connectivity between its IP prefixes and other ASes, it will announce those prefixes to those ASes. Each AS contains one or more routers configured with BGP, known as BGP speakers. Each speaker contains forwarding tables that provide the information necessary to forward a packet based on the destination and the prefix available in the table. BGP speakers send UPDATE messages to other BGP speakers in order to announce or withdraw routes. Upon receiving these update messages, an AS selects the best route to a certain prefix based on its internal policy.

Although BGP plays an essential role in the Internet, it has considerable limitations in terms of security. One example of its lack of security happened on August 2013 when a company called Hacking Team helped the Italian police regain control over computers that were being monitored by them. Hacking Team worked with an Italian Web host called Aruba announcing to the global routing system 256 IP addresses that it did not own. This caused all the traffic directed to the 256 IP addresses to be redirected to the Hacking Team. This was the first known case of an ISP performing a *route hijacking attack* intentionally [2].

978-1-5090-3216-7/16/$31.00

These security problems mainly come from the potential to interfere with route announcements in order to corrupt BGP routing. Attackers can exploit this vulnerability to claim ownership of victim prefixes and announce them to their upstream providers. Providers that do not verify the origin of the announcements may end up injecting these in to the global routing system, which leads network packets to reach incorrect destinations. In some cases, attackers may intercept traffic and forward it to its destination, compromising *confidentiality* without being noticed.

The vulnerability of the BGP protocol has been well-known for over two decades. Several solutions have been proposed, but none is widely adopted and deployed. These solutions mainly fall into filtering and cryptography methods [3]–[6], which require changes in routers configurations, router software and a public key infrastructure. Others proposals [7]–[10] rely on passive monitoring of BGP data, so they are easier to deploy; however, they suffer from high false positive rate, since they access public registries that are frequently outdated. Finally, there are systems that use only data-plane information, by executing active probing, but can easily be bypassed [11], or require vantage points [12].

We present DARSHANA (or DaRsHANa, from Detecting Route HijAckiNg – in Sanskrit, Darshana means to see, vision or glimpse) that works by continuously observing network information to detect route hijacking attacks. Our main goal is to detect if Internet traffic is diverted to be eavesdropped in arbitrary places around the world, when the adversary has no access to the path normally taken by the traffic. Therefore, the security property we are most interested in is communication *confidentiality*. DARSHANA has the advantage of being implemented in the data-plane, above the OSI network layer, therefore it can be implemented in terminals connected to the Internet, instead of being specific to Internet Service Providers (ISP) and large Internet companies. DARSHANA uses a set of monitoring techniques like: traceroute, latency measurements and IP traceback mechanisms that can effectively monitor the routes that packets are taking. Ultimately, this allows detecting route hijacking that could be used to eavesdrop on a communication to break confidentiality. We do not intend to substitute the use of best practices to configure BGP, or several prevention mechanisms that have already been proposed like [3]–[6], [8], [13].

The system applies active probing techniques which enables the detection in near real-time. The order of execution of these

techniques is defined in terms of overhead and reliability: techniques with lower overhead and reliability are executed more often; when needed, heavier, more reliable techniques are used. The system does not depend solely on a specific technique to be able to accurately detect attacks.

We performed an experimental evaluation by deploying nodes in PlanetLab and Amazon AWS. We show that nodes can identify when traffic is being hijacked, although this is more difficult if the hijacker is close to the source.

The contributions of this paper are three-fold. First, we propose the design and implementation of DARSHANA, a route hijacking detection system that is accurate, does not need access to privileged information, does not require changes in routers software, is redundant enough to deal with attacker countermeasures, and does not need vantage points. Second, we present a new mechanism that uses the propagation delay in order to detect route hijacking. Third, we analyze and conduct experiments in large scale environments to evaluate DARSHANA.

II. Background

BGP does not ensure that BGP routers use the AS number they have been allocated, or that the ASes holds the prefixes they originate. Therefore, a router can be configured to advertise a prefix from an address space belonging to another AS in an action known as route hijacking or IP prefix hijacking [13]. This action can happen in the following forms:

Hijack the entire prefix. The hijacker announces the exact prefix of the victim, meaning that the same prefix has two different origins.

Hijack only a sub-prefix. The offender announces a more specific prefix from an already announced prefix (e.g., the victim announces 200.200.0.0/16, the attacker 200.200.200.0/24). Due to the longest prefix matching rule, ASes that receive these announcements may direct traffic towards the wrong AS.

These forms of attacks can impact routing, leading to:

Blackhole. An AS drops all the packets received. The Pakistan Telecom / YouTube incident originated a blackhole where all the traffic sent to YouTube was redirected to Pakistan Telecom. Since there was no working path back to YouTube, Pakistan Telecom was forced to drop all packets [14].

Interception. The attacker announces a fake route to an AS, that forwards traffic of the victim to the original server. The contents of the intercepted traffic can be analyzed/changed, before sending it to the legitimate destination [15]. This type of attack requires an untampered working path that will route the traffic back to the legitimate destination.

BGP security procedures today consist mainly on filtering suspicious BGP announcements, e.g., announcements that contain loopback addresses or addresses that are not owned by the AS that announced it. The problem of this approach is that detecting invalid route announcements is more challenging when the offending AS is several hops away.

An accurate routing registry would have prefix ownership, AS-level connectivity and routing policies enabled in each AS, helping ASes to verify the legitimacy of the advertisements that they receive. The drawbacks of this model mainly include, the lack of desire of ISPs to share their proprietary routing policies.

In this work, we focus on interception attacks and propose a solution that does not rely completely on Internet registries.

III. DARSHANA

We introduce DARSHANA a route hijacking detection system that uses only data-plane information.

A. Mechanisms used in the system

This section presents the mechanisms used in DARSHANA. We indicate the short names we use for each between parentheses (e.g., Lat for the first mechanism). Table I shows a summary of the mechanisms.

1) Monitoring network latency (Lat): One of the metrics used in our system is the RTT (round trip time). Each node that is monitoring another (node) keeps information about the total time that each packet takes from source to destination and from destination to source. In a hijacking event the end-to-end latency between a certain source and a destination tends to change significantly. Measuring the RTT has some benefits like low overhead and the fact that time is a factor that is hard for an attacker to evade. On the other hand, an increase in RTT is hard to distinguish from network congestion.

We designed a new version of ping that we denote as *cryptographic ping*. The objective is to avoid having an adversary respond to a ping request earlier, before the request reaches the destination, leading to readings of RTT that are lower than the real value. The new mechanism works as follows. The machine that is monitoring A marks time and sends a nonce to a machine that is being monitored B. B will cipher the nonce with its private key and send it back. A marks the time again and will verify the received signed nonce by applying the public key of B. If the nonce matches, A calculates the round trip time by subtracting the first marked time from the last marked time. Without this ping, the hijacker, since he has hijacked the traffic, could answer to the ping probes sooner, ultimately fooling the system. This way we can guarantee authenticity and uniqueness. This requires the server to run code and share its public key.

2) Estimating hop count (Hop): We propose adding the hop count, the number of intermediate devices between a source and a destination, as one more criteria to detect a route hijacking attack. According to [12] the hop count to a certain destination generally remains unchanged over time. When a prefix is hijacked, the hop count tends to change. In an interception attack, the traffic takes a detour to the AS of the hijacker, then it is forwarded to the legitimate destination. This deviation can change significantly the hop count if the hijacker is far from the source, which is likely due to the size of the Internet. In contrast to the RTT, the hop count is not affected by congestion. However, other less frequent events

TABLE I
THE METHODS, BENEFITS AND DRAWBACKS OF THE MECHANISMS USED BY DARSHANA PRESENTED IN SECTION III-A

Mechanism	Detection	Benefits	Drawbacks
Monitoring network latency (Lat)	High latency could mean traffic hijack.	Easy to measure.	Latency is also affected by congestion so it does not indicate hijacking with certainty.
Estimating hop count (Hop)	The hop count is usually stable, so a high increase in hop count could be induced by traffic hijack.	Usually stable.	Link failures and legitimate route changes may trigger alteration in the network topology.
Calculating path similarity (Path)	Paths may end up showing significant disagreement when traffic is hijacked.	Filters small legitimate route changes.	Not all route changes are the result of traffic hijack.
Monitoring propagation delay (Prop)	Propagation delay gives the time that a bit takes in the wire, meaning that in a hijacking event this value may show an anomolous value.	Provides insights about the attack even when traceroute does not give results.	Requires a period of initialization, to estimate all the other latencies.

like link failures and operational route changes may cause it naturally.

3) Calculating path similarity (Path): The system tracks the path that packets are following. It periodically stores the path obtained using traceroute and translates the IP addresses found to autonomous systems numbers (ASN). This mapping increases accuracy, because we only need one router from a autonomous system to correctly obtain a path that packets are taking. The correlation between the new path measurement and the previous path measurement may provide insights about the occurrence of the attack. In a hijacking event, since the traffic has taken a detour, the paths measured may end up showing significant differences. The level of this difference sets apart legitimate route changes and hijacking situations. On the contrary, legitimate changes are not expected to result in a dramatic route change.

4) Monitoring propagation delay (Prop): We propose a new technique that isolates the propagation delay from the RTT and uses this metric to declare a route hijacking. This technique is divided into two phases. The second phase is activated only if the system stops obtaining results from the Path mechanism, indicating an attacker is interfering with this mechanism.

Phase one. Consider that the RTT can be decomposed in the following delays: transmission delay (σ_{trans}), propagation delay (σ_{prop}), queuing delay (σ_{queue}) and processing delay (σ_{proc}) as depicted in $RTT = \sigma_{trans} + \sigma_{prop} + \sigma_{queue} + \sigma_{proc}$. The propagation delay is the time that a bit takes in the communication medium from a node to another node. This delay can be calculated as the ratio between the link length and the propagation speed on that medium.

The system uses the IP addresses of the origin and the destination to obtain their approximate geographical coordinates. The link length is calculated by computing the shortest distance between both. For the propagation speed, we make a conservative approximation by considering that all nodes are connected with fiber-optics, which has higher propagation speed than alternative media (copper, air). We use the usual approximation that fiber optics operates at 2/3 the speed of light [16]. The minimum possible propagation delay is given by formula 1, where *o* represents the origin, *d* is the destination and *c* is the speed of light:

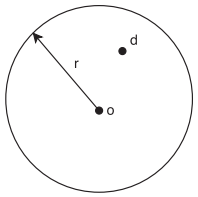


Fig. 1. The maximum propagation delay is represented as a circle defined by the source *o* with radius *r* where the destination *d* is inside of the circle.

$$\sigma_{prop} = \frac{3}{c} \times ShortestDistance(o, d) \quad (1)$$

Besides the propagation delay, the system estimates the sum of the others latencies ($\sigma_{trans,queue,proc}$) by $\sigma_{trans,queue,proc} = RTT - \sigma_{prop}$.

Phase two. When the system obtains an anomalous RTT and stops receiving results from Path, it selects the minimum value of $\sigma_{trans,queue,proc}$ and the maximum value of the RTT estimated. By $max(\sigma_{prop}) = max(RTT) - min(\sigma_{trans,queue,proc})$, we obtain an upper bound on the value of the propagation delay. This allows drawing a circle around the source with a radius *r* that represents the maximum propagation delay (represented in Figure 1). If the distance between *d* and *o* is greater than *r* we detect a route hijacking. This mechanism allows detecting route hijacking even if the Path measurements cease to exist. However, it requires a period of initialization, to estimate the different latencies.

B. System operation

This section presents the operation of DARSHANA. Figure 2 divides the mechanisms presented in the previous section in components and presents their relations. DARSHANA has the following components:

Active Probing. In this component three mechanisms come into play: Lat, Hop, and Prop (first phase). The system constantly takes values for RTT, hop count and the path that packets are taking. The system probes the RTT more often because this is the mechanism with the lowest overhead. Upon detecting an anomaly in the RTT the system passes to more reliable mechanisms, as this anomaly could be caused by temporary congestion in the network. The next mechanism

is estimating the hop count, for the reasons explained in Section III-A2. This metric is more reliable than Lat so it is used to filter out small legitimate changes. This component also executes the first phase of monitoring propagation delay, calculates this delay with the shortest distance in a straight line between the source and the destination and estimates the other latencies belonging to the RTT.

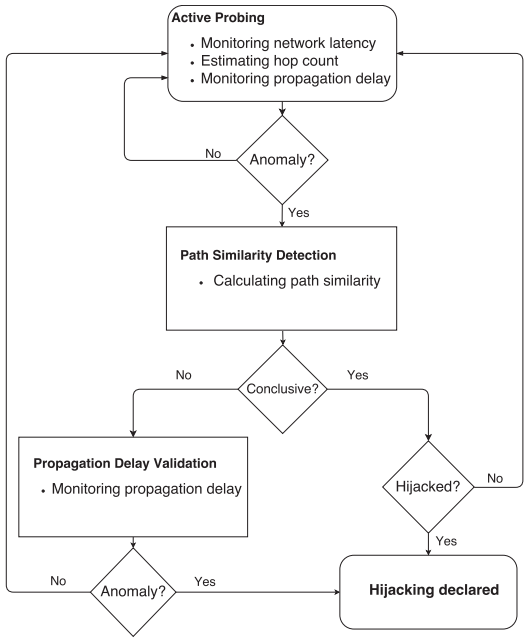


Fig. 2. Fluxogram of DARSHANA, with the mechanisms organized in different components

Path Similarity Detection (Path). Traceroutes with different protocols (ICMP, UDP, TCP) are issued. The system uses different protocols because routers may be configured to block certain protocols [17]. The path that contains the most nodes is chosen and stored. If enough results were received, then the new path will be compared with the last path obtained by the Active Probing. Disagreement above a certain threshold may indicate the existence of the attack.

Propagation Delay Validation (Prop, second phase). In case no conclusive results are received from path similarity detection, the $max(\sigma_{prop})$ and the $anomalous(\sigma_{prop})$ are calculated. The maximum propagation delay is computed by $max(\sigma_{prop}) = max(RTT) - min(\sigma_{trans,queue,proc})$. The anomalous propagation delay is calculated with $anomalous(\sigma_{prop}) = anomalous(RTT) - max(\sigma_{trans,queue,proc})$. This calculated propagation delay is compared with $max(\sigma_{prop})$.

Hijacking declared. Upon conclusion of the method chosen, an analysis is made and presented to the sender of the traffic.

C. The system in detail

In this section, we present the implementation decisions of DARSHANA.

Active probing. DARSHANA issues cryptographic pings and Paris traceroute [18] probes with different periods. Paris traceroute is known to evade anomalies like loops, cycles and diamonds. These anomalies stem from the fact that a load balancer sends probes of traceroute to different interfaces based on the header of the probes. By not varying the fields used by a load balancer, Paris traceroute enables probes to be forwarded in the same interface even if load balancers exist.

Three values are obtained by executing traceroute: hop count, traffic path and propagation delay. For calculating the hop count we use traceroute. We only need to execute a partial traceroute with a TTL that is close to the destination in the majority of times. TTL = 1 is only used when we do not know about the destination.

We characterize the traffic path in terms of a set of autonomous systems, so each node of the result of the traceroute is mapped to the corresponding autonomous system using the CYMRU database [19].

Finally, the propagation delay is calculated by first, translating the IPs of the source and destination to geographical coordinates using MaxMind database [20], then the shortest distance is calculated between them and passed to the propagation delay by using the formula presented in Section III-A4.

Each iteration of the cryptographic ping gives a new sample of RTT and by subtracting the RTT with the propagation delay, we estimate the other latencies of the RTT.

Path Similarity Detection. New samples of RTT and hop count are compared with the exponential weighted moving average of past samples, the formula for the average is the following: $sample = (1 - \alpha) \times sample + \alpha \times sample_{new}$. The moving average allows DARSHANA to adapt to the normal changes in the network. If the quotients between the new samples of both RTT and hop count with the exponential weighted moving average passes certain defined thresholds T_{Lat} and T_{Hop}, Paris traceroutes are issued to the destination in an attempt to reveal the cause of the anomalies. If there are enough elements in the resulting path, then this path is compared to the last path stored. The comparison of these two paths can be computed from $path$ and $path'$ using the Sorensen-Dice coefficient: $sim = 2|path \cap path'|/(|path| + |path'|)$. This gives the similarity in a number that ranges from [0,1]. 0 means that there is no similarity at all and 1 means that the items of the two paths are the same. If the similarity is below a threshold T_{Path}, then a route hijacking is declared.

Propagation Delay Validation. In case the traceroutes executed in the previous module do not produce any results, DARSHANA calculates the σ_{prop} with the RTT and $\sigma_{trans,queue,proc}$ that were estimated. More precisely, the system will compute the $max(\sigma_{prop})$, by subtracting the $max(RTT)$, found before the anomaly, and the $min(\sigma_{trans,queue,proc})$. This computed delay will be compared with $anomalous(\sigma_{prop})$ resulted from the subtraction of the $anomalous(RTT)$ with the $max(\sigma_{trans,queue,proc})$. If $\frac{anomalous(\sigma_{prop})}{max(\sigma_{prop})}$ is higher than a defined threshold T_{Prop}, then a route hijacking is declared.

IV. Evaluation

Simulations of prefix hijackings were conducted to validate our proposed implementation in terms of performance and cost. The objective of the experimental evaluation is to answer two important questions: (1) How effective is DARSHANA in detecting attacks? (Section IV-B) (2) How many times is DARSHANA forced to execute techniques with higher overhead in normal conditions when there is no attack? (Section IV-C)

The experiment was done in PlanetLab Europe [21] and AWS EC2 [22]. PlanetLab offers a geographically diverse set of nodes which provides more choice to build scenarios. However, the restriction of only being able to access nodes from Europe limits the testing in larger scale scenarios. AWS permits access to instances in different continents but does not provide much geographical diversity. With PlanetLab we use nodes from Portugal (POR), Ireland (IRE), France (FRA), Germany (GER) and Poland (POL). From AWS we used instances from N. Virginia (VA), N. California (NA) and South Korea (S. Korea). Figure 3 shows a world map with all the nodes used from PlanetLab and AWS marked in black circles and squares, respectively.

Fig. 3. Nodes used from PlanetLab and AWS

A. Simulating route hijacking attacks

Before we present the tests done, it is important to explain how the simulation of the attacks is made. We simulate only the interception attack, because the blackhole attack ends up being just an interruption of communication, therefore it is easy to detect. In order to simulate the attack, we need three nodes: one node that is the source of the Internet traffic; a node that will serve as the destination; and another node that is trying to hijack traffic by receiving it and then sending it to the legitimate destination.

B. Performance of the system

In order to evaluate the performance of DARSHANA, we measured the percentage of times that DARSHANA detects existing attacks and assess the false positives in different scenarios (i.e., false route hijacks reported). We compared DARSHANA with the individual mechanism: Lat, Hop, Path, Prop. Each scenario of the experiment was repeated 30 times.

1) Small scale scenarios: We tested small scale scenarios with nodes from PlanetLab. Each scenario is composed by three nodes. Two of them have a source-destination relation and the third one serves as the hijacker. Throughout the scenarios the source and the destination are fixed and the

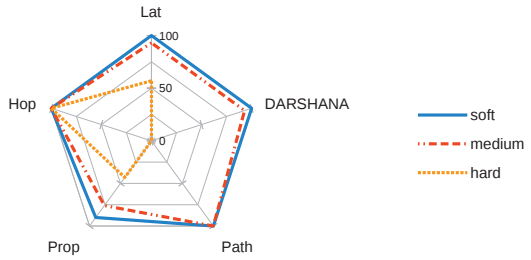


Fig. 4. The percentage of times that each mechanism detects a simulated route hijacking attack. The scenario involves Portugal as the source, Ireland as the destination and France as the hijacker. The labels soft, medium and hard represent different sets of thresholds for each mechanism.

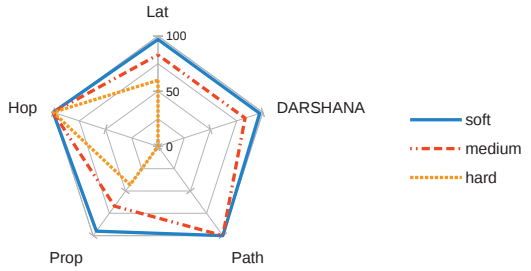


Fig. 5. The percentage of times that each mechanism detects a simulated route hijacking attack. The scenario involves Portugal as the source, Ireland as the destination and Poland as the hijacker.

hijacker varies its distance to the source. We selected a node from Portugal as the source, a node from Ireland as the destination and the hijackers are nodes from France and Poland. The distances from source to hijacker were chosen in a way that would enable us to determine the cases when DARSHANA has more difficulty in detecting the attack. The results are presented in Figures 4 and 5.

The figures show the percentage of times that each mechanism detects a simulated route hijacking attack under different scenarios. Each figure contains three labels: soft, medium and hard. They specify qualitatively the thresholds that were used for each mechanism. There are four thresholds, T_{Lat}, T_{Hop}, T_{Path} and T_{Prop}, that indicate how much measurements of RTT, hop count, path and propagation delay have to deviate in order to declare a route hijacking. The values of the thresholds used in the experiments were defined based on many experiments done before the evaluation here reported. These values are presented in Table II.

Observing the results, we can conclude that soft thresholds lead Lat to detect all hijacks. However, this also leads to false positives and, in the case of DARSHANA, prevents the other mechanisms from actuating and removing such false positives, while keeping a high detection rate. The Hop and the Path mechanism present 0 or 100% values. This is due to the fact that these mechanisms provide constant results through time. Therefore for certain values of T_{Hop} and T_{Path}, these mechanisms will detect or not the simulated attack. In regard

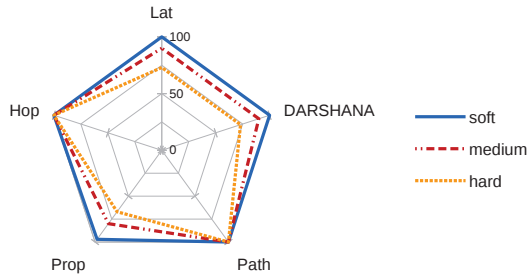


Fig. 6. The percentage of times that each mechanism detects a simulated route hijacking attack. The scenario involves N. Virginia as the source, N. California as the destination and Germany as the hijacker.

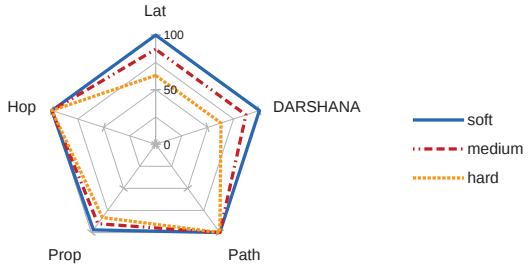


Fig. 7. The percentage of times that each mechanism detects a simulated route hijacking attack. The scenario involves Ireland as the source, Germany as the destination and South Korea as the hijacker.

to the propagation delay mechanism and observing Figures 4, 5 and Table II, the hard label in Figure 4 corresponds to T_{Prop} = 4.6 and the soft label in Figure 5 is equal to T_{Prop} = 7.4. In Figure 5 with the soft label, the detection of the attack is very close to 100%, but by observing Figure 4 we can see that for the hard label, this mechanism can only identify the attack less than 50% of the times. This means that changes in propagation delay are much more observable as the hijacker increases its distance to the source of the traffic.

When the source and the hijacker are close, the packets do not traverse many different autonomous systems and so DARSHANA is not able to detect the hijack with the hard thresholds.

2) Real scenarios: While our analysis in the previous section provided some insights about the capacity of detection of our system, we wanted to test our detection mechanism using historical prefix hijacking events and confirm that our mechanism behaves better by having the hijacker farther away. We simulated two scenarios. It was not possible to choose nodes from the exact locations in which these scenarios took place so we chose nearby nodes. The first scenario corresponds to the Belarusian Traffic Diversion [23], where traffic from New York was diverted to Belarusian ISP GlobalOneBel before arriving to the intended destination, Los Angeles. To simulate this, we deployed two nodes (micro-instances) in two different Amazon AWS regions: N. Virginia and N. California. The node from N. Virginia is the source, the node from N. California is the destination. We used a node from PlanetLab in Poland to serve as a hijacker and to represent the Belarusian ISP GlobalOneBel. The second scenario emulates the China 18-Minute Mystery [24], in which, allegedly, traffic between London and Germany took a detour through China. We simulate this by selecting a node from PlanetLab in Ireland as the source, a node from Germany as the destination and a micro-instance of Amazon AWS in Seoul as the hijacker. The results can be found in Figures 6 and 7. In these scenarios there is substantially more change, between the samples after attack and the samples prior to the simulated attack, than in the small scenarios experiments. The values for thresholds are shown in Table II.

To better understand why DARSHANA presents superior detection values in relation to the experiments done in Section IV-B1, we need to have an idea of the paths that packets take from source to destination, before the hijacking and after the hijacking. Table III shows the number of the ASes the traffic traverses, before the attack and after. It is possible to observe that the normal path and the hijacked path from the small scale scenarios share more numbers than the paths from real case scenarios. Furthermore, detecting the attack between two instances of Amazon AWS is easy, because there is not a lot path diversity as we can see from the normal path between N. Virginia and N. California.

3) False positives: There is a false positive when a scheme claims to have detected an attack that did not exist. We evaluated the false positives for each individual mechanism of our system during a run of 1h15m. The false positives were calculated by executing each detection mechanism with scenarios without running the attack (i.e., without hijacking). By capturing the amount of alerts given by a mechanism we get the false positive rate $\#alerts/\#samples$, where $\#alerts$ is the number of alerts and $\#samples$ the number of samples taken. We tested for three different scenarios and each scenario contains a source and a destination. For the first scenario, we chose a node from POR as the source and a node from IRE as the destination; in the second, the source is a node from IRE and the destination is a node from GER; finally for the last scenario the source is a node from VA and the destination is a node from CA.

For Path and Hop the number of false positives observed was 0, because there would have to be legitimate route changes to cause them, but these are not frequent and none was observed. For Prop the number of false positives was also 0, as the mechanism always searches for the maximum RTT stored to compute the maximum propagation delay ever observed. Unless a great anomaly in RTT is found, the mechanism will not raise an alarm. For Lat, we received a new sample from 30 to 30 seconds getting a total of 150 samples per scenario. The results are presented in Figure 8. The values for the thresholds were chosen with the objective to reveal variation in the false positive rate. As we can see in all sets of columns the false positive rate is bigger for softer thresholds. This makes sense since small thresholds mean that a small variance of RTT is considered an attack. For the soft label the value used was 1.2,

TABLE II
VALUES OF THRESHOLDS USED FOR EACH SCENARIO. S, D AND H ARE THE SOURCE, DESTINATION AND HIJACKER, RESPECTIVELY.

	S:POR | D:IRE | H:FRA			S:POR | D:IRE | H:POL			S:VA | D:CA | H:POL			S:IRE | D:GER | H:S.Korea		
Mechanisms	Soft	Medium	Hard	Soft	Medium	Hard	Soft	Medium	Hard	Soft	Medium	Hard
Lat	1.2	1.3	1.4	2.1	2.2	2.3	3.6	3.65	3.7	13	13.5	14
Hop	1.05	1.1	1.15	1.05	1.1	1.15	2.05	2.1	2.15	2.2	2.25	2.3
Path	0.9	0.8	0.7	0.9	0.8	0.7	0.9	0.8	0.7	0.9	0.8	0.7
Prop	3.6	4.1	4.6	7.4	7.9	8.4	12.5	13	13.5	70	75	80

TABLE III
NUMBERS OF THE ASES THAT PACKETS TRAVERSE

Hijacker	From - To					
	POR - IRE		IRE - GER		VA - CA	
	Normal	Hijacked	Normal	Hijacked	Normal	Hijacked
FRA	1930,21320,1213	1930,21320,2200,15557,1213	-	-	-	-
POL	1930,21320,1213	1930,21320,8501,8890,1213	-	-	16509	16509,2603,8501,8890,16509
S.Korea	-	-	1213,21320,680	1213,3356,2516, 16509,4766,174,680	-	-

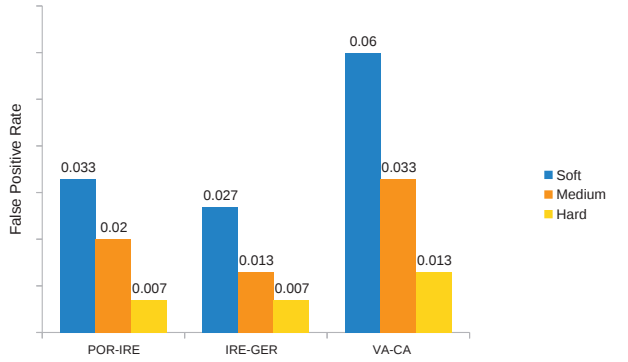


Fig. 8. False positive rate of RTT in different scenarios. The *Y* axis refers to the false positive rate and the *X* axis represents the different scenarios tested.

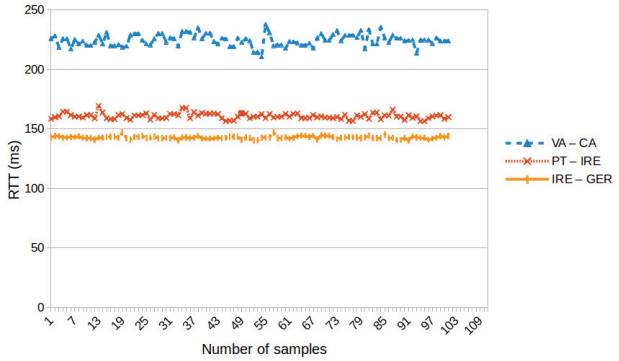


Fig. 9. RTT values in different scenarios. The *Y* axis refers to the RTT values in milliseconds and the *X* axis represents the number of samples.

for medium the value was 1.3 and for the hard label the value chosen was 1.4.

The results for DARSHANA were obtained with the soft thresholds for Lat. However, on the contrary of Lat, the false positive rate for DARSHANA was 0 in all scenarios, as the other mechanisms (Path, Hop, and Prop) filtered the false positives of Lat, leading to 0 false positives as obtained with each of the 3 individually.

C. Cost

DARSHANA keeps probing for RTT with period *k*. Unless anomalies in RTT are verified, leading the system to use techniques with bigger overhead, like Hop and Path. We evaluate the cost as how many times DARSHANA is forced to execute heavier techniques in normal conditions. The scenarios used were the same as in Section IV-B3. Setting a probing rate for RTT to 60 seconds, Figure 9 illustrates the values for round trip time in different scenarios.

During this period the mean deviations of the samples were low. The scenario with VA and CA, has the biggest mean deviation of approximately 5.02. This implies that the RTT usually remains constant, being difficult to observe anomalies and pass to heavier methods. Considering a value of T_{Lat} = 1.5, the total ping and traceroute messages for this period follow the following formulas, where $\#Msg\_Ping$ and $\#Msg\_Traceroute$ correspond to number of ping and traceroute messages, respectively $\#Msg\_Ping = T \times k$ and $\#Msg\_Traceroute = \#Msg\_Ping/n$.

Where T is the total time of the experiment, k is the ping period and n is the traceroute period. For this experiment $\#Msg\_Ping = 100 \times 1 = 100$ ping messages and $\#Msg\_Traceroute = 100/5 = 20$ traceroute messages. All of this demonstrates that even for low values of T_{Lat}, the total number of $\#Msg\_Ping$ and $\#Msg\_Traceroute$ end up only being dependent on k and n.

V. RELATED WORK

Many solutions have been proposed for the IP prefix hijacking problem. Some of them are crypto-based such as [3]–[6]. These solutions require deep changes in routers and network protocols. BGP routers need to sign and verify announcements which leads to a non negligible overhead.

Other solutions like [7]–[10] are more deployable because they do not require changes in routers, they only need access to public registries, like Route Views and European IP Networks (RIPE) to conduct passive monitoring and look out for

Multiple Origin Autonomous Systems (MOAS) [25]. An IP prefix should only be generated by a single AS, so this conflict may indicate a prefix hijacking. The problem associated to these solutions is that many times the public registries may be outdated and inaccurate, leading to an increase of the number of false positives.

Finally, there are solutions that rely only on the data plane like ours. They are not constrained by the availability of BGP information and are more accurate. [12] uses a set of monitors to detect prefix hijacking in real time. These vantage points monitor a prefix from topologically diverse areas. Each monitor keeps track of the hop count and the path to a target prefix and if past measurements disagree with new ones then a route hijacking is declared, the need for vantage points end up limiting the system. On the other hand, [11] detects IP prefix hijacking by observing unreachability events. It is owner-centric, in a point that the mechanism keeps sending probes to transit ASes. If enough ASes stop responding, the system declares the attack. If the attacker forwards the responses of the probes back to the sender, the attack is not detected.

In this paper we make use of the propagation delay as another criteria to detect route hijacking. This delay has been used in [26], which proposed a system that presents undeniable proof about traffic traversing a certain forbidden zone defined by the sender. To know if a certain relay node is not in the forbidden region, the minimum possible RTT from the source to any node in the forbidden zone was calculated, with the propagation delay. In case the RTT from the source to the relay node is less than the RTT calculated earlier, then the relay node is not in the forbidden region.

The design of the lightweight and end-host-based probing techniques was inspired by Hubble [27], where low overhead probing techniques are used first and heavier, but more reliable techniques, are only used when there is such a need.

VI. Conclusion

This paper presented DARSHANA, a route hijacking detection system. By only applying active probing methods, we ensure accuracy and deployability. Different techniques turn the system redundant enough to not be avoided by attackers. The design of the detection system minimizes the overhead, by using techniques with low overhead more often. Techniques with greater reliability and overhead are only executed when necessary. Our system is the first to use the propagation delay in this context, providing one more metric for the purpose of detection. We evaluated the system with small scale and real scenarios.

Acknowledgements This work was supported by the European Commission through project H2020-653884 (SafeCloud) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

[1] S. H. Y. Rekhter, T. Li, "A border gateway protocol 4 (RFC 4271)," January 2006.

[2] D. Goodin, "Hacking Team orchestrated brazen BGP hack to hijack IPs it did not own," 2015. [Online]. Available: http://arstechnica.com/security/2015/07/hacking-team-orchestrated-brazen-bgp-hack-to-hijack-ips-it-didnt-own/

[3] M. Lepinski, "BGPSEC protocol specification, draft-ietf-sidr-bgpsec-protocol-17," 2016.

[4] S.Kent, C.Lynn, and K.Seo, "Secure border gateway protocol (S-BGP)," *IEEE JSAC Special Issue on Network Security*, 2000.

[5] W.Aiello, J.Ioannidis, and P.McDaniel, "Origin authentication in inter-domain routing," *Proceedings of ACM Conference on Computer and Communications Security*, 2003.

[6] K.Butler, P.McDaniel, and W.Aiello, "Optimizing BGP security by exploiting path stability," *Proceedings ACM Conference on Computer and Communications Security*, 2006.

[7] M. Lad, D. Massey, D. Pei, and Y. Wu, "PHAS: A prefix hijack alert system," *Proceedings of the Usenix Security Conference*, pp. 153–166, 2006.

[8] J. Karlin, S. Forrest, and J. Rexford, "Pretty good BGP: improving BGP by cautiously adopting routes," *Proceedings of the 2006 IEEE International Conference on Network Protocols*, pp. 290–299, 2006.

[9] X. Hu, Mao, and Z. Morley, "Accurate Real-time Identification of IP Prefix Hijacking," *IEEE Symposium on Security and Privacy*, no. 2, pp. 3–17, 2007.

[10] X. Shi, Y. Xiang, Z. Wang, X. Yin, and J. Wu, "Detecting prefix hijackings in the Internet with ARGUS," *Proceedings of the 2012 ACM Internet Measurement Conference*, 2012.

[11] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "iSPY: detecting IP prefix hijacking on my own," *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, 2008.

[12] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A light-weight distributed scheme for detecting IP prefix hijacks in real-time," *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 277–288, 2007.

[13] K. Butler, T. Farley, P. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions," *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, 2010.

[14] M. Brown. (2008) Pakistan hijacks youtube. [Online]. Available: http://research.dyn.com/2008/02/pakistan-hijacks-youtube-1/

[15] H. Ballani, P. Francis, and X. Zhang, "A study of prefix hijacking and interception in the Internet," *ACM SIGCOMM Computer Communication*, 2007.

[16] "The handbook for radio communications, 89th edition," 2012.

[17] M. Luckie, Y. Hyun, and B. Huffaker, "Traceroute probe method and forward IP path inference," *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement conference*, p. 311, 2008.

[18] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with Paris traceroute," *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, pp. 153–158, 2006.

[19] Team cymru. [Online]. Available: http://www.team-cymru.org/IP-ASN-mapping.html

[20] Maxmind. [Online]. Available: http://dev.maxmind.com/

[21] Planetlab : global research networks that supports the development of new network services. [Online]. Available: https://www.planet-lab.eu/

[22] Amazon web services. [Online]. Available: https://aws.amazon.com/

[23] J. Cowie. (2013) The new threat: Targeted Internet traffic misdirection. [Online]. Available: http://research.dyn.com/2013/11/mitm-internet-hijacking/

[24] ——. (2010) China 18-minute mystery. [Online]. Available: http://research.dyn.com/2010/11/chinas-18-minute-mystery/

[25] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "An analysis of BGP multiple origin AS (MOAS) conflicts," *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement Workshop*, pp. 31–35, 2001.

[26] D. Levin, Y. Lee, L. Valenta, Z. Li, V. Lai, C. Lumezanu, N. Spring, and B. Bhattacharjee, "Alibi routing," *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pp. 611–624, 2015.

[27] J. John, E. Katz-Bessett, H. Madhyastha, A. Krishnamurthy, D.Wetherall, and T. Anderson, "Studying blackholes in the Internet with hubble," *Proceedings of NSDI*, 2008.

MACHETE: Multi-path Communication for Security

Diogo Raposo, Miguel L. Pardal, Luís Rodrigues, Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

***Abstract*—Communication through the Internet raises privacy and confidentiality concerns. Protocols such as HTTPS may be used to protect the communication, but occasionally vulnerabilities that may allow snooping on packet content are discovered. To address this issue, we present MACHETE, an application-layer multi-path communication mechanism that provides additional confidentiality by splitting data streams in different physical paths. MACHETE has to handle two challenges: sending packets over different paths when Internet's routing imposes a single path between pairs of network interfaces; splitting streams of data sent over TCP connections. MACHETE is the first to exploit MultiPath TCP (MPTCP) for security purposes. It leverages overlay networks and multihoming to handle the first challenge and MPTCP to handle the second. MACHETE establishes an overlay network and scatters the data over the available paths, thus reducing the effectiveness of snooping attacks. Mechanisms are provided to select paths based on path diversity.**

***Index Terms*—Multi-path Routing, Communication Confidentiality, Eavesdropping, Communication Privacy, MultiPath TCP**

I. Introduction

Sending information over the Internet has the disadvantage of making it vulnerable to eavesdropping by unauthorized third parties. This problem is especially important for organizations that handle critical data, such as governments, military, or healthcare. Communication protocols based on cryptographic mechanisms such as HTTPS and IPsec are the common solution to this problem. However, recent events show that it may be possible to break these protocols under certain conditions, and suggest that powerful adversaries may be able to do it if they access the encrypted data. For example, Adrian *et al.* presented a flaw in the Diffie-Hellman key exchange that allows downgrading the security of a TLS connection for a specified 512-bit group [1]. They claim that a nation-state may have the computational power to attack 1024-bit groups, which would allow decryption of many TLS channels over the Internet that implement this method.

We present MAChETe (*Multi-pAth Communication for sEcuriTy*), a means to mitigate the impact of such vulnerabilities. This system consists on using *MultiPath TCP* (MPTCP) [2]–[4] and *overlay networks* [5], [6] to split communication flows on different physical paths, possibly over a multihomed subnetwork [7], [8], as a defense-in-depth mechanism.

The rationale is that more effort is required to eavesdrop data split over several flows in comparison to a single flow. The problem addressed in this paper is, therefore, achieving additional communication *confidentiality* for critical data while still assuming confidence in the cryptography mechanisms.

978-1-5090-3216-7/16/$31.00

MACHETE has to handle two challenges. The first consists in sending packets over *different paths* when Internet's routing imposes a single path between a pair of source and destination network addresses. Overlay routing enables doing *application-layer routing*, allowing packets to deviate from the routing imposed at network level, by the Internet's routers and routing protocols. Overlay networks, in combination with multihoming, are used to create path diversity, allowing flows to be split over physically disjoint paths. Using a topology-aware decision algorithm, several *overlay nodes* are chosen, according to their location. Each node will create a single-hop overlay path to the destination, generating an overlay network.

The second challenge is to split the stream of data sent over a TCP connection. MPTCP is a recent extension of the TCP protocol that has the ability to distribute and send data among the different network interfaces of a device, e.g., the IEEE 802.3 ("wired", "Ethernet") and the 802.11 ("wireless", "WiFi") interfaces of a personal computer. However, MPTCP neither ensures the use of different physical paths, nor their diversity, as it was created mostly with performance in mind. The paths used by a MPTCP connection are imposed by the network interfaces of the source and destination hosts.

The combination of MPTCP with application-layer routing is itself a third challenge. Our objective is that MACHETE works at the *application layer*, without modifications to lower layers, but it has to route packets sent at transport layer under the control of MPTCP. MPTCP is a transport-layer protocol, so applications provide it source and destination IP addresses and ports. However, the overlay nodes have their own IP addresses and ports, unrelated to the previous ones. Therefore MACHETE has to play with the destination IP addresses and ports for communication to be possible.

The paper has three main contributions: (1) MACHETE is a system that improves communication confidentiality by splitting TCP data streams over diverse physical paths leveraging MPTCP, overlay networks, and multihoming; (2) It is the first work that leverages MPTCP for security and the first to combine MPTCP with application-layer routing and overlay networks; (3) Provides an experimental evaluation of MACHETE over the Internet in a wide-area deployment.

II. Background and Related Work

This section covers background and related work on the mechanisms used in MACHETE: MultiPath TCP, overlay

routing, and multihoming.

A. MultiPath TCP

MultiPath TCP (MPTCP) is an extension of the TCP protocol that enables endpoints to use several IP addresses and interfaces simultaneously when communicating [2], [3]. The protocol discovers which interfaces are available to use, establishes a connection, and splits the traffic among them. It presents the same programming interface as TCP, however the data is spread across several flows. The option field in the regular TCP protocol is filled with MPTCP data structures in order to inform the other end-point about the capability of implementing this protocol and to add flows to the communication. MPTCP has two important components on its configuration: path manager and packet scheduler.

The *path manager* is the module that handles how the flows are created in an MPTCP connection. The implementation of the protocol in Linux currently provides four schemes [4]: `default` does not create new flows, but accepts incoming; `fullmesh` creates a full-mesh of flows with all available interfaces/addresses in the device; `ndiffports` takes only one pair of IP addresses and modifies the source port to create the number of flows set by the user; `binder` uses the loose source routing algorithm of the Binder system [9].

The *scheduler* handles the distribution of the TCP packets (segments) over the flows, in close collaboration with TCP's congestion control mechanism [10]. MPTCP does not use a single congestion window as TCP, but one per flow. Similarly to TCP, the congestion control mechanism manages the size of each congestion window based on the round-trip time (RTT) of the flow and other factors (timeouts, reception of acknowledgments). The implementation of MPTCP for Linux by default fills the flow congestion window before starting to schedule packets on the next flow. Although in terms of performance it is important to take advantage of the throughput of the channels, in terms of splitting data for confidentiality it may be a disadvantage. In a communication composed by two flows where one has twice the throughput of the other, that flow will tend to send twice the amount of data of the other, which leads to a higher amount of data is susceptible to being spied upon. Linux's MPTCP implementation provides three scheduling modes: `default`, the one we just presented and the one with best performance; only uses another flow if the window of the flow in use does not allow sending data that is pending; starts sending using the flows with lower RTT; `fast round-robin` which uses sequentially all flows but fills the congestion window of a flow before starting with the next; `strict round-robin`, does real round-robin by sending the same amount of data through all the flows in sequence; waits for a flow to have free space to send a packet before scheduling the next one.

MPTCP is very similar to TCP in terms of security. Specifically, the RFC says that "The basic security goal of Multipath TCP (...) can be stated as: provide a solution that is no worse than standard TCP" [2]. There are a few works concerned with the security of MPTCP [11], [12].

B. Path Diversity

Path diversity can be achieved in multiple ways in a Multipath Communication. *Overlay Routing* and *Multihoming* are among some of the options.

Overlay routing allows the creation of a virtual network (an overlay network) on top of an already existing network infrastructure, like the Internet, without modifying it. The nodes of the network are hosts, i.e., machines that implement the network stack up to the OSI application layer. An overlay link may connect two nodes either directly or indirectly, through other nodes. These nodes route (forward) the packets at the application layer to the next or final node of the link. This adds a level of indirection in relation to the underlying OSI network layer topology. At the network layer the packets travel through the routes imposed by the Internet routing protocols, namely the Border Gateway Protocol v4 (BGPv4) [13]. At application layer the traffic may be deviated from these routes by sending it through overlay nodes in other locations. The original motivation for overlay routing is resilience [5], [6]. In case a network layer route is congested or faulty, routing done at the application layer may allow passing the traffic through other routes.

Another method of achieving path diversity is to use *multihoming*. This approach consists simply in having a customer network linked to two or more ISPs. Resilience and performance are the main advantages of this approach [14]. Different providers offer different performance levels to different parts of the network, so choosing the "right" provider will result in a performance increase [7], [8]. Akella *et al.* [8] evaluate the use of multihoming, using Hand-shake Round Trip Time (HRTT) as a measurement unit for data centers and enterprises. These studies [7], [8] conclude that simply by using two providers the performance is increased by at least 25% and that the improvements are very small beyond four providers.

III. MACHETE

MACHETE is an application-layer mechanism for improving communication confidentiality by splitting packets in different paths. It uses MPTCP over an overlay network to create multi-path communication. By setting up a network composed by several nodes it is possible to implement an overlay network, which consists of several links between the source and destination of a communication. MPTCP will use these overlay links to split the data to be transferred. Path diversity is sought by exploiting diversity between Autonomous Systems (ASs). MACHETE uses single-hop overlay routing as there seems to be no gain in using more hops, both in terms of performance and diversity [15].

The architecture of MACHETE is represented in Figure 1 and has three main components: Multi-path devices which communicate using MACHETE, that also can play the role of server (wait for connections) or client (start connections), similarly to TCP; Overlay nodes which are the nodes of the overlay network that forward messages on behalf of multi-path devices and create alternative communication paths; Multi-

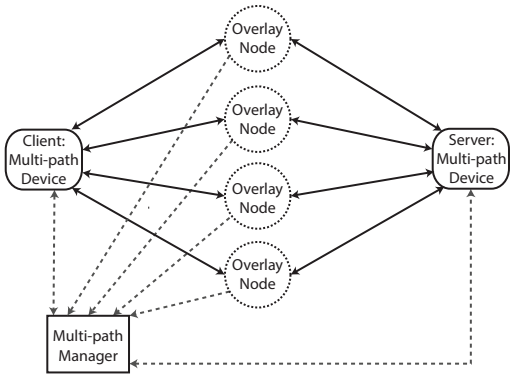


Fig. 1. The architecture of MACHETE. The solid lines represent data communication flows and the dashed lines control communication (e.g., node registration in the overlay network).

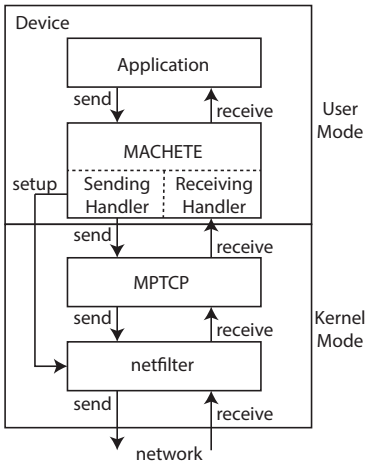


Fig. 2. MACHETE multi-path device architecture. MACHETE proper is a user level process running in a computer, which connects the application to the communication stack in the kernel (MPTCP) and netfilter (using `iptables`). MACHETE only establishes new rules when it creates a new connection (essentially an MPTCP connection).

path manager which is the component in charge of keeping track of the nodes that compose the overlay network.

This section presents MACHETE abstractly but in some places delves into the details of its implementation in Linux.

A. Threat Model

MACHETE is concerned about attacks against the confidentiality of data exchanged, so it considers passive attackers that eavesdrop on communication at certain physical locations. We assume that the attackers can eavesdrop on all packets at those locations, so confidentiality has to be achieved by reducing the locations where all traffic passes.

We assume that communications between the manager and each other component of MACHETE use default secure channels with a configuration which maximizes security, e.g., TLS using a 2048 bit key with elliptic curve Diffie-Hellman key exchange. Therefore the attackers can not compromise the manager's communication integrity and confidentiality.

We assume that the devices and nodes of the system are trustworthy, i.e., that they follow the protocol. This assumption has to be assured using proper security mechanisms, such as hardening, sandboxing, and access control. MACHETE is, however, prepared to recover from node crashes.

The multi-path manager might be a single-point of failure of the architecture, so it is replicated. We assume that a subset of the replicas can be compromised by an attacker or crash and we use a specific scheme to make overall multi-path manager tolerate these issues.

It is also important to notice that MACHETE has the objective of dealing with the most resourceful adversaries such as a nation-states. An adversary this resourceful can acquire control over several ASs in his area. Therefore it is important that the nodes are placed in a wide geographic area. Beyond that, it can also be stated that the closer an attacker is to the source or destination of the communication, the easier it is to find a single point of interception[1]. This being, multihoming is a very important component of MACHETE's deployment as it is discussed further ahead in this document.

[1]In this case, a autonomous system routing many or all of the data streams in a multi-path communication is considered a single point of failure

B. Multi-path Manager

The multi-path manager is the component that contains information about every entity in the network. Its function is to register every node and device addresses and to provide that information to devices that aim to communicate.

The multi-path manager was not developed from scratch but instead is a *tuple space* that implements Linda's generative coordination model [16]. A tuple space is a repository of data items called *tuples* and provides mainly three operations: insert tuple (`out`), read tuple (`rd`), and remove tuple (`in`).

MACHETE uses a specific tuple space called DepSpace [17], [18]. DepSpace is replicated in order to tolerate faults in some of the replicas. Specifically, it continues to operate correctly despite the failure of up to f out of $3f + 1$ replicas (typically 1 out of 4). DepSpace is Byzantine fault-tolerant, so it provides its service correctly even if f replicas are compromised or fail arbitrarily. Whenever server multi-path devices and overlay nodes start to run, they register with the multi-path manager by inserting a tuple on the tuple space.

C. Multi-path Device

A multi-path device is designated as a computer that uses MACHETE to communicate. The architecture of such a device is represented in Figure 2. This component dynamically establishes paths and splits the packets among them.

After a device registers itself on the multi-path manager, the process of transferring a stream of data (e.g., sending a file) is composed of three steps: *path setup*, *data transfer* and *path tear down*. Figure 3 represents this process. Next we describe each of these steps, dividing the first in two substeps.

MPTCP requires devices to have several network addresses to create more than one flow. If the device has several physical

interfaces, possibly connected to more than one provider – *multihoming* –, each one has an IP address. If that is not the case or that number of addresses is not enough, more than one address can be assigned to each interface, e.g., using Linux's virtual network interfaces [19]. Having two addresses (in total) on each device is enough to establish a network composed of four paths, which in general is enough to achieve the objective.

1) Path setup – choosing overlay nodes: The process starts by querying the multi-path manager about the available overlay nodes. Although the manager replies with all nodes available in the network, the number of nodes to be used by a certain connection, N_n, is a configuration parameter.

The overlay nodes are chosen taking into consideration the path *diversity* they provide. If there are several paths with the same diversity, the path with best *performance* (e.g., lowest RTT) is chosen. In the current version of MACHETE, the metric of diversity among two paths used is the number common ASs on both paths (higher number means worse diversity). For a path, the ASs are obtained using layer-4 traceroute [20], which provides precisely the ASs of the nodes along a path. The metric of performance is the RTT, measured using the `tokyo-ping` tool, which avoids some anomalies in `ping` [21]. When available, multihoming tends to improve diversity as the first ASs along the path will already be different, whereas with single-homming the opposite is true.

In MACHETE the path manager is set to `fullmesh`, to allow defining the number of flows in a way that makes MPTCP use the number of overlay nodes defined (N_n). This manager will create a network mesh composed by all the available interfaces/addresses in both the source and destination.

To balance the data among all nodes and obtain the expected confidentiality, the best packet scheduler is `strict round-robin`. This scheduler is configured with the number of packets sent in each flow before passing the turn to the next flow. To reduce the information sent in each flow (thus in each path), this parameter is set to 1. The `fast round-robin` scheduler can also be used if the communication is encrypted and the amount of bytes sent is high enough to ensure that not all communication passes in the same node, as it is not possible to decrypt data if it is not complete. The amount of bytes sent being enough or not to make the communication pass in more than one node is analyzed in Section IV-C2.

2) Path setup – managing addresses and ports: As already pointed out, the combination of MPTCP with application-layer routing is challenging. MACHETE works at application layer but it has to route packets sent by, and under the control of, a lower layer protocol: MPTCP, at transport layer.

Similarly to what happens with TCP, in MPTCP all packets sent over a connection take two pairs of IP addresses and port numbers, one for the source device, another for the destination (the difference in relation to TCP is that source and destination may have more than one address/port pair). However, in MACHETE the destination address and port may have to be different: (1) if the packet is leaving the sender device, the destination address/port should be those of the overlay node for the packet's flow; (2) if the packet is leaving an overlay node, the destination address/port should be those of the destination device and the source address/port should be those of the overlay node; (3) if the packet is returning to the overlay node, the destination address/port should be those of the source device and the source address/port should be those of the overlay node.

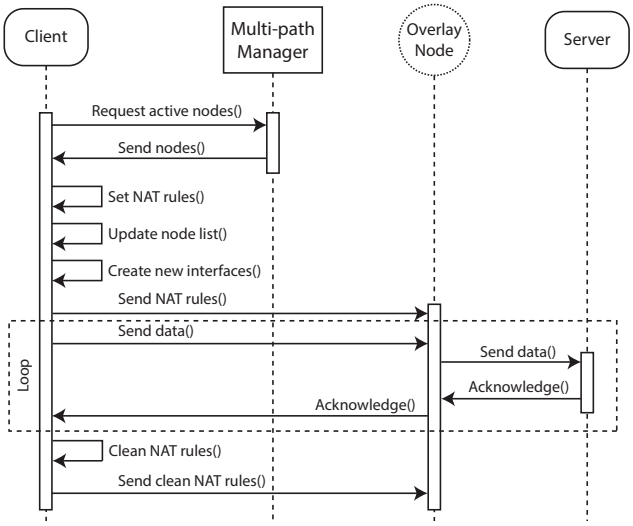


Fig. 3. MACHETE communication setup and data transfer example with a single overlay node.

The application requires MACHETE, thus also MPTCP, to send packets to the destination address and port. When a device does the setup of a path, it has to force these alternative addresses and ports to be used. To do it MACHETE leverages Linux's *netfilter* framework and the `iptables` command [22]. This framework allows doing network address translation (NAT), packet filtering, and other forms of packet handling. MACHETE uses it for network address translation.

When a path is setup, the `iptables` command is used to tell netfilter to change the destination IP address and port by those of an overlay node, depending on the flow (case (1) above; arrow *setup* in Figure 2). MPTCP inserts the destination IP address/port in the packets, but netfilter exchanges them before they are transmitted into the network. The `iptables` command inserts NAT rules for that purpose in the `output` chain, which is the set of rules applied to traffic being sent by a computer. For each link, a NAT rule is set[2].

Once this is done, the device informs each node about the rules they have to establish. In the overlay node it is necessary to route the traffic in both directions: when forwarding to the server (case (2) above) and when returning to the client (case (3) above). As soon as all nodes confirm that the rules are set, the data transfer may begin.

Figure 3 shows a time diagram that represents this process with a single overlay node.

[2]The format of the `iptables` rule is: `iptables -t nat -A -p tcp -s <source address> -d <destination address> -j DNAT --to-destination <new destination address>`

3) Data transfer: MACHETE uses MPTCP to establish a connection to the destination device. The client application will create a socket and provide it one of the server's address/port pairs; the MPTCP protocol will handle the passive creation of the flows. Despite the fact that netfilter modifies the destination addresses to deviate the connection's packets through the overlay nodes, the connection and each of its flows end up established similarly to what would normally happen with MPTCP.

This connection has two data streams, one in each direction, so that the client and server can send data to each other. This is represented in Figure 2 through the send and receive arrows. Notice again that the scheduler should be set to `strict round-robin`, otherwise MPTCP will fill each flow until its congestion window is full instead of sending packets using all flows, which is not desirable from the confidentiality point of view.

Each packet will suffer changes on its source and destination address twice: first in the source device, second in the overlay node. The same will happen to the acknowledgement packets.

4) Path tear down: To terminate a connection, the client device notifies the nodes that compose the overlay paths to remove the rules. The overlay device is listening on a specific port for receiving this indication, so that the packets destined to the node itself are never re-routed. Again, this device waits for all nodes to reply before removing its own rules. After all the steps are done the communication can be declared as finished. If the client fails to inform the nodes about the rules removal, the rules can stay established, since it is specific for a pair of source and destination addresses and, therefore, does not modify other connection's correct behavior or the possibility for the same source to create an identical connection.

D. Overlay Node

The overlay node is the component that plays the role of application-layer router, i.e., which forwards the packets received from the client device to the server device and vice-versa.

Overlay nodes receive from clients NAT rules and add/remove them from netfilter. These rules are set, again, with the `iptables` tool, this time using the `prerouting` and `postrouting` chains. The first chain leverages the changes on the traffic immediately after it was received by and interface and the second leverages the changes right before it leaves that interface. For each overlay network, four rules are established, two to change the source and destination when forwarding to the destination and two when forwarding to the source, as mentioned above in cases (2) and (3).

IV. Experimental Evaluation

This section presents the evaluation of MACHETE. We placed hosts in the Amazon AWS EC2 service [23] in nine different regions (Ireland, Frankfurt, North Virginia, California, Oregon, Tokyo, Seoul, Singapore and Sydney) and one in Portugal. We used up to 8 overlay nodes, one in each of the AWS regions, except for Ireland that contains the server.

TABLE I
Least diverse pair of paths in terms of number of common ASs with the single-home configuration. The paths are designated by the location of the overlay node.

Pair of paths	Common ASs	Common ASs (except first 7)
Singapore, Tokyo	13	6
Frankfurt, Seoul	12	5
Frankfurt, Tokyo	12	5
California, Seoul	12	5
California, Tokyo	12	5
Oregon, Tokyo	12	5
Seoul, Tokyo	12	5

TABLE II
Least diverse pair of paths in terms of number of common ASs with the dual-home configuration, in comparison to the single-home configuration. The paths are again designated by the location of the overlay node. In the dual-home configuration the left path uses the original connection and the right the 4G connection.

Pair of paths	Common ASs single-homed	Common ASs dual-homed
Oregon, Sydney	10	3
Oregon, Tokyo	12	3
Oregon, Seoul	11	2
Sydney, Tokyo	10	2
Frankfurt, California	9	1
Frankfurt, Oregon	10	1
Frankfurt, Seoul	12	1

Moreover we placed the client in the Portugal node. Therefore, between the client and server there are 8 single-hop overlay paths: one per overlay node.

Recall that the objective is to provide confidentiality by splitting communication over physically diverse paths with an acceptable performance. Therefore, the evaluation provides an assessment of the diversity in our scenario, presents a performance benchmark of the system, and analyses the confidentiality achieved.

A. Diversity

As stated before, confidentiality is only achieved if the paths are topologically disjoint, as attackers eavesdrop on traffic at certain locations (Section III-A). The approach used to verify the topology of the paths is to trace each route's chain of ASs from the source device to each node and from that same node to the destination. For that purpose we use layer-4 traceroute (i.e., the `lft` tool).

Table I shows the number of common ASs in the pairs of paths with highest value, between the 8 single-hop overlay paths, where each is designate by the location of the overlay node. There are at least 7 ASs in common in all paths leaving the client (Portugal). The reason for this lack of diversity is the fact that we did not use multihoming. Moreover, several ASs belong to Amazon, as also expected.

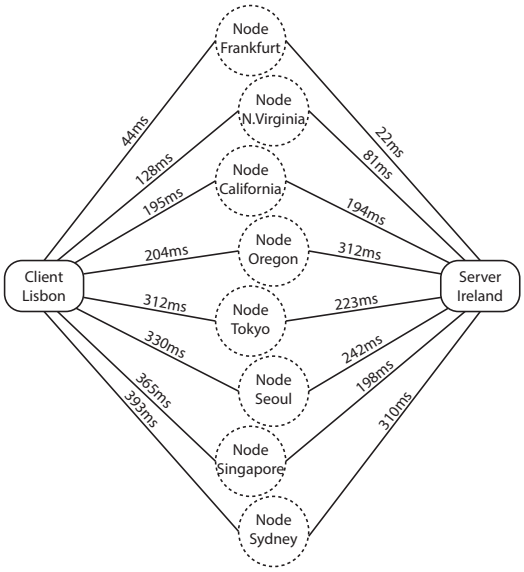


Fig. 4. Latencies between the Portugal host and the EC2 hosts used for the experimental evaluation.

We did an additional experiment to confirm that multihoming is beneficial in terms of diversity. We connected a second interface of the client device to a public provider of 4G service through a smartphone, then we used `lft` to obtain the ASs traversed by the paths. As shown in Table II, using multihoming provides an evident diversity, where the common nodes are again part of AWS's network. Notice that we used this multihoming configuration only for this test; the single-home configuration was used in all the experiments presented in the following sections. Multihoming is revealed to be a key component for MACHETE to achieve path diversity.

B. Performance

The performance evaluation considers three different aspects: the impact of adding paths on the delay of transferring files, the performance with diverse paths when transferring files, and the performance of path set up and tear down.

Figure 4 provides some insight on the network by showing the latencies between the hosts in the different locations, obtained with the `tokyo-ping` tool.

MACHETE forced MPTCP to use all the paths defined for every experiment by changing the number of IP addresses at the client (i.e., Portugal): 2 addresses for 2 paths, 3 addresses for 3 paths, etc. The client had a single network interface; the server had a single interface and a single IP address. All measurements were repeated 30 times.

1) Impact of adding paths: The evaluation consisted in observing the performance when paths (equivalently, nodes) were added one by one based on latency to the cluster: first in Frankfurt, next in N. Virginia, California, Oregon, Tokyo, Seoul, Singapore and finally Sydney (that has a the highest latency, as observed in Figure 4). The size of the files varied from 1 Byte to 1 GByte. In this experiment we used the `fast round-robin` scheduler to improve performance (and the `fullmesh` path manager which is fixed for MACHETE).

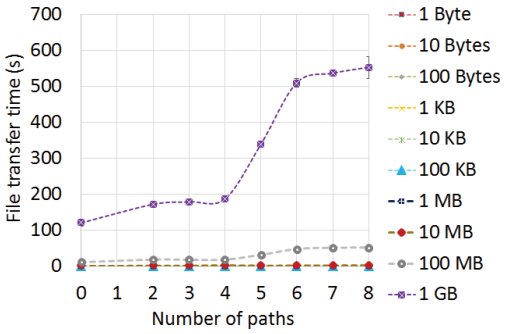


Fig. 5. Time to transfer a file versus number of paths. 0 paths means a normal TCP connection.

TABLE III
AVERAGE TIME OF SENDING FILES USING TWO NODES WITH TWO TYPES OF ROUND-ROBIN SCHEDULERS, COMPARED TO A NORMAL TCP CONNECTION. ALL VALUES ARE PRESENTED IN MILLISECONDS. EACH EVALUATION WAS REPEATED 30 TIMES.

File size	TCP	Fast r.-r.	Strict r.-r.
1 B	49	81	97
10 B	49	95	94
100 B	49	87	98
1 KB	49	94	90
10 KB	49	181	122
100 KB	49	265	201
1 MB	304	382	409
10 MB	1644	2295	3106
100 MB	11556	18836	23452
1 GB	121069	172215	218332

Figure 5 shows the values obtained for the time to transfer files of all sizes and from 2 to 8 paths, plus using a standard TCP connection (with no overlay nodes), and includes 95% confidence intervals, although most are too small to be visible.

The figure shows that splitting the packets in up to four different paths does not generate considerable overhead on the communication. From the fifth the duration increases due to the overlay nodes that compose the network at that point being farther away from both the source and destination. It is important to note that to achieve a physically disjoint network, using more nodes in the overlay network will result in selecting these further away from the source and destination.

2) Performance with diverse paths: Considering the diversity achieved in each of the eight regions used on the previous tests, this evaluation considers the two paths with highest diversity, i.e., those with overlay nodes at Frankfurt and California.

Table III, shows the overhead of using these two nodes, in comparison to a normal TCP stream. As shown, there is an overhead of 42% when using the `fast round-robin` scheduler and of 80% when using the `strict round-robin` scheduler. This overhead is the result of sending traffic through a node that is geographi-

cally distant from the source and the destination, California. When using a round-robin packet scheduler the whole multi-path connection is conditioned to each path's throughput. In fact, for the `strict round-robin` scheduler, the whole throughput is highly dependent of the path with the smallest bandwidth or highest congestion, since it waits for this channel to have free window space before sending to the next one.

3) Path set up and tear down: Figure 6 shows the time for setting up and tearing down the overlay paths. The current MACHETE implementation is suboptimal in the sense that both the setup and tear down phases are executed sequentially. According to the location of the node, this time will vary, however, as it can be seen, it always takes longer than one second, but never more than two in our scenario.

C. Confidentiality

The usual way of considering confidentiality in the security and cryptography literature is by relying on cryptographic protocols. For example, in protocols like IPsec AH/ESP or TLS, confidentiality is guaranteed as far as no vulnerabilities exist in the protocol design, implementation, and configuration. In this work we do not aim to provide such guarantees but to improve confidentiality in case communication is eavesdropped, (1) either it is not encrypted or (2) if it is encrypted but there is a vulnerability. This means that confidentiality was not studied in an absolute perspective, even though it is possible that in the second scenario, it might mitigate cryptography vulnerabilities to provide full confidentiality.

This different way of considering confidentiality led us to transmit a visual intuition of the MACHETE approach: an image is transmitted over MACHETE where an eavesdropper has access to one of the flows and reconstructs the image with that data. For the figures we did the reconstruction assuming the adversary managed to guess the metadata (figure size, color depth, etc.) even if the captured flow did not contain it.

When evaluating the confidentiality that MACHETE offers it is necessary to remember the operation of MPTCP. The most important factor is the scheduling that is used. MPTCP implements different types of scheduling, however, splitting data in packets in a round-robin fashion is the best approach to achieve confidentiality. The multi-path protocol implements two types of round-robin: `fast round-robin`, which takes advantage of the whole throughput of that channel, and `strict round-robin`, that waits for the next channel to have window space before sending the packet. The former is expected to perform faster, but the second to provide access to less data to an eavesdropper.

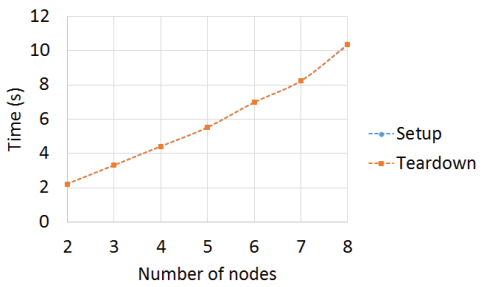


Fig. 6. Time for setting up and tearing down the overlay paths versus number of overlay nodes used.

We evaluate two aspects of confidentiality: the effect of the scheduling algorithm, and the effect of the file size.

1) Effect of the scheduling algorithm: Figures 7b and 7c show the different amount of data captured by two of four channels when sending the bitmap picture (the Linux penguin) shown in Figure 7a, with both types of round-robin scheduling. The channels shown in each figure are the ones that receive the most distinct amount of data, i.e., the one that receives the most (on the left) and the one that receives the least (on the right).

As shown on Figure 7b, using `strict round-robin` it is possible to notice that both channels receive approximately the same amount of data, resulting of the even distribution of packets. However, a pattern can also be noticed on its reconstruction.

Figure 7c shows the results of capturing the data when the `fast round-robin` scheduler is used. As expected, the flow where less data was transferred was the one passing through Sydney's node, the one with lowest throughput. As mentioned before, this scheduler depends on the throughput of each channel when distributing data, since a channel with better throughput has a larger congestion window to be filled.

In the standard MACHETE configuration, the result is the balance between flows observed in Figure 7b. The performance in the two cases was different, though. By filling the congestion windows with `fast round-robin` scheduling, the file that had 17MB was sent in 4 seconds. By using the strict round-robin the file took 88 seconds to be transferred, which is much slower. The first mode achieves a throughput of 34 Mb/s, whereas the second a mere 1.9 Mb/s.

In short both approaches have their advantages and disadvantages: the first one takes longer and might be susceptible to easier data reconstruction, but provides a good control on how the packets are distributed; the second has its packet distribution dependent of each flows' throughput, but transfers the files faster.

2) Effect of the file size: Another factor to take into account is the size of the files sent. At this point it is important to remember MPTCP's behavior when creating new flows. The first flow does not wait for the creation of new flows to start transferring data. This means that for very small files ($<$10KB) MPTCP does not split the packets through any new flows, since this data is sent before any new flow can be established for the stream. Regarding larger files, it is only necessary to experiment with the fast round-robin mode, since the strict round-robin is not influenced by congestion window sizes and, therefore, the sizes are not a factor to take into account.

Figure 8 shows the results of capturing the data transferred in the flow with highest throughput (which is the same as mentioned above of 34 Mb/s), when sending the same image with different sizes: from 1 MB to 17 MB. As it can be

(a) The original image sent.

(b) The data sent over the flows is balanced (strict round robin).

(c) The data sent over the flows is unbalanced (fast round robin).

Fig. 7. Original image and two reconstructions considering the eavesdropper has access to 2 of the 4 flows in each case.

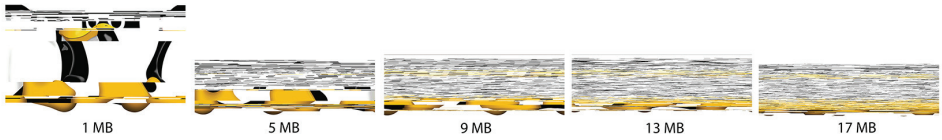


Fig. 8. Results of capturing the data transferred by the flow with most throughput. The same image was sent with different sizes.

observed, larger files have stronger resistance to eavesdropping as data is better split among the paths.

V. Conclusions

MACHETE is a first effort on providing confidentiality to communications by splitting the packet flows among different physical paths. By establishing dynamic overlay networks, composed by several paths with a single overlay node it was possible to provide physical path diversity. Using MPTCP it was possible to develop a system that transfer data streams (instead of isolated packets) without compromising performance. We evaluated the performance and confidentiality achieved by our implementation, showing that, not only it prevents the attacker from accessing considerable amounts of data, in the case it is trying to spy on the communication, but also provides different tradeoffs between confidentiality and performance. We believe, that efficiently splitting the communication over physically disjoint channels is the key to maintain confidentiality.

Acknowledgements

This work was supported by the European Commission through project H2020-653884 (SafeCloud) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. Vandersloot, E. Wustrow, and S. Z. Paul, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015.

[2] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath TCP development, IETF RFC 6182," 2011.

[3] S. Barré, C. Paasch, and O. Bonaventure, "Multipath TCP: from theory to practice," in *Networking 2011*. Springer, 2011, pp. 444–457.

[4] C. Paasch, S. Barre *et al.*, "Multipath TCP in the Linux kernel, available from http://www.multipath-tcp.org."

[5] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001, pp. 131–145.

[6] Y. Amir and C. Danilov, "Reliable communication in overlay networks," *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 511–520, 2003.

[7] A. Akella, B. Maggs, S. Seshan, and A. Shaikh, "On the Performance Benefits of Multihoming Route Control," *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 91–104, 2008.

[8] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, "A measurement-based analysis of multihoming," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, pp. 353–364.

[9] L. Boccassi, M. M. Fayed, and M. K. Marina, "Binder: a system to aggregate multiple Internet gateways in community networks," in *Proceedings of the 2013 ACM MobiCom Workshop on Lowest Cost Denominator Networking for Universal Access*, 2013, pp. 3–8.

[10] D. Wischik and C. Raiciu, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, 2011, pp. 99–112.

[11] J. Díez, M. Bagnulo, F. Valera, and I. Vidal, "Security for multipath TCP: a constructive approach," *International Journal of Internet Protocol Technology*, vol. 6, no. 3, pp. 146–155, 2011.

[12] O. Bonaventure, "MPTLS: Making TLS and multipath TCP stronger together," *IETF, Individual Submission, Internet Draft draft-bonaventure-mptcp-tls-00*, 2014.

[13] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4), RFC 1771," 1995.

[14] J. He and J. Rexford, "Toward Internet-wide multipath routing," *IEEE Network*, vol. 22, no. 2, pp. 16–21, 2008.

[15] J. Han, D. Watson, and F. Jahanian, "Topology aware overlay networks," *Proceedings of the IEEE INFOCOM*, vol. 4, pp. 2554–2565, 2005.

[16] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programing Languages and Systems*, vol. 7, no. 1, pp. 80–112, Jan. 1985.

[17] A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga, "DepSpace: a Byzantine fault-tolerant coordination service," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Apr. 2008, pp. 163–176.

[18] "DepSpace." [Online]. Available: https://github.com/bft-smart/depspace

[19] J.-S. Kim, K. Kim, and S.-I. Jung, "Building a high-performance communication layer over virtual interface architecture on Linux clusters," in *Proceedings of the 15th ACM International Conference on Supercomputing*, 2001, pp. 335–347.

[20] "Layer four traceroute (lft) project." [Online]. Available: http://pwhois.org/lft/index.who

[21] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush, "From Paris to Tokyo: On the suitability of ping to measure latency," in *Proceedings of the 2013 ACM Internet Measurement Conference*, 2013, pp. 427–432.

[22] R. Russell, "Linux 2.4 packet filtering howto, revision 1.26," Jan. 2002.

[23] "Amazon Web Services." [Online]. Available: https://aws.amazon.com/

Feature Set Tuning in Statistical Learning Network Intrusion Detection

Arnaldo Gouveia[1,2] Miguel Correia[2]
[1]Portugal Telecom [2]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Lisboa, Portugal

***Abstract*—The detection of security-related events using machine learning approaches has been extensively investigated. In particular, machine learning applied to network intrusion detection systems (NIDS) has attracted a lot of attention due to its good generalization and unknown attack detection capabilities. A number of classification techniques have been used for this purpose, revealing good generalization properties. In this paper we go one step further by evaluating the performance of NIDSs when feature set tuning and reduction are realized. We evaluate a number of state of the art learning algorithms that are raising much interest but have not been used for intrusion detection yet. We compare a representative set of algorithms: Ada, ROC-based learners, two types of Classification Trees, Boosted Logistic Regression, Generalized Linear Models, Gradient Boosting Machines, and Neural Networks. The main objective is to reduce the number of features used – thus also the size of the data processed – to improve speed while maintaining adequate accuracy.**

I. Introduction

With the continuous growth in number and impact of network attacks, *network intrusion detection systems* (NIDS) are increasingly becoming critical security elements. From a research standpoint, although investigated for many years [16], NIDSs continue to get a lot of attention due to their practical interest regarding effective detection of malicious attacks, while processing large data volumes with machine learning algorithms [17], [30].

Anomaly-based network intrusion detection is a particularly promising approach as it allows detecting previously unknown attacks. This approach resorts to machine learning to create models of normal behavior, then detecting deviations. In such IDSs, attacks are detected when anomalies are identified. The generalization machine learning algorithms achieve in the learning phase allows the identification of anomalies even if the normal traffic observed in runtime is not identical to the traffic of the learning phase. False positives and false negatives are the cost of this generalization capability.

Deep learning is currently a hot topic in machine learning [1], [2]. Deep learning techniques have been achieving excellent results in recognition of speech, faces, and images in general, to name just a few examples [1], [22]. Although initially the term deep learning referred to neural networks with many layers ("deep"), today other algorithms apparently very different are known to have similar behavior and also put under the term. This paper also explores the use of other algorithms seldom employed in NIDSs: Generalized Linear Models (GLM) and Gradient Boosting Machines (GBM). Other more commonly referenced algorithms are also compared, namely Classification Trees and Neural Networks.

978-1-5090-3216-7/16/$31.00

The goal of this comparison is evaluating the potential of *feature set tuning* to improve detection speed while maintaining accuracy at acceptable levels. This is especially important in times in which NIDSs have to process the high traffic loads that travel our networks. Feature selection is an important part of the process of dataset tuning. Therefore it is important for machine learning-based NIDS development in order to save time while providing an insight into the underlying feature details relevant to the classification process. It is a well-known fact that not all traffic features (or attributes) are equally useful to detect attacks/intrusions [14], [27]. Therefore feature set tuning allows reducing the feature set, improving the training dataset with the goal of obtaining speed gains while maintaining an acceptable accuracy [27].

Our results show that after tuning all algorithms present similar metrics, although the number of features reduced depends on the algorithm. GBM has achieved the highest feature reduction, closely followed by RPART.

The main contributions of this paper are: (1) a comparison of intrusion detection performance with a set of relevant machine learning algorithms; (2) a study showing how feature set tuning allows reducing datasets and maintaining accuracy at similar acceptable levels. In this paper, we test tune the UNB ISCX Intrusion Detection Evaluation Dataset while validating a number of representative machine learning algorithms from the literature.

II. The Dataset

The UNB ISCX Intrusion Detection Evaluation Dataset was developed in an attempt to provide a quality dataset for network intrusion detection research [23]. The approach for defining this dataset involved identifying features that would allow effective detection, while minimizing processing costs.

Cost-based models have often been used regarding feature definition in fraud based detection. Stolfo et al. [25] have shown that cost-based assertions developed for fraud detection can be generalized and applied to network intrusion detection as a criteria for feature finding. With this approach in mind, the authors defined a set of features intrinsically related to specific classes of traffic anomalies like Probing, Remote to Local, Denial of Service, and User to Remote attacks. By maximizing cost in specific cost models, a number of features have been identified by the authors and used in the UNB ISCX Intrusion Detection Evaluation Dataset. In this context the features chosen were the best candidates for maximizing the types of cost characteristic to intrusions: (1) damage cost: the amount of damage caused by an attack if intrusion detection is not attained. (2) challenge cost: the cost to act upon a potential

Class	Train dataset attacks	Test dataset only attacks
Probing	portsweep, ipsweep, satan, guesspasswd, spy, nmap	snmpguess, saint, mscan, xsnoop
DoS	back, smurf, neptune, land, pod, teardrop, buffer overflow, warezclient, warezmaster	apache2, worm, udpstorm, xterm
R2L	imap, phf, multihop	snmpget, httptunnel, xlock, sendmail, ps,
U2R	loadmodule, ftp write, rootkit	sqlattack , mailbomb, processtable, perl

TABLE I. ATTACKS IN THE UNB ISCX TRAIN / TEST DATASETS (ALL ATTACKS FROM THE FIRST EXIST ALSO IN THE SECOND)

intrusion when it is detected; and (3) operational cost: the resources needed identify the attacks.

The UNB ISCX dataset is composed of sequences of entries in the form of records labeled as either *normal* or *attack*. Each entry contains a set of characteristics of a *flow*, i.e., of a sequence of IP packets starting at a time instant and ending at another, between which data flows between two IP addresses using a transport-layer protocol (TCP, UDP) and an application-layer protocol (HTTP, SMTP, SSH, IMAP, POP3, or FTP). The dataset is fairly balanced with prior class probabilities of 0.465736 for the *normal* class and 0.534264 for the *anomaly* class.

The attacks represented in the UNB ISCX dataset fall into four classes:

Denial of Service Attacks (Dos). In this class of attacks the attacker renders computing or memory resources too busy or too full to handle legitimate requests or denies legitimate users access to a machine, e.g., land, syn flood, etc.

User to Root Attacks (U2R). This is a class of attacks in which the attacker starts out with access to a normal user account on the system and is able to exploit some vulnerability to gain unauthorized root access to the system. e.g., loadmodule or perl.

Remote to Login Attacks (R2L). This type of attack occurs when an external attacker exploits some vulnerability to gain local access as a user in that machine, e.g., ftp write, http tunnel, password guess, etc.

Probing Attacks. These are attempts to gather information about any systems for the purpose of circumventing its security controls, e.g., network scans, port sweep, nmap, satan, mscan, etc.

The UNB ISCX dataset is composed of two sub-datasets: a *train dataset*, used for training a NIDS, and a *test dataset*, used for testing. Both have the same structure and contain all four types of attacks. However, the test dataset has more attacks as shown in Table I, to allow evaluating the ability of algorithms to generalize. The train dataset has around 2.2 GB of data, whereas the test dataset has 0.8 GB.

Each record of the dataset is characterized by features that fall into three categories: basic, content, and traffic. These features are represented in Table II.

Feature	Detail
duration	length of the flow in seconds
protocol-type	type of the protocol, e.g., TCP, UDP, ICMP
service	network service, e.g., HTTP, telnet
src-bytes	num. of data bytes from source to destination
dst-bytes	num. of data bytes from destination to source
flag	status of the flow, normal or error
lang	1 if flow is for the same host/port; 0 otherwise
wrong-fragment	num. of erroneous fragments
urgent	num. of urgent packets
hot	num. of hot indicators
num-failed-logins	num. of failed login attempts
logged-in	1 if successfully logged in; 0 otherwise
num-compromised	num. of compromised conditions
root-shell	1 if root shell is obtained; 0 otherwise
su-attempted	1 if su root command attempted; 0 otherwise
num-root	num. of root accesses
num-file-creations	num. of file creation operations
num-shells	num. of shell prompt
num-access-files	num. of operations on access control files
num-outbound-cmds	num. of outbound commands in a ftp session
is-host-login	1 if the login belongs to the hot list; 0 otherwise
is-guest-login	1 if the login is a guest login; 0 otherwise
count	num. of connections to the same host as current
serror-rate	% of connections that have SYN errors
rerror-rate	% of connections that have REJ errors
same-srv-rate	% of connections to the same service
diff-srv-rate	% of connections to different services
srv-count	num. of connections to the same service as current
srv-serror-rate	% of connections that have SYN errors
srv-rerror-rate	% of connections that have REJ errors
srv-diff-host-rate	% of connections to different hosts
dst-host-count	num. of connections to the same destination host
dst-host-srv-count	num. of connections to the same service as current
dst-host-same-srv-rate	% of connections to the same service
dst-host-diff-srv-rate	% of connections to different services
dst-host-same-src-port-rate	% of connections from same source and port
dst-host-srv-diff-host-rate	% of connections to different services
dst-host-serror-rate	% of connections that have SYN errors
dst-host-srv-serror-rate	% of connections that have SYN errors per service
dst-host-rerror-rate	% of connections that have REJ errors
dst-host-srv-rerror-rate	% of connections that have REJ errors per service

TABLE II. FEATURES USED TO CHARACTERIZE EACH FLOW IN THE DATASET: BASIC (TOP), CONTENT (MIDDLE), TRAFFIC (BOTTOM).

	Actual Normal	Actual Anomaly
Predicted Normal	T_P	F_N
Predicted Anomaly	F_P	T_N

TABLE III. CONFUSION TABLE MODEL FOR METRICS

III. PERFORMANCE METRICS

We use a set of metrics to compare the algorithms and the effect of feature tuning. These metrics are mostly obtained from the confusion matrix (see Table III). A positive is the detection of a malicious flow, and a negative a non-detection. The detection of lack of detection can be right (true) or wrong (false).

Accuracy. Accuracy measures how well a classification test identifies an event class. The accuracy is the sum of true results (both true positives and true negatives) divided by the total number of observations (sum of all true positives, true negatives, false positives and false negatives): $Acc = (T_P + T_N)/(T_P + T_N + F_P + F_N)$. Accuracy ranges from 0 to 1, being the 1 the most favourable for a strictly balanced dataset.

No information rate. The no-information rate metric is the proportion of the most common class.

Kappa. The kappa statistic is a measure of agreement between

observations, where an observation is a validation round. We use this metric as defined by Cohen [6]: $\kappa = (P_o - P_e)(1 - P_e)$, where P_o is the observed accuracy and P_e the expected accuracy under random agreement [29]. The most favourable case for $kappa$ is perfect agreement, which would equate to a kappa value of 1.

Sensitivity (or True Positive Rate). Sensitivity is the probability that a test will indicate a positive condition among the set of positives: $T_{PR} = T_P/(T_P + F_N)$. Sensitivity ranges from 0 to 1 with 1 being the most favorable case for a strictly balanced dataset.

Specificity (or True Negative Rate). Specificity measures the proportion of negatives that are correctly identified as such: $T_{NR} = T_N/(T_N + F_P)$. Specificity ranges from 0 to 1, with 1 being the most favorable case for a strictly balanced dataset.

Positive Predicted Value. The Positive Predicted Value (PRV) is the proportion of true positive results referenced to the sum of true positives and false positives results: $P_{PV} = T_P/(T_P + F_P)$. PRV ranges from 0 to 1, with 1 being the most favorable case for a strictly balanced dataset.

Negative Predicted Value. The Negative Predicted Value (NPV) is the proportion of true negative results referenced to the sum of true negatives and false negative results: $N_{PV} = T_N/(T_N + F_N)$. NPV ranges from 0 to 1, with 1 being 1 the most favorable case for a strictly balanced dataset.

Positive Likelihood Ratio. In medicine, likelihood ratios are used for confirming a diagnostic test. They use the sensitivity and specificity of the test to determine whether a test result confirms the probability that a condition (such as a disease state) exists [28]. In intrusion detection this metric can be used as a confirmation that positive indicators are supported by malicious and intentional activity and it has been used in IDS related research [12]. It is given by $LR_+ = Sensitivity/(1 - Specificity)$. Values greater than 1 and greater confirm the existence of intentional malicious behaviour activity with ever increasing probability as LR_+ grows.

Balanced Accuracy. Balanced accuracy is given by $cT_P/(T_P + F_P) + (1 - c)T_N/(T_N + F_N)$, where c belongs to the interval $[0, 1]$ translating the imbalance (c and $1 - c$ are in practice the priors). If the classifier performs equally well on either class, this term reduces to the conventional accuracy (number of correct predictions divided by number of predictions): $T_P/(2(T_P + F_P)) + T_N/(2(T_N + F_N))$. Balanced Accuracy ranges from 0 to 1, with 1 being the most favorable case for a strictly balanced dataset.

IV. The Method

The core of the experiment is a comparison of a set of representative machine learning algorithms in the context of network intrusion detection. The steps followed were:

- Selection of a dataset of network traffic designed for network intrusion detection evaluation;
- Tune the training of the algorithms and extract the subset of the most relevant features using the R `caret` package `varImp()` function. When no tuning parameters are provided by `caret`, default values are used;
- Train and test with a feature reduced dataset.

We start with the first and leave the rest for Section V.

A. Feature Importance Determination

From an optimization perspective it is relevant to know what is the importance of the various features (or variables) used in the learners in terms of their contribution to the final performance result. A number of approaches have been used in the past for this purpose, namely Correlation-based Feature Selection, Information Gain, Gain Ratio, and the ROC based variable importance [3]. In this paper we investigate the use of the area under curve (AUC) of the receiver operating characteristic (ROC) curve as a performance measure for NIDS based on some machine learning learners. The variable importance list has been obtained with the `varImp()` function of the `caret` R package, which uses a ROC AUC maximization approach. Feature #15 of the original dataset was discarded because it was constant in both train and test datasets. The importance scale is presented in a percentual relative range for the 20 most relevant features in each algorithm.

B. Model Independent Feature Importance Selection

If there is no model-specific way to estimate feature importance (or the argument `useModel = FALSE` is used in `varImp()`), the importance of each feature is evaluated individually using a metric-based approach. This approach is model-independent. In face of this argument the `useModel = FALSE` option has been the choice for running the `varImp()` function in the models in which this option applies, namely Classification Trees and Random Forests.

ROC curve analysis is the approach used for reducing the feature set. For 2-class problems like intrusion detection (classes attack/no-attack), a series of cutoffs is applied to the algorithm data to predict the class. The sensitivity and specificity are computed for each cutoff and the ROC curve area is computed as the measure of variable importance. For a specific class, the maximum area under the curve across the relevant pair-wise AUCs is used as the variable importance metric. The model selected as the best is the candidate with the highest accuracy. If more than one tuning parameter is optimal then the function will try to choose the combination that corresponds to the least complex model [15].

C. Computational Environment

The data preparation phase, involving feature preprocessing, has been done using WEKA 3.6. WEKA stands for the Waikato Environment for Knowledge Analysis, which was developed at the University of Waikato of New Zealand. All the remaining experiments were performed with the R environment for statistical computing supported by the R Foundation for Statistical Computing [19]. R is widely used among statisticians and data miners for developing statistical software and data analysis. We used the following R packages: caret 6.0-68, rpart 4.1-10, neural net 1.32, C50 0.1.0-24, gbm 2.1.1, pROC 1.8 and rooc 1.2.

V. The Algorithms

The list of learners we considered is not exhaustive, but it contains a representative subset of good performance learners. This section presents the study itself. For each algorithm we explain briefly how it works and how it was tuned in order to obtain results as good as possible. The implementations of the algorithms used were those in R and its packages.

A. Ada (Boosted Classification Trees)

Ada is used in conjunction with other (weak) learning algorithms to improve their performance using boosting [10]. In this case we employ Ada as a booster of classification trees, used as weak learners.

In Boosted Classification Trees the training events that are misclassified (a signal event fell on a background leaf or vice-versa) have their weights increased (boosted), and a new tree is formed. This procedure is then repeated for the new tree. In this way many trees are built up. The score from the m_{th} individual tree T_m is taken as $+1$ if the event falls on a signal leaf and -1 if the event falls on a background leaf. The final score is taken as a weighted sum of the scores of the individual leaves [20].

Tuning. Cross-Validated (10 fold) re-sampling (also 10 fold) has been used. For classification using package `ada` the tuning parameters are: `Number of Trees` (iter), `Max Tree Depth` (maxdepth) and `nu` (learning rate). The Tuning parameter `nu` was held constant at a value of 0.1. Accuracy was used to select the optimal model using the largest value. The final values used for the model were `iter = 150, maxdepth = 3` and `nu = 0.1`.

B. Ranking Classifier

This is a classification method based on receiver operating characteristics (ROC). The method value rocc is chosen by `caret` from package `rocc` with the tuning parameter `xgenes`. Briefly speaking, features are selected according to their contribution to the ranked AUC value using the training set. The function performs classification by leave-one-out-cross-validation (LOOCV) using the ROC based classifier: features are combined to a group by the mean ROC value expression. Afterwards these samples are ranked according to their contribution to the AUC value. The feature group that yields optimal accuracy in the training samples is then used to classify new samples.

Tuning. The only tuning parameter available in `caret` is the number of variables retained in each LOOCV cycle: `xgenes`. Accuracy was used to select the optimal model using the largest value. The final value used for the model was `xgenes = 2`.

C. Classification Trees

Classification and regression trees were first described by by Brieman et al. [5]. Classification or Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. Decision trees create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The deeper the tree, the more complex the decision rules and the fitter the model [5]. We considered two algorithms: C5.0 and Rpart.

C5.0. This is a classification tree model that works by splitting the sample based on the field that provides the maximum information gain. The C5.0 model can split samples based on the largest information gain. The sample subset that is get from the former split is split after. The process will continue until the sample subset cannot be split. Finally a pruning process is executed examining the lowest level split upwards. Those sample subsets that do not have a significant contribution to the model information gain will be dropped.

Tuning. Cross-Validation (10 fold) has been used. The option for method definition in `caret` has been `method = C5.0Tree`. This method has no available tuning parameters in `caret`.

Rpart. Rpart can be generated through the `rpart` package. `Rpart` builds classification or regression models of a very general structure using a two stage procedure; the resulting models can be represented as binary trees. One of the decision trees implementation used was the one in R's `rpart` package.

Tuning. Cross-Validated (10 fold) re-sampling (3 fold) has been used. The option for `caret` has been `method = rpart2`. This method in `caret` has only available as tuning parameters `maxdepth` (maximum tree depth). Accuracy was used to select the optimal model using the largest value. The final value used for the model was `maxdepth = 3`.

D. Boosted Logistic Regression

Logistic regression was developed by Duncan and Walker [26] and Cox [7]. The Boosted Logistic regression learner has been described first hand by Friedman et al. [10]. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating the classes using a logistic function, which is the cumulative logistic distribution. Boosting sequentially applies a classification algorithm to weighted versions of the training data and then takes a weighted majority vote of the sequence of classifiers thus produced. In the context of `caret` package it can be used as a classifier.

Tuning. Cross-Validated (10 fold) re-sampling (also 10 fold) has been used. The method `LogitBoost` in `caret` admits `nIter` (number of boosting terations) as tuning parameter. The optimal number of iterations found for optimizing the accuracy figure was 31.

E. Generalized Linear Models

Generalized linear models (GLM) are a generalization of linear regression. Linear regression models the dependency of a response y on a vector of features $x(y \sim x^T\beta + \beta - 0)$. These models are built with the assumptions that y has a Gaussian distribution with a variance of σ^2 and the mean is a linear function of x with an offset of some constant $\beta - 0$, i.e., $y = \mathcal{N}(x^T\beta + \beta - 0; \sigma^2)$. These assumptions can be overly restrictive for real-world data that does not necessarily have a Gaussian distribution. GLM generalizes linear regression in the following way: it adds a non-linear link

function that transforms the expectation of response variable, so that $link(y) = x^T\beta + \beta - 0$ and allows variance to depend on the predicted value by specifying the conditional distribution of the response variable or the family argument.

Tuning. Cross-Validated (10 fold) re-sampling (also 10 fold) has been used. The GLM model, specified by "`method = glm`", offers no tuning parameters in `caret`.

F. Gradient Boosting Machines

Gradient Boosting Machines (GBM) is an algorithm designed to produce a model built by an ensemble of weak prediction models. It builds the model in a stage-wise fashion and is generalized by allowing an arbitrarily differentiable loss function. GBM fits consecutive trees as weak predictors where each solves for the net error of the prior. The idea of gradient boosting originated in Breiman's observation that boosting can be interpreted as an optimization algorithm using a suitable cost function [4].

Tuning. Cross-Validated (10 fold) re-sampling (also 10 fold) has been used. The tuning parameters offered by the `caret` package are: `n.trees` (number of boosting iterations), `interaction.depth` (maximum tree depth), `shrinkage` (shrinkage), `n.minobsinnode` (minimum terminal node size). The maximum depth of a tree is a tree specific parameter used to control over-fitting as higher depth will allow the model to learn algorithmic rules very specific to a particular sample. In the tuning phase a set of maximum depth of 2, 3 and 4 have been used.

A non-negative integer that defines the number of trees, (`trees`) has been tested with two values: 5 and 10. The learning rate, as a boosting parameter, has been set to `0.1`. This determines the impact of each tree on the final outcome. Lower values are generally preferred as they make the model robust to the specific characteristics of the tree allowing it to generalize well, although with a cost in terms of processing time. Several good and approximate models have been obtained. We presented the results for one with number of `trees=10, mean depth=4, min leaves=15` and `max leaves=16`.

G. Neural Networks

The variety of deep learning architectures is vast and includes examples such as Autoencoders, Multilayer Perceptron, Recurrent Neural Networks (RNNs), Restricted Boltzmann Machines (RBMs), Self Organizing Maps (SOMs) and Convolutional Neural Networks. In the scope of this category of classifiers a more shallow option of an one hidden layer neural net has been used, namely the `nnet` model of the `caret` package.

Tuning. Ten fold cross-validation was used. Tuning parameter `size` was tuned with values of 8 and 16. The value for `decay` was tuned with values of `1e-04` and `1e-03`. Accuracy values were used to select the optimal model using their largest value after 500 iterations. The final values chosen for the model were `size = 16` and `decay = 1e-04`.

VI. Result Analysis and Discussion

The results for all algorithms are presented in Table IV for the case of no feature selection, and Table V for the case of a reduced set of features. The values for accuracy, balanced accuracy, kappa and positive likelihood ratio are shown in Figures 1, 2, 3 and 4. As the dataset has class priors near 50%, the balanced accuracy exhibits the same behaviour as the accuracy. It is commonly accepted that a good model has to have low training error and low generalization error (or test error). However the need for generalization often leads to accuracy levels that are not near 1 for most algorithms, for a balanced dataset.

Overall speaking the accuracy and balanced accuracy exhibit reasonably good values, although in bands that potentially preclude the onus of overfitting. On the other hand, the values of kappa fall in ranges that are considered moderate (range 0.41 to 0.60) to substantial agreement (range 0.61 to 0.80) [29].

Domingos [8] states a number of conditions for learners to produce useful results. Among others we have the following: (1) Overfitting is to be avoided. For instance the presence of noise may aggravate overfitting, i.e., existence of mislabeled instances contradicting the class labels of another similar record may be a liability in this sense. (2) Representativeness does not imply learnability. Lack of representative instances in the training data may be useless but at the same time there may be representative data that may not be learnable. Therefore, features have to aid learnability. (3) Feature engineering is a key factor for obtaining good results.

As in our case the learner models do not overfit and the number of features is reduced, we may conclude that either the discarded features do not contribute to representativeness or they do not impact significantly the learner's performance because they do not contribute enough to learnability. As the features were engineered with specific cost-related criteria the latter seems to be a better candidate to explain the results. Also one may raise the question on how these set of classifiers may behave with other datasets. From our perspective the answer is already given by Domingos [8] when he states the conditions for a classifier to be useful.

There is an obvious similitude in the set of features and their relative importance concerning the Ada and ROC learners as seen in Table VI. This behaviour is most likely due to the known equivalence between Ada boosted learners and rank based learners (which use the same ROC AUC criteria) [21]. Although similarity verifies as expressed in Tables IV and V, performance results are not identical which is a direct consequence of the dissimilitude of the learners themselves.

It also noteworthy that there is virtually no difference between the results regarding Tables IV and V for the Boosted Logistic Regression (used as a classifier as remarked before), certainly due to specific properties of boosting. In this context Duchi et al. [9] have shown that for some types of Logistic Boosting the selective elimination of features had only a marginal effect on the test error.

Table VII shows in detail the performance differential from the complete feature set case to the reduced feature set case. For decision trees algorithms `RPART` and `C5.0` there

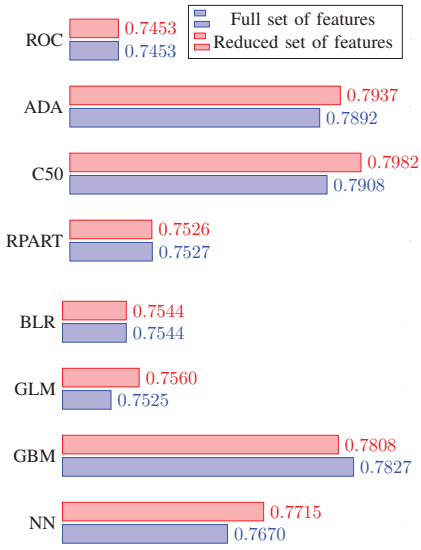


Fig. 1. Accuracy values.

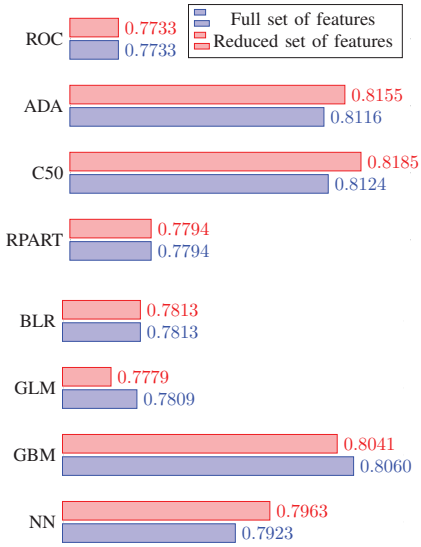


Fig. 2. Balanced accuracy values.

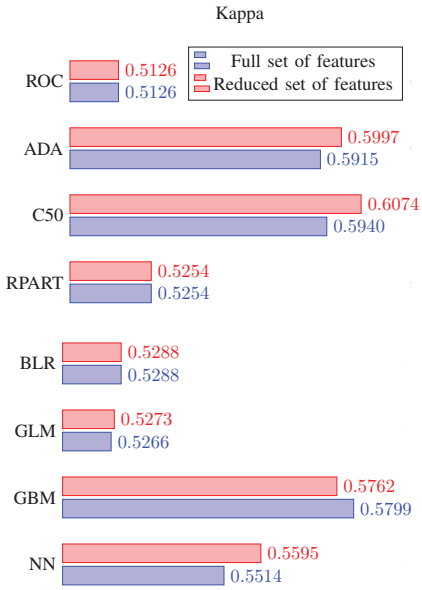


Fig. 3. Kappa values.

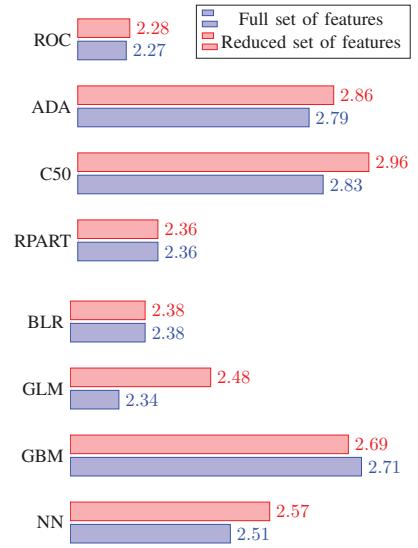


Fig. 4. Positive Likelihood Ratio values.

is clearly a lack of significantive change from the full set case to the reduced feature set case, with the exception of `Neg Pred Value` for `RPART`. This is likely due to the efficiency characteristics of decision trees, already noticed for small datasets and for the `C4.5` classifier (predecessor of the `C5.0` used here) [13].

In the case of neural networks (NN) feature reduction did not affect significantly the metrics obtained. It is known that neural networks are sensitive to the variety of train features used [11], [18], [24]. This relates to the setting of the internal network weights that are directly influenced by the input information. In our case as the number of features was reduced, the information for training is also less. As such this is not an expected result, although admittedly possible if the discarded features have brought no information.

The values for the Positive Likelihood Ratio, as illustrated in Figure 4 are of the same order of magnitude in all cases. This magnitude of the Likelihood Ratio validates the feature reduced dataset as well as the starting dataset in what concerns the positive assertion that this feature set was the result of malicious activity.

The effect of feature tuning was a reduction of the number of features. The number of features reduced for each algorithm is shown in the last row of Table VI. It is clear that similar results for similar performance metrics can be obtained at savings of a variable number of features (see also Tables IV and V). `GBM` has come up as the most feature-efficient learner, closely followed by `RPART`.

One common criteria for model acceptance is that the overall accuracy shall be higher than the no-information rate.

This latter metric was equal to 0.5709 in all algorithms (so it was omitted from the tables). It is remarkable that in any tested case the *p-value* or, equivalently, the probability that the accuracy results are obtained by chance are nearly null. This means that there is a significative relation between the features and their attributed class. In all cases the overall accuracy validates at over a 90% significance level with a *p-value* nearly null: 2.2^{-16}.

VII. Conclusions

Several conclusions can be extracted from the comparative analysis of the algorithms. A number of metrics have been used for this comparison, as seen in Tables IV and V. The set of algorithms tested proved to convey equivalent learning models and results under the same test conditions. This supports the conclusion of Shiravi at al. [23] regarding the quality of the UNB ISCX dataset. One of the main objections raised to this dataset was related to the disparity of results obtained. However, our results with the dataset seem to indicate similar results given the same experimental conditions with an extended variety of machine learning algorithms.

We have confirmed the validity of the ROC criteria for feature reduction. In this regard comparing the performance of classifiers with and without feature reduction proved favourable for the feature reduction case where performance metrics have shown to remain stable without loosing generalization power due to an eventual marked increase in accuracy. Examining the results from the feature reduction we observed that the resulting set of features and their importance varies considerably from learner to learner.

Our results show that feature optimality is classifier-dependent, at least for the ROC selection criteria. GBM has come up as the most feature-efficient learner, closely followed by RPART. Despite the disparity of results regarding features to discard, the learners have show substantial similitude regarding the metrics observed. This fact may hint at the value of ROC based ranking learners in choosing the best features case to case, learner to learner.

Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

[1] G. Anthes. Deep learning comes of age. *Communications of the ACM*, 56(6):13–15, 2013.

[2] I. Arel, D. C. Rose, and T. P. Karnowski. Deep machine learning: A new frontier in artificial intelligence research. *IEEE Computational Intelligence Magazine*, 5(4):13–18, 2010.

[3] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, July 1997.

[4] L. Breiman. Arcing the Edge. Technical report, Statistics Department, University of California, Berkeley, June 1997.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[6] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37 – 46, 1960.

[7] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

[8] P. Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, Oct. 2012.

[9] J. Duchi and Y. Singer. Boosting with structural sparsity. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 297–304. ACM, 2009.

[10] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

[11] G. D. Garson. Interpreting neural-network connection weights. *AI Expert*, 6(4):46–51, Apr. 1991.

[12] J. Grana, D. Wolpert, J. Neil, D. Xie, T. Bhattacharya, and R. Bent. A likelihood ratio anomaly detector for identifying within-perimeter computer network attacks. *J. Netw. Comput. Appl.*, 66(C):166–179, May 2016.

[13] L. B. Holder. Intermediate decision trees. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, pages 1056–1062, 1995.

[14] N. Kayacik and M. Heywood. Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets. In *The 3rd Annual Conference on Privacy, Security and Trust*, 2005.

[15] M. Kuhn. Building predictive models in R using the caret package. *R Project*, 2008.

[16] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, 1994.

[17] H. A. Nguyen and D. Choi. Application of data mining to network intrusion detection: Classifier selection model. In *11th Asia-Pacific Network Operations and Management Symposium*, pages 399–408, 2008.

[18] J. D. Olden and D. A. Jackson. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154(1):135–150, 2002.

[19] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012.

[20] B. P. Roe, H.-J. Yang, and J. Zhu. Boosted decision trees, a powerful event classifier. In *Statistical problems in particle physics, astrophysics and cosmology. Proceedings, Conference*, pages 139–142, 2005.

[21] C. Rudin and R. E. Schapire. Margin-based ranking and an equivalence between adaboost and rankboost. *Journal of Machine Learning Research*, 10:2193–2232, Dec. 2009.

[22] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[23] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, May 2012.

[24] M. Stevenson, R. Winter, and B. Widrow. Sensitivity of feedforward neural networks to weight errors. *IEEE Transactions on Neural Networks*, 1(1):71–80, Mar 1990.

[25] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan. Cost-based modeling for fraud and intrusion detection: Results from the jam project. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, pages 130–144, 2000.

[26] D. B. D. Strother H. Walker. Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 54(1/2):167–179, 1967.

[27] S. Suthaharan and T. Panchagnula. Relevance feature selection with data cleaning for intrusion detection system. In *Southeastcon, 2012 Proceedings of IEEE*, pages 1–6, March 2012.

[28] J. A. Swets. The relative operating characteristic in psychology. *Science*, 182(4116):990–1000, 1973.

[29] A. Viera and J. Garrett. Understanding interobserver agreement: The kappa statistic. *Family Medicine*, 37, 5 2005.

[30] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda. Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th ACM Annual Computer Security Applications Conference*, 2013.

Regression Metrics	ADA	ROC	C5.0	RPART	BLR	GLM	GBM	NN
Accuracy	0.7892	0.7453	0.7908	0.7526	0.7544	0.7525	0.7827	0.7670
Kappa	0.5915	0.5126	0.5940	0.5254	0.5288	0.5266	0.5799	0.5514
Specificity	0.6533	0.5756	0.6602	0.5906	0.5922	0.5806	0.6415	0.6135
Sensitivity	0.9699	0.9711	0.9646	0.9682	0.9703	0.9812	0.9706	0.9712
Neg Pred Value	0.9665	0.9636	0.9612	0.9611	0.9637	0.9763	0.9667	0.9659
Pos Pred Valuee	0.6777	0.6323	0.6809	0.6399	0.6414	0.6375	0.6705	0.6538
Likelihood Ratio(+)	2.79	2.27	2.83	2.36	2.38	2.34	2.71	2.51
Balanced Accuracy	0.8116	0.7733	0.8124	0.7794	0.7813	0.7809	0.8060	0.7923

TABLE IV. PERFORMANCE WITHOUT REDUCTION BY FEATURE SELECTION.

Regression Metrics vs. learner	ADA	ROC	C5.0	RPART	BLR	GLM	GBM	NN
Accuracy	0.7937	0.7453	0.7982	0.7527	0.7544	0.7560	0.7808	0.7715
Kappa	0.5997	0.5126	0.6074	0.5254	0.5288	0.5273	0.5762	0.5595
Specificity	0.6614	0.5756	0.6753	0.5906	0.5922	0.6234	0.6398	0.6216
Sensitivity	0.9696	0.9711	0.9617	0.9682	0.9703	0.9324	0.9684	0.9709
Neg Pred Value	0.9666	0.9636	0.9592	0.9612	0.9637	0.9246	0.9642	0.9660
Pos Pred Valuee	0.6828	0.6323	0.6900	0.6400	0.6414	0.6504	0.6689	0.6585
Likelihood Ratio(+)	2.86	2.28	2.96	2.36	2.38	2.48	2.69	2.57
Balanced Accuracy	0.8155	0.7733	0.8185	0.7794	0.7813	0.7779	0.8041	0.7963

TABLE V. SUMMARY PERFORMANCE WITH A REDUCED FEATURE SET DETERMINED BY FEATURE SELECTION.

Feature vs. Learner	ADA	ROC	C5.0	RPART	GLM	GBM	BLR	NN
dst-host-count	60.05	60.05	0	1.89	30.62	0	60.05	20.73
dst-bytes	100.00	100.00	46.67	0	0	26.50	100.00	0
count	83.64	83.64	41.89	15.61	67.87	7.56	83.64	55.17
src-bytes	99.92	99.92	100.00	100.00	0	100.00	99.92	63.93
dst-bytes	0	0	0	92.04	0	0	0	54.91
duration	0	0	1.35	0	15.57	0.82	0	20.10
srv-count	0	0	0	8.68	0	1.48	0	34.79
logged-in	87.21	87.21	45.44	0	0	0	87.21	24.64
num-shells	0	0	43.88	0	0	0	0	0
num-compromised	0	0	53.64	0	42.17	0	0	0
dst-host-srv-rerror-rate	0	0	0	0	0	0	0	0
dst-host-same-srv-rate	93.00	93.00	45.55	0	0	0.62	93.00	16.34
srv-serror-rate	77.25	77.25	0	0	34.93	0	77.25	11.17
wrong-fragment	0	0	0	0	100.00	0	0	13.70
hot	0	0	48.22	0	69.71	2.80	0	0
service	44.62	44.62	7.14	0	0	0	44.62	100.00
rerror-rate	0	0	0	0	0	0	0	13.59
serror-rate	80.78	80.78	0	0	0	0	80.78	11.57
urgent	0	0	0	0	0	0	0	0
dst-host-same-src-port-rate	44.34	44.34	55.57	0	63.76	4.11	44.34	34.36
is-guest-login	0	0	0	0	77.82	0	0	0
dst-host-srv-count	98.55	98.55	49.73	5.91	72.05	5.96	98.55	36.81
protocol-type	40.28	40.28	52.98	6.70	64.45	5.32	40.28	29.51
num-root	0	0	0	0	42.43	0	0	0
srv-rerror-rate	0	0	0	0	38.99	0	0	0
dst-host-rerror-rate	0	0	47.93	0	28.79	0	0	0
srv-diff-host-rate	45.80	45.80	0	0	27.78	0	45.80	0
num-file-creations	0	0	0	0	21.26	0	0	0
dst-host-srv-serror-rate	76.06	76.06	2.52	0	17.25	0	76.06	0
dst-host-serror-rate	81.80	81.80	0	0	16.59	0	81.80	11.07
same-srv-rate	94.68	94.68	43.92	81.85	15.27	0	94.68	0
diff-srv-rate	88.63	88.63	0	80.33	0	0	88.63	0
dst-host-srv-diff-host-rate	54.95	54.95	49.74	4.95	0	0	54.95	15.90
dst-host-diff-srv-rate	84.92	84.92	5.33	0	0	0	84.92	17.33
num-failed-logins	0	0	49.36	0	0	0	0	0
flag	91.11	91.11	47.75	75.56	47.30	0	91.11	88.48
Number of discarded features	20	20	20	29	20	30	20	20

TABLE VI. RELATIVE IMPORTANCE FOR EACH FEATURE BY LEARNER REPRESENTED IN A RELATIVE PERCENTUAL SCALE.

Regression Metrics vs. learner	ADA	ROC	C5.0	RPART	BLR	GLM	GBM	NN
Accuracy	1	0	1	0	0	0	0	0
Kappa	1	0	2	0	0	0	-1	0
Specificity	1	0	2	0	0	7	0	1
Sensitivity	0	0	0	0	0	-5	0	0
Neg Pred Value	0	0	0	0	0	-6	0	0
Pos Pred Value	1	0	1	0	0	2	0	0
Balanced Accuracy	0	0	1	0	0	0	0	0

TABLE VII. SUMMARY OF DIFFERENCES BETWEEN ORIGINAL AND FEATURE REDUCED DATASETS. VALUES ARE EXPRESSED IN PERCENTAGE. VALUES LESS THAN ONE ARE REPRESENTED AS NULLS.

A Security Policy Query Engine for Fully Automated Resolution of Anomalies in Firewall Configurations

Ahmed Bouhoula
Lycée Louis le Grand – 123 Rue Saint-Jacques
75005 Paris, France
Email: ahmed.bouhoula@gmail.com

Anis Yazidi
Department of Computer Science – Oslo and Akershus
University College – Oslo, Norway
Email: anis.yazidi@hioa.no

***Abstract*—Legacy work on correcting firewall anomalies operate with the premise of creating totally disjunctive rules. Unfortunately, such solutions are impractical from implementation point of view as they lead to an explosion of the number of firewall rules. In a related previous work, we proposed a new approach for performing assisted corrective actions, which in contrast to the-state-of-the-art family of radically disjunctive approaches, does not lead to a prohibitive increase of the configuration size. In this sense, we allow *relaxation* in the correction process by clearly distinguishing between constructive anomalies that can be tolerated and destructive anomalies that should be systematically fixed.**

However, a main disadvantage of the latter approach was its dependency on the guided input from the administrator which controversially introduces a new risk for human errors. In order to circumvent the latter disadvantage, we present in this paper a Firewall Policy Query Engine (FPQE) that renders the whole process of anomaly resolution a fully automated one and which does not require any human intervention. In this sense, instead of prompting the administrator for inserting the proper order corrective actions, FPQE executes those queries against a high level firewall policy. We have implemented the FPQE and the first results of integrating it with our legacy anomaly resolver are promising.

I. Introduction

With the dramatic growth of the Internet, network security has become a focal concern during this last decade. Firewalls are widely deployed in private networks as an inherent part of their security. However, the effectiveness of a firewall is dramatically jeopardized by the presence of anomalies within its filtering rules. In fact, the filtering rules may include anomalies resulting in critical security vulnerabilities. The analysis of filtering rules and anomalies discovery have gained a lot of attention. A significant work was reported in this area [1]–[9]. Most of the emphasis has been given to the classification and discovery of firewall anomalies. Other studies have focused on optimizing the filtering process time [10]–[12]. However, few studies have been performed to resolve these anomalies. The most notable of these studies are [1] and [13] which only focused on one of the conflict problems, namely, rule correlation in filtering policies. Other remarkable studies [14], [15] have defined a set of recommendations for correcting policy anomalies. However [14] and [15] did not develop a concrete approach to resolve packet filter conflicts. Another study that tried to probe into the conflict resolution issue is reported in [16]. The main shortcoming of [16] is that the proposed corrective actions do not handle the correlation anomaly. In [17], [18], a set of algorithms for rewriting firewall rules were presented. The new rewritten rules are completely free of errors and equivalent to the initial misconfigured firewall rules. However, the complexity of a rule re-writing algorithm is very high and leads to an explosion in the number of the firewall rules. Dealing with an assisted correction of policy anomalies seems to be a captivating and challenging task. In fact the correction is a highly complicated task that threatens to overwhelm human attention as the number of rules increases. Moreover, this task is prone to errors. In fact, modifying the rules order may create new anomalies instead of correcting the existent ones. Hence, a special attention has to be paid when ordering the rules. From this perspective raises the need of an automatic system to correct anomalies within filtering rules. In our recent work reported in [19], we prove that ordering does not always work to correct the anomalies defined in [14]. We should underline that our corrective system presented in [19] is not fully automatic. Our claim was that corrective system is guided by the administrator choices in order to reflect exactly the desired policy. However, paradoxically, this claim is questionable since by querying the administrator, we might introduce a new level of error during the correction phase. In this paper, we present a Firewall Policy Query Engine (FPQE) which stores the high level network policy. The FPQE can be assimilated to an Oracle that replaces the administrator in answering to queries about the relative order of correlated rules or conflicting rules [19]. The FPQE query engine is integrated with the resolver presented in [19] so that to guide the correction process whenever there is ambiguity concerning the relative order of conflicting rules.

978-1-5090-3216-7/16/$31.00

II. Modelling The FPQE

FPQE receives as input the Security Policy (SP) which is a set SP of directives which are formulae defining whether

packets are accepted or denied.

SP is called consistent if the domain of accepted packets and the domain of denied packets are disjoint.

An SP is presented as a finite set of directives[1]:

$$SP = \{(s_i, d_i, a_i) \textbf{ Except } \{(s_{ij}, d_{ij}) | 1 \leq j \leq k_i \text{ and } s_{ij} \subseteq s_i \text{ and } d_{ij} \subseteq d_i\} | 1 \leq i \leq m\}$$

For each i, s_i is the source address of the directive, d_i is the destination address and a_i is the action of the directive (Accept or Deny). m denotes the number of directives and k_i is the number of exceptions from the i^{th} directive. A directive may include a set of exceptions in the form (s_{ij}, d_{ij}).

Representing a Security Policy with a Boolean Formula:

A security policy SP can be transformed into a Boolean formula B_{SP} such as B_{SP} evaluates to True with an assignment which corresponds to a packet p if and only if SP accepts the packet p [20].

$$\forall p,\ SP\ accepts\ p\ \Leftrightarrow p\ \vDash B_{SP}$$

For each i, let $B_i(p)$ be the Boolean formula corresponding to directive $r_i = (s_i,\ d_i,\ a_i)$. Let D be the default action which has to be used if no directive in SP contains the domain of the input packet p. In more formal terms, this is expressed as:

$$B_{SP}(p) = \left(\bigvee_{i=1}^{m} B_i(p)\right) \vee (D \wedge \forall i \in [1..m] \,\mathrm{dom}(p) \notin \mathrm{dom}(B_i))$$

For each i, and each packet $p = (sp,\ dp)$, if the action equals Accept (a_i=Accept), then the input packet is accepted if its domain is included in the domain of the directive and not included in the domain of all the exceptions. Formally, this is given by:

$$B_i(p) = ((sp\ \subseteq s_i) \wedge (dp\ \subseteq d_i)) \wedge (\bigwedge_{j=1}^{k_i} (sp \nsubseteq\ s_{ij}) \vee (dp \nsubseteq\ d_{ij}))$$

If the action equals Deny (a_i=Deny) then the input packet is accepted if its domain is included in the domain of one of the exceptions. Formally, this is given by:

$$\mathrm{B}_i(p) = \bigvee_{j=1}^{k_i} ((sp\ \subseteq s_{ij}) \wedge (dp\ \subseteq d_{ij}))$$

Filtering rules:

We will consider the access list syntax used in Cisco routers (see [21]). Here an example of such syntax:

access-list 100 permit tcp host 10.11.1.2 host 170.16.10.6

This rule permits tcp traffic from 10.11.1.2 host machine to 170.16.10.6 host machine. In the following and for simplicity issues we will represent a firewall filtering rule as a triplet (s_i, d_i, a_i) where s_i is the source address of the rule, d_i is the destination address and a_i is the action (Accept or Deny).

Representing Filtering rules with a Boolean Formula:

A list of filtering rules FR can be transformed into a Boolean formula B_{FR} such as B_{FR} evaluates to True with an assignment which corresponds to a packet p if and only if FR accepts the packet p.

$$\forall p,\ FR\ accepts\ p\ \Leftrightarrow p \vDash B_{FR}$$

Let FR be a list of filtering rules and an input packet $p = (sp,\ dp)$, the following inference system returns the Boolean formula which corresponds to FR: $B_{FR}(p)$

$$\text{Accept Rule } \frac{(s_i,\ d_i,\ accept) :: rest}{((sp\ \subseteq s_i) \wedge (dp\ \subseteq d_i)) \bigvee B_{rest}(p)}$$

The Accept Rule applies if the action of the first rule in FR is accept. The packet is accepted if it is matched with the domain of the rule OR it is accepted by the *rest* of the filtering rules.

$$\text{Denied Rule } \frac{(s_i,\ d_i,\ deny) :: rest}{((\mathrm{sp}\ \nsubseteq s_i) \vee (dp\ \nsubseteq d_i)) \bigwedge B_{rest}(p)}$$

The Denied Rule applies if the action of the first rule in FR is deny. The packet is accepted if it is not matched with the domain of the rule AND the *rest* of the rules accept it.

$$\text{Default Action } \frac{[\,]}{V} \text{ where V = False}$$

If *rest* becomes empty list, we return False since we assume that the default action is deny.

At this juncture, we shall explain *how to check whether a set of rules are correct and complete with respect to a security policy.* Let FR be a set of filtering rules, SP a security policy, B_{FR} and B_{SP} the corresponding Boolean formulae.

FR is correct with respect to SP iff:

$$\forall p,\ B_{FR}(p) \Leftrightarrow B_{SP}(p)$$

If $B_{FR}(p) \nLeftrightarrow B_{SP}(p)$ is not satisfiable, then FR is correct with respect to SP, otherwise it is not correct with respect to SP. We use MinSAT solver [22] to check satisfiability.

$$\nexists p,\ B_{FR}(p) \nLeftrightarrow B_{SP}(p)$$

If FR is not correct with respect to SP, we give a packet which is a counter-example and the first rule in FR which is not correct with respect to SP.

A. Issuing the queries in FPQE with a pair of anomalous rules

In the previous section, we have explained how to check the validity of the *whole* firewall filtering rules with respect to the security policy. However, as explained in [19], the resolution of anomalies relies on checking the validity of the order of a *pair* of correlated rules, as well as checking which rule should apply in case of contradiction. It is important to note

[1]For the sake of simplicity issues and without loss of generality, we will not consider the source port, the destination port and the protocol of the directives.

that generally, if we consider the validity of two rules R_x and R_y with respect to the SP, then Boolean formula for the equivalence with the SP must be modified to accommodate the fact that we are considering the subset of the FR composed of R_x and R_y and not the whole list of filtering rules FR. Hence, the correct expression that accommodate this is:

$$\forall p \in dom(R_x) \cup dom(R_y), B_{FR}(p) \Leftrightarrow B_{\text{SP}}(p)$$

At this juncture, we give the expression of the query that we issue given two rules R_x and R_y that are either correlated or in contradiction, where R_x precedes R_y:

$$\nexists p \in dom(R_x) \cup dom(R_y), B_{\text{FR}}(p) \nLeftrightarrow B_{\text{SP}}(p) \qquad (1)$$

We shall now consider two cases relevant for the correction of anomalies, namely, correlation and contradiction:

Case 1: R_x ***and*** R_y ***are correlated:*** If the above expression is verified (Eq. 1), and if R_x and R_y present the correlation anomaly, then their order must be inverted. In fact, as explained in [19], this means that we found a counter-example packet that both matches R_x and R_y and that has wrong action which does not comply with the SP. Thus, the relative order of R_x and R_y is wrong. Hence, by inverting the order of R_x and R_y, the packets that are matched in same time by R_x and R_y will be assigned the right action that complies with the SP. If the above expression is not verified (Eq. 1), and if R_x and R_y present the correlation anomaly, then the order of the correlated rules R_x and R_y complies with the policy and should be preserved.

Case 2: R_x ***and*** R_y ***are in contradiction:*** If Eq. 1 is verified, then a counter-example is found and thus the filtering rules do not comply with SP. In this case, rule R_y shall be removed. Otherwise, rule R_x should be removed.

III. Integration of the FPQE with The Firewall Policy Resolver

In this section, we shall explain how the FPQE is integrated with our recent firewall policy resolver presented in [19]. We propose a set of corrective actions which are based on our previous work and which involve the FPQE:

Automatic rule removal: In the case of redundancy it is recommended to remove the specific rule which is included in the general rule.

Automatic rule permutation: In the case of shadowing it is recommended to permute the conflicting rules in order to obtain the anomaly generalization which is tolerated.

Automatic rule removal using FPQE: This action is applied in the case of contradiction anomaly. The FPQE query engine is invoked so that to choose the rule to be deleted depending on the security policy requirements[2].

Automatic rule permutation using FPQE: In the case of correlation the FPQE query engine chooses the proper order that complies with the adopted policy by issuing a query that determines the action for the set of packets lying at the intersection of the correlated rules. The resolver [19] tolerates the existence of the generalization anomaly and we will consider the rest of anomalies as policy errors to resolve. In fact, generalization is often used to exclude a specific part of the traffic from a general filtering action. Therefore, we tolerate the existence of this anomaly. Shadowing is considered as a critical error in the firewall policy as the shadowed rule never applies, resulting in an unexpected filter response for the packets that match the shadowed rule. Shadowing can be resolved by reversing the relative order of the two involved rules. In the case of correlation, a packet whose header matches the intersection of the headers of the two filtering rules causes ambiguity in packet classification because the adopted policy depends on the order of the two conflicting rules. In this case, the FPQE is queried with pair of rules to determine the order of the rules that complies with his desired policy. In the case of contradiction, the shadowed rule will never be activated. Therefore, the FPQE is used to choose which of the two conflicting rules should apply. The other rule will be removed from the filtering rules list since it increases unnecessarily the number of filtering rules.

The FPQE works in interaction with the resolver [19]. The resolver analyses the filtering rules and reports the detected anomalies. It will interact the FPQE whenever there is an ambiguity in the order of the rules. FPQE is written in C++. After interaction with the FPQE, the correction of the anomalies proceeds as described in [19] by topologically sorting the rules based on the precedence relationships and eventual cycle breaking. Thus, the only difference between [19] and the advocated approach in this paper is the fact that we rather rely on the FPQE query engine instead of prompting the administrator for decisions concerning the order of conflicting rules. Since the aim of this paper is to shed light on the FPQE query engine, details about the correction process described in [19] are omitted here. Fig. 1 illustrates the interaction between the FPQE and the anomaly resolver system described in [19].

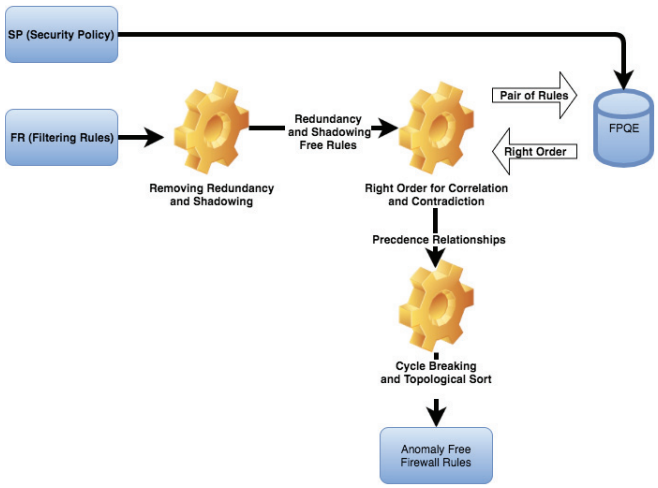


Fig. 1. Reporting the detected anomalies

[2]Please note that in [19] we rely on administrator while in this paper we resort to the FPQE query engine instead.

IV. Example: Checking FR with respect to SP

In order to transform the security policy SP and the filtering rules FR into Boolean formulae, the source and destination IP addresses will be converted into 32 Boolean variables (each address can be written in 32 bits). Thus, checking whether the domain of a packet p is included in the domain of a filtering rule reduces to verifying the satisfiability of some Boolean expression. For example let us consider the addresses $Src_1 = 120.*.*.*$ and $Src2 = 120.127.*.*$. In order to check whether $Src_2 \subset Src_1$, we have to consider the 32 variables assigned to source addresses. Note that since $120 = 01111000_2$, then the Boolean forumla for checking the inclusion within $Src1$ is given by:

$$\neg a_1 \wedge a_2 \wedge a_3 \wedge a_4 \wedge a_5 \wedge \neg a_6 \wedge \neg a_7 \wedge \neg a_8$$

In order to illustrate the possibility of correcting the order of the FR filtering rules automatically without interaction with the network administrator, we consider a simple example. We suppose a university has it own internal network with three domains. D_0 denotes the whole network domain of the university. D_1 is a sub-domain including the students and professors. D_2 is a sub-domain including the Professors and Administration. For the sake of simplicity we suppose that D_0 contains ip addresses that are integers and we assume that $D_0 = [1, 7]$, $D_1 = [1, 4]$ and $D_2 = [4, 7]$.

We suppose that security policy SP is defined by the following directive:

$$(D_0, any, accept) \textbf{ Except } \text{(Students, facebook)}$$

The above directive states that all users in D_0 can access any website except the students can not access Facebook.

In addition, we suppose that the filtering rules FR are composed of R_1 and R_2 given by:

$$R_1 : (D_1, Facebook, deny)$$
$$R_2 : (D_2, Facebook, accept)$$

Note that by definition, $D_1 = [1, 4]$ can be expressed as:

$$D_1 = \neg a_1 \vee (a_1 \wedge \neg a_2 \wedge \neg a_3)$$

Similarly, $D_2 = [4, 7]$ can be expressed as:

$$D_2 = a_1$$

Thus, the security policy SP is given by:

$$\begin{aligned} B_{\text{SP}}(p) = \neg D_1 \vee D_2 &= [a_1 \wedge (\neg a_1 \vee a_2 \vee a_3)] \vee a_1 \\ &= (a_1 \vee a_1) \wedge (a_1 \vee \neg a_1 \vee a_2 \vee a_3) \\ &= a_1 \wedge (T \vee a_2 \vee a_3) \\ &= a_1 \end{aligned}$$

On the other hand, the filtering rules FR are given by:

$$\begin{aligned} B_{\text{FR}}(p) = \neg D_1 \wedge D_2 &= [a_1 \vee (\neg a_1 \vee a_2 \vee a_3)] \wedge a_1 \\ &= a_1 \vee (\neg a_1 \vee a_2 \vee a_3) \end{aligned}$$

Now, we are ready to check whether the FR is equivalent to SP using the following Boolean formula.

$$\begin{aligned} B_{\text{FR}}(p) \Leftrightarrow B_{\text{SP}}(p) = \neg a_1 \Leftrightarrow [a_1 \vee (\neg a_1 \vee a_2 \vee a_3)] \\ = \underbrace{[\neg a_1 \Rightarrow (\neg a_1 \vee a_2 \vee a_3)]}_{S_1} \\ \wedge \underbrace{[(\neg a_1 \vee a_2 \vee a_3) \Rightarrow \neg a_1]}_{S_2} \end{aligned}$$

We compute S_1 as:

$$\begin{aligned} S_1 &= \neg a_1 \Rightarrow (\neg a_1 \vee a_2 \vee a_3) \\ &= a_1 \vee (\neg a_1 \vee a_2 \vee a_3) \\ &= a_1 \end{aligned}$$

Similarly, we compute S_2 as:

$$\begin{aligned} S_2 &= (\neg a_1 \vee a_2 \vee a_3) \Rightarrow \neg a_1 \\ &= \neg a_1 \vee (a_1 \wedge \neg a_2 \wedge \neg a_3) \vee \neg a_1 \\ &= (a_1 \vee \neg a_1) \wedge (\neg a_2 \wedge \neg a_1) \wedge (\neg a_3 \wedge \neg a_1) \\ &= (\neg a_2 \wedge \neg a_1) \wedge (\neg a_3 \wedge \neg a_1) \end{aligned}$$

Thus, we finally obtain:

$$B_{\text{FR}}(p) \Leftrightarrow B_{\text{SP}}(p) = a_1 \wedge (\neg a_2 \wedge \neg a_1) \wedge (\neg a_3 \wedge \neg a_1) \quad (2)$$

We deploy a MiniSAT SAT Solver in order to obtain a counter example, i.e, a packet p that satisfies $B_{\text{FR}} \Leftrightarrow B_{\text{SP}}$. The MiniSat SAT solver expects input to be in Conjunctive Normal Form (CNF) in the DIMACS format [23].

The input of the MiniSAT SAT Solver corresponding to Eq. 2 is given by:

```
p cnf 3 3
1 0
-1 -2 0
-1 -3 0
```

As seen in Fig. 2, the output returns a counter-example: $a_1 = T$, $a_2 = F$, and $a_3 = F$.

Therefore the FR and the SP are not equivalent, which means that the order of the two correlated rules R_1 and R_2 must be inverted.

V. Conclusion

Firewall rules misconfiguration is a substantial issue in the area of network security. Valuable studies in this field have provided an answer to the anomalies discovery issue. However, the correction of these anomalies is still a rich axis of research. This task is extremely delicate and poses serious challenges with regards to the importance of rules order. In this paper, we have proposed a fully automated approach and set of tools for correcting anomalies within filtering rules. The correction is guided by the FPQE engine to yield a required precision while reflecting the adopted security policy without requiring any intervention from the administrator.

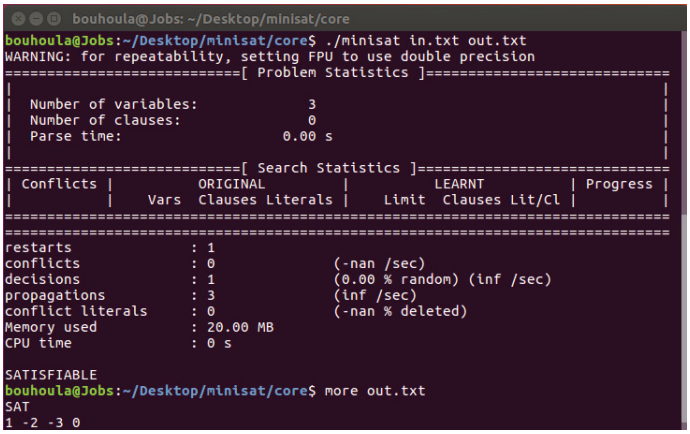


Fig. 2. Screenshot illustrating the execution of MiniSAT SAT Solver and showing the output in DIMACS format.

As we mentioned previously, the MiniSat SAT solver expects input to be in conjunctive normal form. The transformation of Boolean formulas into CNF format will increase their size exponentially. However, the Tseitin technique [24] allows a translation to equi-satisfiable CNF formulas with linear increase of the size but it needs to add new variables. Currently, we are working on using Limboole [25] instead of the MiniSat Sat solver, which allows to check satisfiability on arbitrary formulas, and not just satisfiability for formulas in conjunctive normal form, like the DIMACS format. This allows us to handle large set of security policy and filtering rules, in reasonably good time. Furthermore, we plan to integrate the FPQE with other security equipments that are prone to policy configuration errors such as intrusion detection systems.

REFERENCES

[1] A. Hari, S. Suri, and G. Parulkar, "Detecting and resolving packet filter conflicts," in *Proceedings of INFOCOM 2000*, vol. 3. IEEE, 2000, pp. 1203–1212.

[2] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE, 1999, pp. 17–31.

[3] J. D. Guttman, "Filtering postures: Local enforcement for global policies," in *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE, 1997, pp. 120–129.

[4] S. Hinrichs, "Policy-based management: Bridging the gap," in *Proceedings of the 15th Annual Conference on Computer Security Applications (ACSAC'99)*. IEEE, 1999, pp. 209–218.

[5] A. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*. IEEE, 2000, pp. 177–187.

[6] A. Wool, "Architecting the lumeta firewall analyzer." in *USENIX Security Symposium*, 2001, pp. 85–97.

[7] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," in *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, 2001, pp. 100–107.

[8] C. Pornavalai and T. Chomsiri, "Firewall policy analyzing by relational algebra," in *The 2004 International Technical Conference on Circuits/Systems, Computers and Communications*, 2004.

[9] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "Fireman: A toolkit for firewall modeling and analysis," in *2006 IEEE Symposium on Security and Privacy*. IEEE, 2006, pp. 15–pp.

[10] C. Benecke, "A parallel packet screen for high speed networks," in *Proceedings of the 15th Annual Conference on Computer Security Applications (ACSAC'99)*. IEEE, 1999, pp. 67–74.

[11] H. Hamed, A. El-Atawy, and E. Al-Shaer, "Adaptive statistical optimization techniques for firewall packet filtering," in *Proceedings of INFOCOM 2006*, 2006, pp. 1–12.

[12] ——, "On dynamic optimization of packet matching in high-speed firewalls," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1817–1830, 2006.

[13] D. Eppstein and S. Muthukrishnan, "Internet packet filter management and rectangle geometry," in *Proceedings of the 2001 ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2001, pp. 827–835.

[14] E. S. Al-Shaer and H. H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *IFIP/IEEE Eighth International Symposium on Integrated Network Management*. IEEE, 2003, pp. 17–30.

[15] C. Basile, D. Canavese, A. Lioy, C. Pitscheider, and F. Valenza, "Inter-function anomaly analysis for correct sdn/nfv deployment," *International Journal of Network Management*, vol. 26, no. 1, pp. 25–43, 2016.

[16] A. K. Bandara, A. Kakas, E. C. Lupu, and A. Russo, "Using argumentation logic for firewall policy specification and analysis," in *Large Scale Management of Distributed Systems*. Springer, 2006, pp. 185–196.

[17] F. Cuppens, N. Cuppens-Boulahia, and J. G. Alfaro, "Detection of network security component misconfiguration by rewriting and correlation," in *Joint Conference on Security in Network Architectures and Security of Information Systems*, 2006.

[18] J. García-Alfaro, F. Cuppens, and N. Cuppens-Boulahia, "Towards filtering and alerting rule rewriting on single-component policies," in *Computer Safety, Reliability, and Security*. Springer, 2006, pp. 182–194.

[19] A. Yazidi and A. Bouhoula, "On assisted packet filter conflicts resolution: An iterative relaxed approach," in *Proceedings of the 41st Annual IEEE Conference on Local Computer Networks (To appear). LCN 2016.* IEEE, 2016.

[20] S. Hazelhurst, "Algorithms for analysing firewall and router access lists," *arXiv preprint cs/0008006*, 2000.

[21] C. S. Inc, "Configuring IP Access Lists," http://www.cisco.com/c/en/us/support/docs/security/ios-firewall/23602-confaccesslists.html, 2007, [Online; accessed 04-August-2016].

[22] C. M. Li, Z. Zhu, F. Manyà, and L. Simon, "Optimizing with minimum satisfiability," *Artificial Intelligence*, vol. 190, pp. 32–44, 2012.

[23] "Minisat sat solver," http://www.dwheeler.com/essays/minisat-user-guide.html, 2006, [Online; accessed 04-August-2016].

[24] G. S. Tseitin, *On the Complexity of Derivation in Propositional Calculus*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 466–483.

[25] "Limboole," http://fmv.jku.at/limboole, 2006, [Online; accessed 28-September-2016].

Enforcement of Global Security Policies in Federated Cloud Networks with Virtual Network Functions

Philippe Massonet, Sébastien Dupont, Arnaud Michot
CETIC Research Center, Charleroi, Belgium
Email: philippe.massonet@cetic.be

Anna Levin
IBM Research Lab, Haifa, Israel
Email: lanna@il.ibm.com

Massimo Villari
University of Messina, Italy
Email:mvillari@unime.it

***Abstract*—Federated cloud networks are formed by federating virtual network segments from different clouds, e.g. in a hybrid cloud, into a single federated network. Such networks should be protected with a global federated cloud network security policy. The availability of network function virtualisation and service function chaining in cloud platforms offers an opportunity for implementing and enforcing global federated cloud network security policies. In this paper we describe an approach for enforcing global security policies in federated cloud networks. The approach relies on a service manifest that specifies the global network security policy. From this manifest configurations of the security functions for the different clouds of the federation are generated. This enables automated deployment and configuration of network security functions across the different clouds. The approach is illustrated with a case study where communications between trusted and untrusted clouds, e.g. public clouds, are encrypted. The paper discusses future work on implementing this architecture for the OpenStack cloud platform with the service function chaining API.**

I. Introduction

With a the growing number of infrastructure cloud services becoming available there are many benefits to interconnecting several cloud services. In hybrid clouds for example one or more private clouds needs to be connected with one or more public clouds so that virtual machines from the private and public clouds can communicate. The hybrid cloud is one example of a cloud federation. Different Cloud federation types such as cloud bursting, cloud brokering or cloud aggregation have been proposed to provide the necessary mechanisms for sharing compute, storage and networking resources.

Federated cloud networking techniques provide mechanisms to federate cloud network resources, and to define an integrated cloud management layer to deploy applications securely and efficiently within the cloud federation. One approach for customising network security is to use network virtualisation technologies such as VLANs in conjunction with Network Function Virtualisation (NFV) and Service Function Chaining (SFC). By combining NFV/SFC with network virtualisation it is possible for cloud tenants to tailor the security of each of their individual virtual networks. In this way the security virtual network functions (VNF) can be tailored to the specific needs of each application. Many different VNF are available such as deep packet inspection, encryption, decryption, intrusion detection or firewalls. SFC can be used to compose the required VNF to meet a tenant's network security requirements.

In this paper, we argue that the deployment and configuration of security VNF across the different clouds of a network federation should be managed by a network federation manager. This component should be responsible for enforcing a coherent global network security policy across the network federation. The approach is illustrated with a motivating example for encrypting and decrypting traffic using SSL with untrusted cloud platforms. The global network security policy is defined in a service manifest. This work builds on a cloud architecture for federating network overlay segments into a federated cloud network [1]. Future work will implement our approach in an OpenStack cloud federation. This is described briefly at the end of the paper.

The paper is organized as follows: Section II describes the motivations for coordinating the deployment of security VNF across the different federated cloud network segments. Section III presents our approach by describing the network service manifest and showing how VNF configurations and deployment are generated. It also illustrates our approach with a motivating example. Section IV presents related work and discuses future work for implementing our approach in an OpenStack cloud federation.

II. Motivations

Federated cloud networks are created by connecting network segments from different clouds [1]. Federated cloud networks must deal with heterogeneous clouds, because they may connect different cloud platforms such as OpenStack, OpenNebula or public clouds such as Amazon AWS or Microsoft Azure. The main benefits of federated cloud networks are the flexibility they provide to customize and manage the federated network according to the specific needs of an application or a tenant. From a security perspective the isolation they provide makes them easier to protect than multi-tenant networks.

This paper shows how to define and enforce global network security policies on federated cloud networks. The security policy is defined in a single service manifest and is then deployed in the different federation clouds using VNF and SFC. The main benefit of this approach is that the deployment,

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

configuration and chaining of the security VNF may be automated and can be verified once they are deployed across the different clouds. Performing this manually across the different clouds by different network administrators would take time and be error prone if the federated cloud network is large and/or the network topology or the security policy change often. The deployment of the security VNF and their chaining across the different cloud network segments is coordinated by a component called the Federated Cloud Network Manager. The approach is illustrated with a case study that encrypts network flows if the destination is untrusted, and does not encrypt it if the destination cloud is trusted.

III. Approach

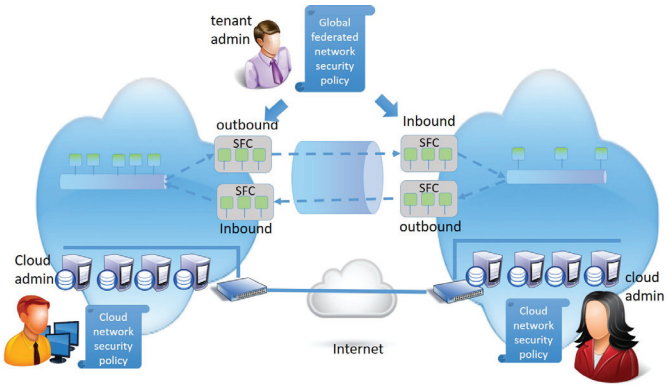


Fig. 1. A Federated Cloud Network Federation

The approach begins with specifying a global network security policy, the generating the VNF configurations for each cloud, and then deploying the VNF in each cloud. Figure 1 shows a federated cloud network distributed across two clouds. The bottom part of the figure shows that each of the two clouds has as physical compute, storage and network resources that are managed by a system administrator and are protected with specific network security policies. The network resources of the two clouds are connected using the internet. The upper part of the figure shows the federated cloud network. In each of the two clouds there is a virtual network segment on which several virtual machines are running. The two network segments are connected by an overlay network [1]. The inbound and outbound flows are routed through an SFC of VNF on each side of the federated network. The top of the figure shows that the administrator of the network federation defines a global network security policy and that VNF configurations are generated to configure the VNF that are deployed as SFC.

Figure 2 shows an encryption/decryption case study with three clouds. The first two clouds are trusted and can communicate without encryption. The third cloud is a public cloud and is untrusted: communications between the first two clouds and the third cloud must be encrypted. Encryption must be done with the public key of the destination cloud. Decryption must be done with the private key of the destination cloud.

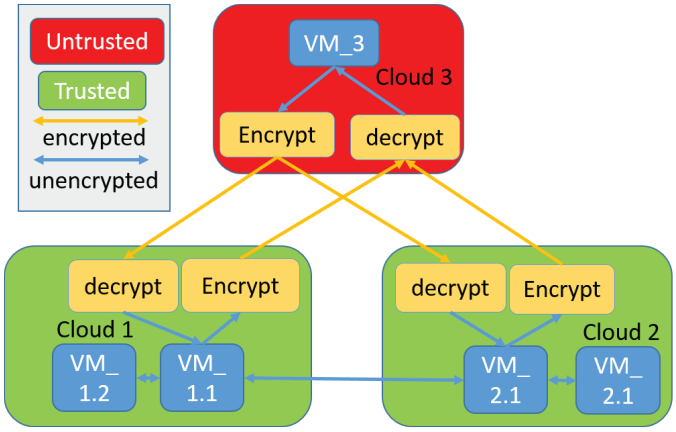


Fig. 2. Cloud case study: trusted and untrusted clouds

The global security policy may be specified as follows and can later be translated into a network program:

if destination cloud X is untrusted **then**
  encrypt network flow with public key of X
else if destination cloud Y is untrusted **then**
  encrypt network flow with public key of Y
else if ... **then**
  ...
else if destination cloud Z is trusted **then**
  do not encrypt network flow
else if source cloud is untrusted **then**
  decrypt the network flow using the private key
else
  destination or source cloud is not part of the federation
end if

Figure 2 shows the resulting implementation of the above global security policy where communications between the three clouds of the federation have been secured according to the federated network security policy. Communications between cloud 1 and cloud 2 are not encrypted since both clouds are trusted. On the other hand, communications with public cloud 3 need to be encrypted because cloud 3 is untrusted. The figure shows how the encryption and decryption Virtual Network Functions have been deployed to secure communications. For example, when VM1 communicates with VM3 all network traffic is encrypted in cloud 1 and systematically decrypted by public cloud 3. All outgoing traffic from VM3 to VM1 is encrypted by cloud 3 and decrypted by cloud 1. In this case the service chaining on each cloud site amounts to an "if ... then .. else": if the traffic destination is an untrusted cloud, then the traffic has to be encrypted.

Figure 3 shows a fragment of a YAML based service manifest (OpenStack HEAT template) and the Python VNF configuration code that is generated. The left part of the figure shows the fragment of the service manifest that defines the encryption VNF from cloud 1 to cloud 3. Encryption is required because cloud3 is an untrusted cloud. The incoming flows are coming from cloud1 virtual machines is unencrypted: this is why the incomingPrivateKey and incomingPublicKey

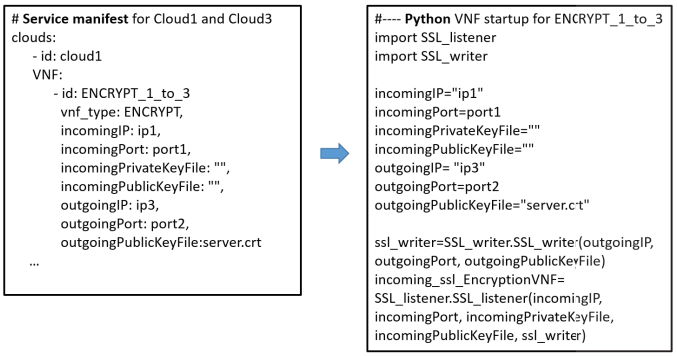


Fig. 3. Service Manifest Fragment and generated VNF Python code

fields are left blank. The incoming traffic is is on incoming IP ip1 and port1. Outgoing traffic to cloud 3 at IP address ip3 and port port2 needs to be encrypted with the public key of cloud 3 "server.crt". The right side of the figure shows the corresponding Python code to configure and start the encryption VNF ENCRYPT_1_to_3. The encryption parameters from the service manifest are translated to Python variables. The encryption VNF encrypts all incoming traffic by creating an instance of the SSL_writer class and passing it as a parameter to the SSL_listener instance. This way all incoming data that is sent to cloud3 is encrypted.

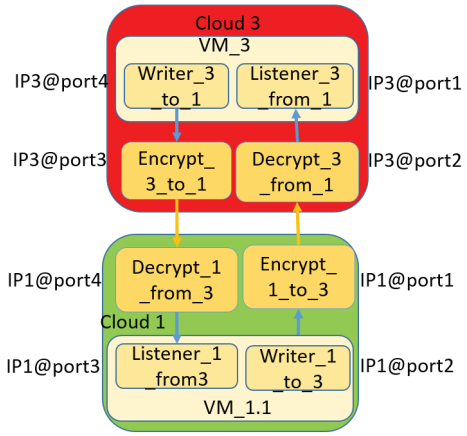


Fig. 4. Ordering and timing constraints for correct startup of VNF

Figure 4 shows the resulting deployment of the encryption and decryption VNF between cloud 1 and cloud3. The figure shows the IP addresses and ports that are used for socket based communication. There are ordering and timing constraints that have to be respected for starting these VNF correctly. The basic constraints come from the use of sockets: listeners must be started before writers because an IP and port must be allocated to listening sockets before writers can connect. The first VNF to be started can be Listener_3_from_1 or listener_1_from_3. If Listener_3_from_1 is started first, then Decrypt_3_from_1, Encrypt_1_to_3 and Writer_1_to_3 must be started in that order. The same must then be done for listener_1_from_3. In addition to the ordering constraint there is a timing constraint: listening sockets must be up and running before writers can connect. This means for example that Decrypt_3_from_1 can only be started once Listener_3_from_1 is up and running. The VNF start up scripts that are generated from the service manifest take into account the ordering and timing constraints.

IV. Discussion and Related Work

A. Discussion

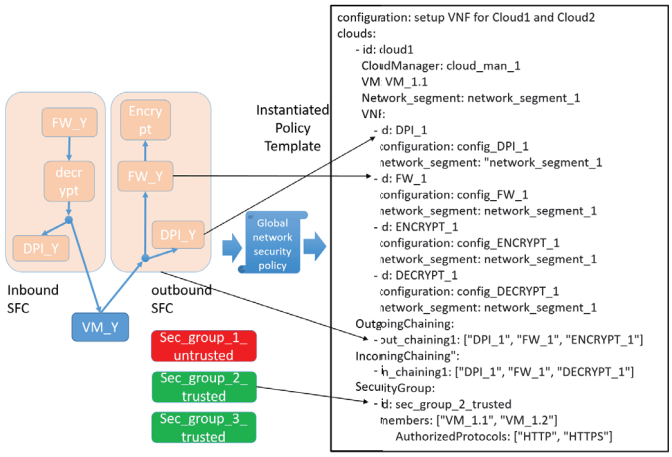


Fig. 5. SFC and security groups

The encryption VNF case study presented above illustrates our approach but remains simple. In reality multiple VNF must be combined using SFC in each cloud. Figure 5 shows a more complex example of security policy. The left part of the figure shows a security policy template that combines a firewall, decryption and DPI VNF for incoming traffic, and a DPI, Firewall and encryption VNF for outbound traffic. This security policy template can then be instantiated into a service manifest on the right side of the figure. The right side of the figure shows the service manifest where the configurations of the DPI and firewall are defined, and where the outbound and inbound chaining is defined. It states that the encryption VNF should be placed at the end of the outgoing service chain and specifies which key to use based on the destination address. Conversely it specifies that the decryption VNF should be placed after the firewall of the incoming service chain and specifies the private key to be used. The lower part of the figure shows how the service manifest can be used to define security groups for the federated network. Security groups are sets of IP filter rules that are applied to a VM instance's networking. The traditional use of security groups is to ensure security between tenants in the cloud. However, it is also possible to use this mechanism to secure traffic in the tenant's network between clouds. Security groups applied to VM's ports by means of OpenFlow filtering rules can be used to protect VM from unauthorized access from untrusted clouds. The default security group denies all incoming traffic.

Figure 6 how access to virtual machines can be controlled with security groups. The left part of the figure shows that VM_3 from the public cloud can only communicate with

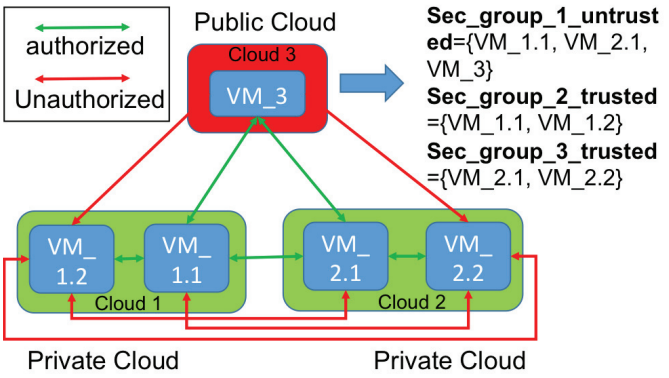


Fig. 6. Security groups example

VM_1.1 in cloud 1 and VM_2.2 in cloud 2. In cloud 1 VM_1.2 can only communicate with VM_1.1. Similarly in cloud 2 VM_2.2 can only communicate with VM_2.1. All other communications are forbidden are indicated by red lines. The right part of the figure shows the resulting definition of security groups that define permitted communications.

The encryption case study is implemented using YAML HEAT templates, and Python code is generated to configure and start up the VNF. The more complex case study presented in this section will be implemented in the future. The case study will be integrated in OpenStack according to the general architecture defined in [2] and the OpenStack specific architecture defined in [3]. The network federation manager will be integrated into OpenStack to manage the network federations. It will configure the different clouds by generating cloud specific HEAT templates from the general service manifest. The OpenStack service function chaining API will be used to configure and deploy the VNF.

B. Related Work

There are standardization efforts on SFC. IETF SFC working group [4] developed a framework for SFC operation and administration. The Open Network Foundation proposed a Service function chaining (SFC) architecture [5]. The architecture includes an SFC network controller responsible for setting up the service chaining; Classifier responsible for classifying traffic flows and Service function forwarder, which is responsible for forwarding packets to the service functions defined in the policy table. The proposed general architecture can be applied to any network. However, in order to implement this architecture in federated heterogeneous networks, there is a need to coordinate SFC network controllers and classifiers, so they will speak the same language and deploy and control NFVs in an efficient way.

In addition to a general architecture definition, there are more detailed SFC issues addressed in the literature. For example, [6] and [7] address VNF service-chain placement issues in different cloud network scenarios. However, they do not address federated network scenario with its specific issues of coordinating SFCs across heterogeneous networks empowered by different virtualization technologies. The authors of [8] propose a design of a resource orchestrator that steers the control of SFCs in a scalable way. Their orchestrator map network functions of a requested SFC to infrastructure network and compute resources. While scalability is an important issue in federated cloud, there are additional issues that have to be addressed, such as securing heterogeneous environments with different levels of trust between them.

V. Conclusion

Federating cloud virtual networks across different clouds provides much needed mechanisms to create customized and isolated networks for applications. Federated cloud networks can be integrated into existing cloud federation mechanisms such as cloud bursting, cloud brokering or cloud aggregation. This paper described an approach for customising federated cloud network security with VNF and SFC. It was argued that the deployment and configuration of security VNF and their chaining across the different federated cloud networks should be automated. The approach was illustrated with a motivating case study of encryption and decryption VNF. Future work involves developing a network federation manager and experimenting with a service chaining API in OpenStack.

Acknowledgment

This work has been supported by the BEACON project, grant agreement number 644048, funded by the European Unions Horizon 2020 Programme under topic ICT-07-2014.

References

[1] A. Levin, K. Barabash, Y. Ben-Itzhak, S. Guenender, and L. Schour, "Networking architecture for seamless cloud interoperability," in *8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015*, pp. 1021–1024, 2015.

[2] R. Moreno-Vozmediano, E. Huedo, I. M. Llorente, R. S. Montero, P. Massonet, M. Villari, G. Merlino, A. Celesti, A. Levin, L. Schour, C. Vázquez, J. Melis, S. Spahr, and D. Whigham, "BEACON: A cloud network federation framework," in *Advances in Service-Oriented and Cloud Computing - Workshops of ESOCC 2015, Taormina, Italy, September 15-17, 2015, Revised Selected Papers*, pp. 325–337, 2015.

[3] A. Celesti, A. Levin, P. Massonet, L. Schour, and M. Villari, "Federated networking services in multiple openstack clouds," in *Advances in Service-Oriented and Cloud Computing - Workshops of ESOCC 2015, Taormina, Italy, September 15-17, 2015, Revised Selected Papers*, pp. 338–352, 2015.

[4] S. Aldrin, R. Krishnan, N. A. C. Pignataro, and A. Ghanwani, "Service function chaining operation, administration and maintenance framework," in *IETF RFC*, Feb 2016.

[5] "L4-l7 service function chaining solution architecture," in *ONF TS-027. Version 1.0*, June 2015.

[6] A. Gupta, M. F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service chaining using virtual network functions in network-enabled cloud systems," in *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, pp. 1–3, Dec 2015.

[7] S. Mehraghdam and H. Karl, "Specification of complex structures in distributed service function chaining using a YANG data model," *CoRR*, vol. abs/1503.02442, 2015.

[8] S. Sahhaf, W. Tavernier, J. Czentye, B. Sonkoly, P. Skldstrm, D. Jocha, and J. Garay, "Scalable architecture for service function chain orchestration," in *2015 Fourth European Workshop on Software Defined Networks*, pp. 19–24, Sept 2015.

Neutralizing Interest Flooding Attacks in Named Data Networks using Cryptographic Route Tokens

Aubrey Alston
The MITRE Corporation, Columbia University
Bedford, MA, New York, NY
aalston@mitre.org, ada2145@columbia.edu

Tamer Refaei
The MITRE Corporation
McLean, VA
mrefaei@mitre.org

***Abstract*—Named Data Networking (NDN) is considered to be a viable candidate to replace the host- centric IP model in the next generation of the Internet. Although NDN is known to be resistant to classical DoS and spoofing attacks, vulnerability to NDN-specific attacks nevertheless arises from the use of stateful routing to satisfy requests (Interests). Prime among these attacks, Interest flooding attacks on Named Data Networks induce denial of service by maliciously occupying memory kept to maintain state of outstanding Interests. While previous work has focused on developing reactive Interest flooding detection and mitigation strategies, we contribute with this an in-packet cryptographic mechanism called the *route token* which proactively provides named data networks a quantifiable degree of security against Interest flooding without relying on a stateful forwarding plane.**

I. Introduction

Named Data Networking (NDN) is an architecture currently being explored as a basis for the next generation of the Internet [1]. NDN is designed to retrieve resources from within the network as identified by their names rather than a host address as a response to the observation that most traffic is centered around dissemination of the same data to many consumers [2] [1]. This paradigm shift alone solves long- standing challenges facing the IP Internet architecture including but not limited to address space exhaustion, NAT traversal, address management, address spoofing detection, and various content distribution and control problems [1].

An outlying question preventing widespread use of NDN concerns the vulnerability of NDN to denial of service (DoS) attacks [3]; NDN networks are especially vulnerable to Interest flooding attacks, a specific form of DoS attack where an attacker exhausts network resources [4] by issuing a large number of requests.

This work contributes and analyzes the first mechanism of its kind in NDN to address this problem: a structural, protected data segment called a *route token* contained in packets propagated through the network. *Route tokens* utilize cryptographic primatives to eliminate the ability of an attacker to consume network memory resources, in doing so removing the upper limit on outstanding requests in the network. Our presentation is organized as follows: we first provide an overview of NDN routing and forwarding, discuss vulnerability introduced by stateful routing, prove the security of our solution, and conclude with a discussion of benefits, limitations, and future work.

978-1-5090-3216-7/16/$31.00

II. Background

A. NDN Routing and Forwarding Overview

The NDN architecture is decentralized, with communication and routing being driven under the semantics of *consumers* and *producers*: consumer processes originate Interest packets, specifying requested named data items, which are propagated towards a producer process, which then provides data which the network propagates back to the consumer along the same path [1].

More specifically, after a consumer constructs an Interest packet [5] which specifies the name of a named data resource, the packet is forwarded towards the producer as follows [1]: The consumer consults a local data structure called a *Forwarding Information Base (FIB)* to determine an outgoing interface based upon the resource name. All intermediary forwarding routers then eventually receive the Interest and forward it towards the producer by (1) consulting a local data structure called a *Content Store (CS)* to serve from a local cache if possible, (2) modifying *Pending Interest Table* to record the Interest and the interface on which it was received, and then (3) forwarding the Interest to the next hop according to the local FIB.

Once a producer or router able to respond from its cache receives the Interest packet, a data packet is constructed and forwarded along the original path of the Interest to the consumer using the PITs of each intermediary router [1].

B. Overview of Interest Flooding Vulnerability

The previous discussion of routing of Interest and data packets in NDN networks showcases per-request state kept by intermediary routers in the PIT. As the size of the PIT grows at a rate effectively linear in the number of outstanding requests, NDN networks without protection may be crippled by even the smallest scale of Interest flooding attacks.

The mechanics of an Interest flooding attacks are simple: by generating a large number of Interests per second, an adversary may occupy the PITs of intermediary routers, causing increased latency. Worse, persistent traffic, benign or malicious, may come to completely saturate or overflow the

PITs of network routers, disabling the network from servicing legitimate requests.

Under these circumstances and the relative ease of issuing an attack, the open problem [4] [1] of removing vulnerability to Interest flooding attacks must be addressed before NDN may be relied on as the basis of the Internet.

C. Related Work

Interest flooding attacks have been long-identified as a problem in NDN [3]. All previously available work on the subject approaches the problem of Interest flooding attacks from the perspective of reactive intrusion detection through heuristic means of determining maximum bandwidth or cooperative methods of filtering malicious traffic [4], [6], [7].

This work differs from previous work in that it provides proactive in-network protection while remaining generally applicable to all forms of information-centric networks. In comparison, previous works are largely reactive, making use of NDN-specific qualities. [4].

III. Route Tokens

A. Preliminaries

We define a *route token* as any data structure included within an NDN packet holding information that may be used to route a data packet along a path to a consumer.

1) Notation and Shorthand: In our discussion and analysis of route tokens, we make use of a limited set of well-defined notation. $A_B^{(P)}$ is shorthand referring to some function or value A_B associated to party P. B may hold the significance of a qualitative label, an identifier, or any other signifier. $F_K(X)$ refers to some function, keyed under K, applied to X. $A||B$ signifies the concatenation of B onto the end of A. $truncate(A, x)$ is a function which evaluates to the first x symbols of A; unless explicitly mentioned, A is interpreted as a binary string.

2) Mechanics of Route Tokens: A route token grows in information as the Interest is propagated towards a producer and is utilized as the data packet is propagated towards a consumer. In general terms, we illustrate the use and function of a route token:

(a) A consumer C constructs an Interest packet *I* and appends to it an empty route token X_0.
(b) C forwards (I, X_0) to the next-hop router on the path to the producer P.
(c) Router i receives (I, X_{i-1}) and obtains X_i by computing some function $add(X_{i-1}, information)$ to add the information required to route a corresponding data packet back towards the consumer.
(d) Router i forwards (I, X_i) to the next-hop router on the path to the producer P.
(e) The Interest eventually reaches P. P constructs a data packet *D* and appends to it X_{N-1} received.
(f) P forwards (D, X_{N-1}) to router $N-1$.
(g) Router i uses the information contained in X_i to determine the interface on which to forward $(D, X'_{i-1} = remove(X_i, information))$.
(h) The previous step is repeated (forwarding $(D, X'_{i-1}$ to router $i-1$ from router i), determining the next hop from X until C receives D.

We note that route tokens eliminate the necessity to rely on the PIT but may simply be used only after the PIT has reached a threshold size.

3) Information Contained by Route Tokens: A route token which contains all information required to route a data packet from a producer to consumer *independently suffices* for the fulfillment of an Interest; a route token must independently suffice to remove PIT state at any one hop.

The information required by each individual router to route data to a consumer the interface on which the corresponding Interest packet was received [1]. By modeling the incoming interfaces along the path as independent random variables, and by making the assumption that there is no consistent distribution among interfaces receiving Interest packets, the lower bound on the size of a *independently sufficient* route token is $b = \sum_{i=1}^{N-1} \lceil log_2(|I(R_i)|) \rceil$ (where N is the number of routers along the path and $I(R_i)$ is the incoming interface at router i) by the Shannon source coding theorem [8].

Using the above bound, we distinguish between *lossy* route tokens and *lossless* route tokens. Lossy route tokens are route tokens which do not adhere the bound *b*, having a necessary loss of information; lossless route tokens are route tokens which adhere to *b*.

4) Security Goals: In addition to the stated goal of preventing malicious memory consumption, we identify two security goals for any route token; *Confidentiality*: route information should only be visible to concerned routers. Formally, an attacker may recover the incoming interface at a router with only negligible probability as a function of the number of bits k required to represent an interface. *Integrity*: Valid route tokens should not allow tampering. More specifically, an attacker may successfully modify one or more hops of the route stored by the route token with only negligible probability as a function of some configurable security parameter s.

B. Symmetric Key-Authenticated Cryptographic Route Tokens

We propose a form of low-overhead lossless route tokens which rely only on symmetric-key operations to perform authentication. This section is organized as follows: (1) an enumeration of the setup and requirements for the application of these tokens, (2) a description of the application of these tokens, and (3) an analysis of security and overhead.

1) Setup and Requirements: In order to make use of these symmetric-key-authenticated route tokens, each network member A must first obtain or choose (1) a number of bits to dedicate to authenticity security, s_A, (2) a keyed pseudorandom secret function S_K, (3) a keyed collision-resistant hash function H_K, (4) a symmetric key used to self-assure confidentiality, $K_{Conf}^{(A)}$, valid under S, (5) a symmetric key used to self-assure token authenticity, $K_{Auth}^{(A)}$, valid under H, (6) a function $g(x_1, ...)$ which may be used to normalize and combine bit strings x_i, perhaps a collision-resistant hash of

hashes, and (7) a window of validity w for generated route tokens.

2) Use: We now describe how symmetric-key-authenticated route tokens are used in the context of the previously discussed mechanics of route tokens.

When a consumer C constructs an Interest packet, the consumer constructs the empty route token as $X_0 = (nonce, a_0 = \emptyset, T_0 = \emptyset)$.

When a router R_i, configured to use n interfaces, receives an Interest packet and token (I, X_{i-1}) on an interface identified by f, R_i adds information by computing an *authenticity component* a_i with respect to the current time t measured to the nearest unit of w and the name of the interest $name(I)$,

$$a_i = a_{i-1} || \; H_{K_{Auth}^{(R_i)}}(g(t,\, name(I),\, f,\, a_{i-1},\, nonce)) \bmod 2^{s_{R_i}} \tag{1}$$

generates a mask used to ensure confidentiality of f, $mask = S_{K_{Conf}^{(R_i)}}(t,\, name(I),\, a_{i-1},\, nonce) \bmod 2^{\lceil log_2(n) \rceil}$, constructs the next-hop route token $X_i = (nonce, a_i, T_i = T_{i-1} || mask \oplus f)$ and forwards (I, X_i) as usual.

When a router R_i then later receives a data packet and token (D, X_{i+1}), R_i first obtains the apparent interface f' contained by the token, on which to forward the packet by recalculating the mask and using it in conjunction with the last $\lceil log_2(n) \rceil$ bits of t_{i+1}. R_i then verifies the authenticity of f' by verifying whether or not $H_{K_{Auth}^{(R_i)}}(g(t,\, name(I),\, f',\, nonce)) \bmod 2^{s_{R_i}}$ is equivalent to the last s_A bits of a_{i+1}. If verification fails, this f' is not accepted, and the router drops the packet and does nothing; otherwise, the router computes

$$\begin{aligned} X_i = (nonce, a_i = truncate(a_{i+1}, |a_{i+1}| - s_{R_i}), \\ T_i = truncate(T_{i+1}, |T_{i+1}| - \lceil log_2(n) \rceil)) \end{aligned} \tag{2}$$

and forwards (D, X_i) on the interface f'.

3) Security Analysis: Confidentiality. We first demonstrate that these lossless route tokens satisfy our formal definition of confidentiality.

Consider a computationally capable attacker who receives a symmetric key- authenticated route token X_i and attempts to recover the incoming interface added at router j along the path encoded within the token. Additionally grant the attacker the ability to determine which sequence of k bits corresponds to the masked incoming interface at router j.

If such an attacker who does not already know f may reliably determine f given $mask \oplus f$, where $mask = S_{K_{Conf}^{(R_i)}}(t,\, name(I),\, a_{i-1},\, nonce) \bmod 2^k$, she is thus able to predict $mask$, contradicting the choice of S as pseudorandom. We thus have that the probability of an attacker determining f, $P[A]$ is negligible in k so long as S guarantees at least k bits of pseudorandomness. Under the same contingency, symmetric key-authenticated route tokens guarantee confidentiality as stated.

Integrity. We now demonstrate that these lossless route tokens also satisfy our formal goal of integrity.

Consider a computationally capable attacker who receives a symmetric key- authenticated route token X_i and attempts to modify the incoming interface at router j encoded. Additionally grant the attacker the capability of modifying the route token to make the apparent interface f' with certain probability and the capability of determining which s_{R_j} bits of a_i correspond to the authenticity component placed by router j.

If such an attacker attempts to independently produce $H_{K_{Auth}^{(R_j)}}(g(t,\, name(I),\, f',\, a_{j-1},\, nonce)) \bmod 2^{s_{R_j}}$ for f', such an attacker may only do so with probability $P[A]$ less than or equal to that with which she may produce a collision in H in s_{R_j} bits. Thus, so long as H provides at least s_{Rj} bits of collision resistance, $P[A] \leq \frac{1}{2^{s_{Rj}}}$

Note also, that the events of successful route modification are independent across all hops. Thus, if we assume a constant s bits of authenticity security for each router, the security of the entire path is cooperative among routers: $P[A_n] \leq \prod_{n=1}^{m} \frac{1}{2^s} \leq \frac{1}{2^{ms}}$.

4) Overhead Analysis: We now analyze these lossless signature-authenticated route tokens with respect to transmission overhead, computation overhead, and storage overhead.

Transmission Overhead. Interest and data packets containing these route tokens take on a small to moderate amount of additional transmission cost due to the inclusion of the token.

At each hop from the consumer to the producer, the Interest packet takes on $s_{R_i} + p$ bits of additional space; note that the maximum degree of additional space is seen only at the last hop from consumer to producer: the route token grows as it moves towards the producer and shrinks as it passes back to the consumer.

Computation Overhead. Routers making use of these route tokens take on low to moderate computation overhead. Each time a router encounters a route token, it must perform a small number of hash operations, requiring on the order of one-thousand additional cycles total [9].

Storage Overhead. Because each router needs to store only a fixed amount of data to service all requests, the associated storage overhead is negligible when compared to PIT-dependent forwarding.

IV. Evaluation

In order to evaluate the performance of route tokens, we have implemented and measured their cost and overhead through discrete-event simulation.

A. Method

Our evaluation was constrained to a series of simulations performed on an Internet-like network topology consisting of thirty consumers, twelve attackers, ten backbone routers, and seven gateways.

All links were simulated full duplex at 300 Mbps capacity; consumers produce a constant rate of traffic of 0.5 Mbps consisting of 100-byte Interests; producers generate 500-byte

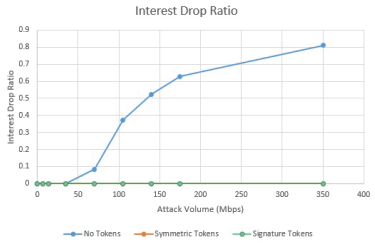


Fig. 1. Interest Drop Ratio

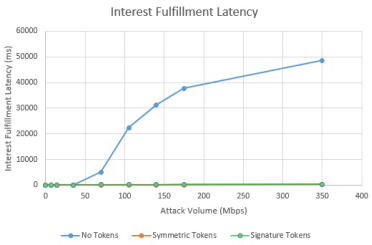


Fig. 2. Interest Fulfillment Latency

data packets. We simulated attacks on the following strategies: (1) PITs having a maximum size of ten-million records (standard NDN forwarding), (2) symmetric-key authenticated tokens with per-hop security of 15 bits, and (3) a form of lossy route tokens called asymmetric-key authenticated tokens (not outlined in this paper) which make use of anonymous bloom filters signed using standard 512-bit RSA.

The target metrics in these simulations were the *interest drop ratio*, the fraction of legitimate interests which are dropped by the network, and the *Interest fulfillment latency*, the time that passes between the generation of an Interest and the arrival of the corresponding data.

B. Results

In this section, we present and discuss the performance of route tokens as measured through simulation.

Figure 1 shows a plot of the portion of legitimate Interests dropped as a function of the configured attack volume. This plot shows that legitimate Interests are never dropped when route tokens are used.

Figure 2 shows a plot of Interest fulfillment latencies as a function of the configured attack volume. For attack rates above 50 Mbps, the latency experienced when relying on standard NDN routing vastly exceeds that for either form of route token.

The result in 3 implies that the only remaining problem is congestion control, since we see significant increases in latency only at link capacity.

C. Limitations

The scope of our solution is limited solely to vulnerability to malicious memory consumption; as such, our solution does not address the more elementary issues represented by denial of service: congestion control. The route token proposed also has a slight limitation in that it imposes a necessary but small transmission and computation overhead.

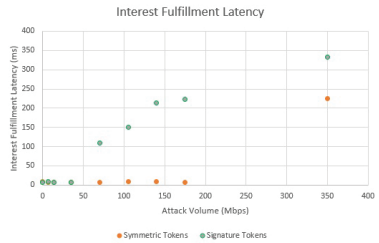


Fig. 3. Interest Fulfillment Latency

V. Future Work

The next step in continuing this work could include exploring means to reduce the overhead associated with route tokens, exploring new means of achieving low-overhead lossy route tokens, or evaluating the benefits of route tokens when used in tandem with conventional network intrusion detection.

VI. Conclusion

NDN is a protocol being considered as a basis for the next generation of the Internet. While NDN's use of stateful forwarding offers powerful capabilities, it also exposes vulnerability to Interest flooding, a DoS attack unique to NDN. While previous work has focused on developing reactive Interest flooding detection and mitigation strategies, we have outlined and contributed an in- packet data structure called the route token which invalidates the risk of memory saturation as a result of Interest flooding. We have analyzed and evaluated this mechanism through simulation to verify that it offers a verifiable degree of security and visibly neutralizes the effect of Interest flooding attacks.

References

[1] L. Zhang, K. Claffy, P. Crowley, C. Papadopoulos, L.Wang, and B.Zhang, "Named data networking," *ACM SIGCOMM Computer Communicaiton Review*, vol. 44, pp. 66–73, Jul. 2014.

[2] V. Jacobson, "A new way to look at networking," YouTube, Google, 2006. [Online]. Available: https://www.youtube.com/watch?v=oCZMoY3q2uM

[3] M. Wahlisch, T. C. Schmidt, and M. Vahlenkamp, "Backscatter from the data plane - threats to stability and security in information-centric network infrastructure," *Computer Networks*, vol. 57, no. 16, pp. 3192–3206, Nov. 2013.

[4] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang, "Interest flooding attack and countermeasures in named data networking," in *IFIP Networking Conference*, Brooklyn, NY, May 2013, pp. 1–9.

[5] (2016, Jul.) Named data networking: Motivation and details. Web. Named Data Networking (NDN). [Online]. Available: https://named-data.net/project/archoverview/

[6] K. Wang, H. Zhou, Y. Qin, and H. Zhang, "Cooperative-filter: Countering interest flooding attacks in named data networking," *Soft Computing*, vol. 18, no. 9, pp. 1803–1813, Apr. 2014.

[7] K. Ding, Y. Liu, H. H. Cho, H. C. Chao, and T. K. Shih, "Cooperative detection and protection for interest flooding attacks in named data networking," *International Journal of Communication Systems*, Oct. 2014.

[8] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

[9] "Speed comparison of popular crypto algorithms," Web, 2016. [Online]. Available: https://www.cryptopp.com/benchmarks.html

Evaluation of Distributed Denial of Service Threat in the Internet of Things

Luis Alberto B. Pacheco, João J. C. Gondim, Priscila A. Solis Barreto and Eduardo Alchieri
Computer Science Department
Universidade de Brasília
Brasília, Brazil
Email: luisbelem@aluno.unb.br,{pris,gondim,alchieri}@unb.br

***Abstract*—There is a concern about possible threats deriving from the widespread adoption of IoT (Internet of Things). The number of devices connected to the Internet is going to increase dramatically, potentiating their security risks. A Distributed Denial of Service (DDoS) attack is a good candidate to explore IoT security vulnerabilities, because of the enormous number of new devices connected to the Internet there is also an increase in the number of possible compromised devices. This study aims to analyze the efficiency of a DDoS attack in a typical IoT environment, by using simulations that, in the best of our knowledge, have not been conducted yet.**

I. Introduction

The new wireless communication technologies that were consolidated in the last decade have been increasing the connection of a great variety of devices. Smart environment, extended to smart cities, defines automated urban environments using interconnected sensor devices. This technology enables a life style with greater quality and safety, besides the optimization of resources such as energy, water, transport, etc.

The union of the presented concepts is called Internet of Things (IoT), it needs to use trustworthy information sources, represented by several distinct sensors incorporated in an ubiquitous manner in the environments.

Automation applications have been an interesting start point in the IoT concept development. Such applications enable precise control over desired environments (houses, offices, etc.), regardless of the distance to the user. This includes the automation of door locks, doors, lamps, moving sensors, thermometers, cameras and other similar devices. The obvious implications of privacy and security of users allow to infer that those applications are very attractive for malicious activities.

As can be historically observed, new applications and softwares can bring huge security treats. New systems and protocols, usually developed in a rapid manner, normally does not predict those treats. An IoT environment can have even more treats due to the large scale of deployment and the integration with the physical world. For instance, security systems in houses and neighborhoods can be disabled, and critical applications in hospitals can be disrupted.

The simplicity of a Denial of Service (DoS) attack [1] makes it an increasing security treat. The characteristics of devices of an IoT environment makes it vulnerable to DoS attacks. Those devices are not only potential targets but can also act as relays, generating illegitimate traffic to disrupt other service. The growth of IoT environments is supposed to propitiate and increase its use in botnets used in Distributed Denial of Service (DDoS) [2] attacks.

978-1-5090-3216-7/16/$31.00

Considering the security treats in IoT, the main contribution of this study is the experimental evaluation (through a simulated environment) of the impact of a DDoS attack in a typical IoT environment. In particular, a network stack with the Constrained Application Protocol (COAP) [3] was used to simulated a reflexion attack.

The remaining of this text is organized as follows. Section 2 details the IEEE 802.15.4 standard and the other protocols utilized in the simulated IoT environment. Section 3 presents related studies. Section 4 describes the simulation scenario and analyses the results obtained. Finally, Section 5 concludes this study and presents future works.

II. Internet of Things

This section describes the protocol stack utilized in the IoT environment simulated in this work, including the IEEE 802.15.4 standard and the COAP.

A. Protocols and IoT stack

The IoT scale factor, the detection restriction of platforms and the need for solutions that work in cooperation with existing security solutions of the Internet, are guidelines that have been used by the working groups created to define standards, such as the Institute of Electrical and Electronics Engineers (IEEE) and Internet Engineering Task Force (IETF). Those guidelines drive the working groups in the conception of new communication and security protocols that will be essential for future IoT applications.

Such solutions are being designed in alignment with the restrictions and characteristics of sensor devices of low energy, low wireless data rate communication and low processing power. Although such characteristics influenced past projects that make use of WSNs (wireless sensor networks) isolated from the Internet, the new standardized solutions are conceived to ensure the interoperability with the existing Internet standards. It also enables the devices to communicate with external entities in the Internet, in the context of IoT [4].

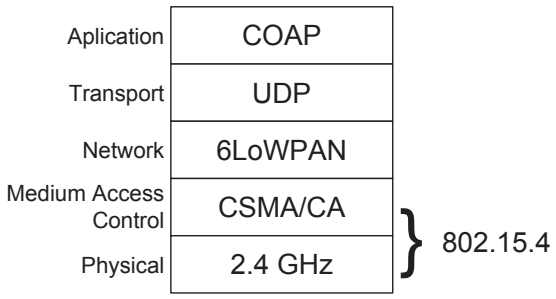


Fig. 1. IoT protocol stack

The protocol stack shown in Figure 1 was conceived with communication standards provided (released or in development) by the cited working groups. The UDP protocol is used in the transport layer, since it has a small overhead when compared with the TCP protocol. The TCP protocol offers several features that incur an higher number f message exchange and size of packet header. The CoAP, 6LoWPAN and 802.15.4 standards are described in the next sections.

B. IEEE 802.15.4

The IEEE 802.15.4 [5] standard specifies the physical (PHY) and medium access control (MAC) layers (layers 1 and 2 of the OSI model [6], respectively) for Low Rate Wireless Personal Area Networks (LRWPANs).

The MAC layer has to provide communication between two devices, which is achieved by a contention mechanism. The IEEE 802.15.4 standard implements the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) scheme together with a deterministic mechanism.

The PHY layer transmits the frame that came from the MAC layer through the medium. In order to handle different country regulations, the IEEE 802.15.4 standard defines several working frequency bands. A compliant device must support at least one of them.

The standard has security features, providing confidentiality, authenticity and replication protection to data transmission. The several security levels enables the devices to choose the cryptography with an authentication code of varying size. When this scheme is used the authentication code validates the header (which is sent without encryption) and the payload, that can be encrypted.

C. 6LoWPAN

Specified by the RFC 4944 [7], IPv6 Over Low Power Wireless Personal Area Network (6LoWPAN) is a standard that compresses IPv6 packets in 802.15.4 frames. Its advantages includes easy connectivity with IP devices, enabling the opportunity of using the existing network infrastructure and an API widely used. 6LoWPAN has its features presented in several RFCs, such as the RFC 4944, that defines the frame format that adapts the IPv6 frame to 802.15.4 frames (that can have only up to 127 bytes). The RFC 4944 was updated by the RFC 6282 [8], enabling multicast transmissions and IPv6 extensions. The RFC 6568 [9] presents deployment scenarios. The most recent updated is contained in the RFC 6775 [10].

D. CoAP

The RFC 7252 [3], released in June of 2014, specifies the Constrained Application Protocol (CoAP), a message exchange application layer protocol for low energy, low processing power devices and constrained networks. It was designed for machine-to-machine (M2M) applications, a paradigm used in residential automation. It defines 4 message types: Confirmable, Non-confirmable, Acknowledgment and Reset.

Messages of the Confirmable (CON) type must be acknowledged. On receiving a CON message, a node sends an Acknowledgment or a Reset message. The Non-confirmable (NON) messages do not require an acknowledgment, which is useful when an eventual message loss does not disrupt the application operation. Reset messages indicates that a CON or NON message was received but could not be properly processed. The CoAP protocol was designed to interact with the Hyper Text Transfer Protocol (HTTP), simplifying the integration with the Internet. It is based on compact message exchange over the UDP transport layer protocol. Messages have a header of 4 bytes followed by a token with up to 8 bytes.

The CoAP protocol does not include security features, instead it uses the Datagram Transport Layer Security (DTLS) [11]. DTLS is analogous to the TLS, but for the UDP protocol. It provides integrity, authenticity, confidentiality, key management and several cryptography algorithms. However DTLS was not designed for IoT, for instance it does not support multicast, which is an advantage of the CoAP in comparison with others application layer protocols.

III. Related works

Although the 6LoWPAN and CoAP protocols bring advancements in reducing the difference between Internet and IoT protocols, those protocols still does not achieve specifications identical to the ones of Internet, mainly for performance reasons. In this regard, it is expected that small differences will remain existing among the protocols of these technologies. Although those differences could be mitigated by translators at the network gateway, they continue to be an obstacle for implementing security between IoT and Internet devices [8].

Several security requirements must be considered in an IoT environment, in particular the communication among sensor devices. For instance, WSNs can be exposed to Internet originated attacks, such as Denial of Service (DoS). A DoS attack is a malicious attempt to consume bandwidth resource of legitimate users [1]. Such attack, when occur from several compromised nodes, are called Distributed DoS (DDoS).

A reflection attack is a DDoS attack that utilizes intermediary hosts. The attacker floods reflectors with the source IP address set as the victim's address. A reflector is any device that when receives a message sends a response. The difference between the request and the response sizes is explored: as

the reflection factor increase, more effective is the attack, amplifying the attacker's generated traffic [2].

There are some studies that address DoS attacks in IoT. Perakovic et al. [12] analyses the availability of services by flood attacks. Tendencies of those attacks where analyzed with data from 2013 to 2015. The analysis shows that it is increasing the number of attacks in the IP and transport layer, in comparison with attacks in the application layer. Within this context, the emerging number of IoT devices could be potential reflectors for illegitimate traffic generation. The study shows that it is increasing the rate of attacks through the Simple Service Discovery Protocol (SSDP), which is utilized in many IoT devices.

Raymond et al.[13] discuss several defense schemes against DDoS attacks in a WSN. This type of attack and its impact in the Quality of Service (QoS) of WSNs based on IP is conducted by [14]. Yu et al. [15] present a review of the main security treats in IoT, displaying a possible roadmap to deal with them. Elkhodr, Shahrestani and Cheung [16] reviewed some of the main concerns to the wide adoption of IoT, such as interoperability, management, security and privacy. Cvitić, Vujić and Husnjak [17] take into account the layers of an IoT stack and discuss the treats of each one of them. Other attacks and defense strategies are discussed in [18] and [19].

IV. Experimental Evaluation of DDoS Attack in IoT

This section presents an experimental evaluation of the impact of a DDoS attack in the IoT context. The DDoS attack aims to explore a network stack with the CoAP protocol, performing an amplified reflection attack. The next sections present a description of the experimental scenarios, followed by the results and its analysis.

A. Simulation scenarios

The simulation scenarios are designed to evaluate the effects of a amplified reflection DDoS attack in a WSN. A common WSN network was deployed, where there is an infiltrated malicious node that sends service discovery request for all sensors with the source address as the victim, which is outside the WSN.

The simulation deploys a WSN in star topology with 5, 25, 50, 75, and 100 nodes. The devices are displayed in grid with 2 meters spacing. The coordinator is the first node of the grid, the attacker is the second and the victim is the last one. The WSN has a throughput of 96 bps for each node, since each one sends a 12 bytes packet every minute.

The simulation was conducted in the NS-3 [20] simulator, an open-source, C++, community driven, network simulator. Each sensor device was deployed with the network protocol stack shown in Figure 1. The IEEE 802.15.4 standard was utilized as the layers 1 (PHY) and 2 (MAC). The MAC operational mode is unslotted CSMA with nodes always listening for the medium when not sending a packet, this mode was chosen in order to maximize the network throughput. The layer 3 (Network) utilizes the 6LoWPAN protocol, which translates IPv6 headers to fit the 802.15.4 packet size limit of 127 bytes. The application layer deploys the CoAP protocol, which utilizes UDP as the transport layer.

B. Results and Analysis

The simulated reflection attack exploits CoAP's resource discovery feature. The attacker sends a resource discovery request packet in multicast (for all network members). The request packet has the "/.well-known/" URI path prefix, which renders a 20 byte packet size. The source IP address of the packet is changed to the victim's one. As the network sensors receive the request packet they must send a response indicating which service they support. In this experiment the devices contain only a temperature sensor, and thus only one service, which is expected from small, low capacity devices. The response message contains 61 bytes, rendering a reflection rate of 3.05 for each device that receives the request packet.

The attacker's data generation rate was varied to find the most optimal injection rate. Traffic was generated at 0.16 bps, 1.6 kbps, 3.2 kbps, 4.0 kbps, 8.0 kbps and 16.0 kbps.

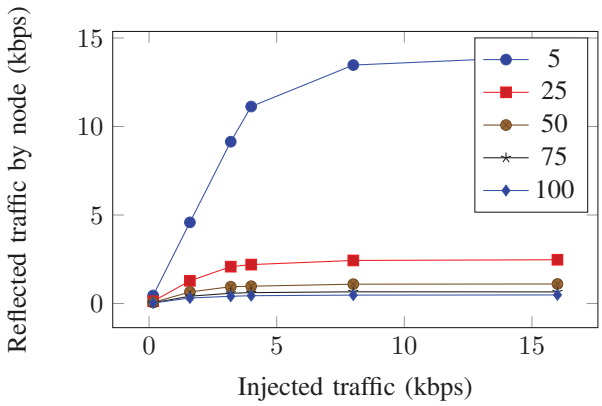


Fig. 2. Average reflection rate

Figures 2, 3 and 4 show the average generated traffic for each node, the total amplification rate and the total generated traffic, respectively. The analysis shows that the reflection occurs with amplification, validating the concerns about using IoT for a DDoS attack. Figure 3 shows that the maximum achieved amplification rate is about 20, for scenarios with more than 5 nodes. The maximum expected amplification rate is 3.05 times the number of devices in the network, rendering 76.25 for 25 nodes, which is much higher than the maximum amplification rate achieved. This difference can be explained by the low data rate characteristic of the network, the IEEE 802.15.4 supports only 250 kbps.

In all scenarios the amplification rate decreases when the injection rate goes above 2 kbps. In the attacker's view, this could be compensated by increasing the number of reflectors in the attack, which is a valid action, since IoT should provide millions of connected devices.

The network saturation, which is characterized by the decreasing in the amplification rate, is caused by the bandwidth

saturation. The maximum injected traffic occurs in the scenario with 5 nodes, decreasing as the number of sensors increase. In order to achieve a higher amplification rate, more WSNs should be used at the same time.

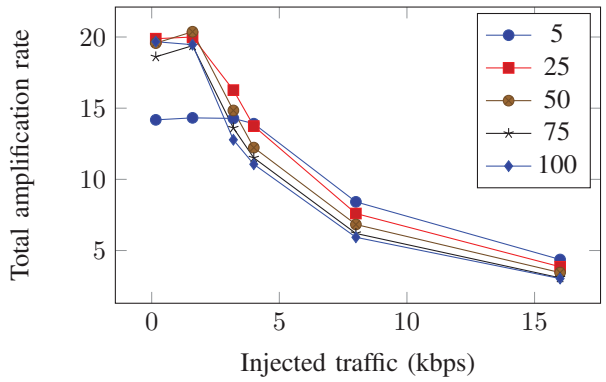


Fig. 3. Total amplification rate

The graphs also show that the WSN network used as reflection starts to receive a DOS attack itself from the attacker, as is shown in the scenario with 100 devices. The involved protocols do not impose a limit in the number of devices deployed in the network. This behavior, although determined by the proposed protocol stack, is expected to persist in other similar IoT scenarios.

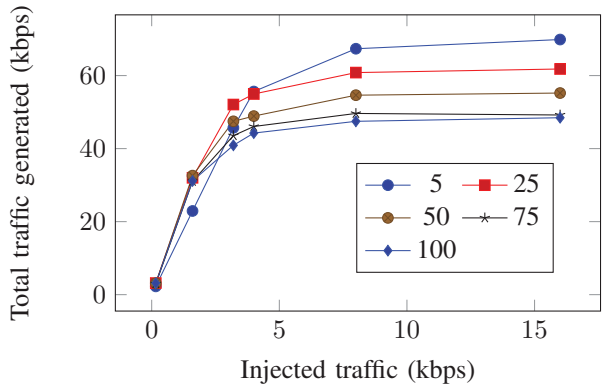


Fig. 4. Total reflection rate

Considering the obtained results, for a successful real DDoS attack using WSNs (as the one deployed in these experiments), it is necessary to use a high number of networks.

Finally, the attack was considered viable, however demands a high number of involved networks. An attacker could have more efficient options to perform a DDoS attack. However the attack's easiness automation can be attractive, even if it may demand fine adjustment and coordination by the attacker.

V. Conclusion

This work studied the viability of using an IoT environment in an amplified reflection DDoS attack. To the best of our knowledge this was the first experimental analysis of this subject that considers the full network stack. A simulation scenario was deployed utilizing the following network protocol stack : IEEE 802.15.4, 6LoWPAN, UDP and CoAP. The simulations analysis shows that a DDoS attack can indeed explore IoT environments. Furthermore, the IoT environment itself can become a target, since its bandwidth and computing resources rapidly depletes.

Although viable, the attack requires a high number of IoT networks to be effective. Some future work is the quantification of the compromise scale. In the attacker's point of view, it is needed to decide if the coordination and fine tune efforts in order to avoid the IoT network saturation are profitable against other attack forms. As future work we also intend to explore other possible risks arising from IoT dissemination.

References

[1] M. McDowell, "Understanding denial-of-service attacks," Department of Homeland Security United States, Tech. Rep., 2009.

[2] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," 2001, aT&T Center for Internet Research at ICSI.

[3] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)(rfc 7252)," 2014.

[4] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the internet of things: a survey of existing protocols and open research issues," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1294–1312, 2015.

[5] IEEE, *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, IEEE Standard for Information Technology, 2011.

[6] "Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model," 1994.

[7] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Rfc 4944," *Transmission of IPv6 packets over IEEE*, vol. 802, no. 4, 2007.

[8] J. Hui and P. Thubert, "Compression format for ipv6 datagrams over ieee 802.15. 4-based networks," 2011.

[9] E. Kim and D. Kaspar, "Design and application spaces for ipv6 over low-power wireless personal area networks (6lowpans)," 2012.

[10] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann, "Neighbor discovery optimization for ipv6 over low-power wireless personal area networks (6lowpans)," Tech. Rep., 2012.

[11] E. Rescorla and N. Modadugu, "Rfc 4347-dtls: datagram transport layer security," 2006.

[12] D. Peraković, M. Periša, and I. Cvitić, "Analysis of the iot impact on volume of ddos attacks," in *PosTel 2015*, 2015.

[13] D. R. Raymond and S. F. Midkiff, "Denial-of-service in wireless sensor networks: Attacks and defenses," *IEEE Pervasive Computing*, vol. 7, no. 1, pp. 74–81, 2008.

[14] A. Le, J. Loo, A. Lasebae, M. Aiash, and Y. Luo, "6lowpan: a study on qos security threats and countermeasures using intrusion detection system approach," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1189–1212, 2012.

[15] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIV. New York, NY, USA: ACM, 2015, pp. 5:1–5:7. [Online]. Available: http://doi.acm.org/10.1145/2834050.2834095

[16] M. Elkhodr, S. Shahrestani, and H. Cheung, "The internet of things: New interoperability, management and security challenges," *arXiv preprint arXiv:1604.04824*, 2016.

[17] I. Cvitić, M. Vujić, and S. Husnjak, "Classification of security risks in the iot environment," in *26TH DAAAM INTERNATIONAL SYMPOSIUM ON INTELLIGENT MANUFACTURING AND AUTOMATION*, 2016.

[18] O. Garcia-Morchon, S. Kumar, R. Struik, S. Keoh, and R. Hummen, "Security considerations in the ip-based internet of things," 2013.

[19] T. Heer, O. Garcia-Morchon, R. Hummen, S. L. Keoh, S. S. Kumar, and K. Wehrle, "Security challenges in the ip-based internet of things," *Wireless Personal Communications*, vol. 61, no. 3, pp. 527–542, 2011.

[20] ns 3 project, "ns-3 network simulator," http://www.nsnam.org/.

A Continuous Enhancement Routing Solution aware of Data Aggregation for Wireless Sensor Networks

Edson Ticona Zegarra, Rafael C. S. Schouery, Flávio K. Miyazawa and Leandro A. Villas
Institute of Computing, University of Campinas, Brazil
Email: edson@lrc.ic.unicamp.br, {rafael, fkm, leandro}@ic.unicamp.br

***Abstract*—Wireless sensor networks consist of hundreds or thousands of nodes with limited energy resources. Due to the high density of nodes in this kind of network, redundant data will be detected by nearby nodes. Since the network lifetime is a key issue in wireless sensor networks, in-network data aggregation can be exploited in order to reduce the number of messages exchanged and consequently reduce the energy consumption. Although there are many data aggregation solutions in wireless sensor networks, most of them leads to low quality routing trees and does not address the load balancing problem, since the same tree is used throughout the network life. To tackle these challenges we propose a Continuous Enhancement Routing Solution named as CER, an approach for computing increasingly better routing trees. CER was extensively compared to three other known solutions: the Shortest Path Tree (SPT), Data Aggregation Aware Routing Protocol (DAARP) and Dynamic Data Aggregation Aware Routing Protocol (DDAARP). The obtained results show that CER outperforms these solutions in all evaluations performed.**

I. Introduction

Wireless Sensor Networks (WSNs) are a special kind of ad-hoc networks that rely on spatially distributed autonomous devices which measure many types of physical quantities like temperature, pressure, pollutants, etc. WSNs are used for defense purposes [1], environmental monitoring, communications, industry [2], agriculture [3], among others that can be critical to save lives and assets.

Devices not only sense but also have to transmit, in a multi-hop fashion, the gathered data to a *sink* node that works as a gateway between the network and the application. The most expensive operation, in terms of energy, is transmission, but nodes are limited by the battery capacity. For that reason, WSN routing algorithms should take into consideration the energy consumption.

A *proactive routing* constantly maintains a routing infrastructure [4], thereby, when an event is detected, the data is immediately dispatched reducing latency times, which may be suitable for high traffic networks. Nonetheless, keeping the routing infrastructure takes more overhead, augmenting the energy consumption on the network nodes. On the other hand, to build a routing infrastructure only when necessary, i.e. when an event happens, is known as *reactive routing*. This kind of routing reduces the amount of transmission by adding some latency and is more appropriate for event-driven networks.

Considering that the devices have a limited processing capacity, we take advantage of what is called in-network data aggregation. The relaying devices, instead of just retransmitting the raw data, can aggregate the data reducing the number of transmitted packets. This scheme often reduces the energy consumption.

With that in mind, we have proposed CER, a hybrid routing algorithm: a proactive part, which gathers some network information such as node position and centralizes into the sink; and a reactive part, which computes the routing tree as the events occur. Relay nodes that have more than one child are able to perform data aggregation. Hence, when multiple packets are received by the relay node, they are aggregated into one single output packet. Moreover, we do not impose restrictions on the computational power of the sink so that it is allowed to perform regular computations. Note that the sink could be connected to a computer with high computing capabilities or to the cloud, so this is not an unrealistic assumption. For computing the routing tree, we use a Biased Random Key Genetic Algorithm (BRKGA) on the sink that returns a low-cost routing tree. This general configuration is depicted in Figure 1.

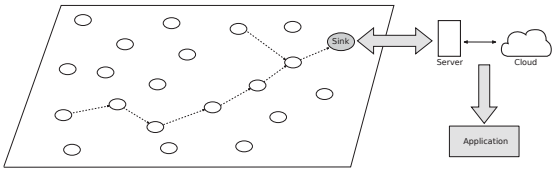


Figure 1. General scenario where the sensor nodes are connected to the sink which computes the routing tree and serves as a gateway with the application.

This paper is organized as follows. On the following section we present current approaches to the problem. On Section III we describe CER; Section IV describes the algorithm we used to compute the routing trees and in Section V we show our results of simulations and compare them with existing solutions. Finally, in Section VI we present our conclusions and present some insights for future work.

II. Related work

In WSN, data aggregation is an approach to aggregate the data sensed by the nodes due to its inherent redundancy, thus reducing the number of transmissions and minimizing the energy consumption. For that reason, we must attempt to find an optimal aggregation tree to extend the network lifetime. The problem of finding an optimal aggregation tree is $\mathcal{NP}$-hard [5], and is equivalent to Steiner Tree problem [6].

978-1-5090-3216-7/16/$31.00

The (rooted) Steiner Tree problem can be defined as follows: given a graph $G = (V, E)$, a set of terminal nodes $S \subseteq V$ and a vertex node $t \in V$, where each edge of E has a non-negative associated cost; must be found a minimum cost tree that connects all the nodes in S to t. The nodes that are part of the solution tree are called *steiner nodes*.

The Shortest Path Tree, *SPT* [6], is a fairly simple approach to the problem in which every node that detects an event reports its data to the sink by using the shortest path. Data aggregation takes place when there are overlapping paths. However, a main disadvantage on SPT is that it is not dynamic and it is highly dependent on the order on which the events occur; the route setup for the first event predisposes the following routes. Moreover, the approach does not consider the end of events to reconstruct the routing tree.

The Information-Fusion Routing Algorithm, *InFRA* [7], tackles the problem by building event based clusters. That is, whenever an event is detected, all the nodes that detected the event are grouped together into a cluster. For each cluster, a node that works as a cluster-head is chosen and will be responsible for aggregate the data in that cluster. Then, cluster-heads send the sensed data to the sink by using the shortest path. InFRA defines the metric *aggregate distance* that is maintained for all the nodes in the network. Each time an event takes places, all the network has to be flooded to update the aggregate distance, allowing it to build dynamic routes. However, constantly maintaining a parameter for the whole network turns the algorithm unscalable. Moreover, InFRA also does not consider the end of events to reconstruct the routing tree.

The Data Aggregation Aware Routing Protocol, *DAARP* [8], is a similar approach to InFRA but the election of cluster-heads is not arbitrary; the nodes chosen for being cluster-heads are the ones closer (in hops) to the sink node. DAARP measures the distance, in hops, to the existing routing tree. When the first event occurs, DAARP sets up the route by using the shortest path to the sink; when the following events occur, DAARP updates the path by choosing a route that leads to nodes closer to the current existing routing tree. To achieve that, each node stores two parameters: its distance, in hops, to the existing routing tree and the next node that has to be followed to reach the mentioned routing tree. A disadvantage on the design is that the routing tree highly depends on the order on which the events appear. DAARP's approach also builds a non-dynamic routing tree; so the first event may build a routing tree that is disadvantageous for the following events. Moreover, DAARP also does not consider the end of events to reconstruct the routing tree.

The Dynamic Data Aggregation Aware Routing Protocol, *DDAARP* [9], in opposition to the previous algorithms, is a centralized approach to the problem. It improves DAARP by creating dynamic routes, as its name suggests, so it is not dependent on the order on which events take place. DDAARP collects the information about the whole network in its configuration phase; then the information is centralized in the sink node. When a new event happens, DDAARP uses a greedy approach to select a route by inserting the smaller number of steiner nodes into the route. However, DDAARP also does not consider the end of events to reconstruct the routing tree.

Our proposal is designed to continuously update and improve the quality of the routing tree. We opted for the usage of BRKGA to generate those trees since it is highly parallelizable and have better performance than simpler approaches like a greedy one. BRKGA has been used in other applications [10], [11] showing promising results in comparison with other approaches for optimization problems. Our proposal guarantees a broader search in order to avoid falling into a local minimum. Thus, the order of events does not predisposes the routing tree for future events. Lastly, in the best of our knowledge, CER is the only solution that considers the end of events to reconstruct the routing tree.

III. CER: Continuous Enhancement Routing

CER aims to find near-optimal routing trees, thus, maximizing data aggregation as long as possible. The solution is composed of four phases: i) configuration phase, described in Subsection III-A, to gather the network configuration and to send it to the sink node; ii) Request/Set Route phase, described in Subsection III-B, handles new events in the network; iii) data transmission, described in Subsection III-C, that is responsible for collecting and sending the data to the sink and finally, iv) Event Ending, described in Subsection III-D, that notifies the sink of an event ending.

A. Configuration phase

In the configuration phase, the sink node broadcasts the *Initial Configuration Message (ICM)*, as shown in Line 1 of Algorithm 1, indicating the *distance*, in hops, and the *next node* that has to be followed in order to reach the sink. When a node receives an ICM, it verifies whether its distance in hops to the sink is bigger than the one contained by the message, which is shown in Line 3 of Algorithm 1. In such a case, the node updates its values with the ones indicated in the message; updates the ICM by increasing the distance in one and setting the next node to itself; and finally, broadcasts the updated ICM, as shown in Lines 4-8 of Algorithm 1. Otherwise, the node just drops the packet. Note that at initialization, all nodes are initialized with distance value to the sink equal to 10000, meaning that they have not been yet configured.

Algorithm 1 Initialization

```
1: Sink node broadcasts ICM
2: for u ∈ V do
3:     if HopsToSink(u) > HopsToSink(ICM) then
4:         HopsToSink(u) ← HopsToSink(ICM)
5:         NextHop(u) ← SenderID(ICM)
6:         HopsToSink(ICM) ← HopsToSink(ICM) + 1
7:         SenderID(ICM) ← ID(u)
8:         Node u Broadcasts ICM
9:     end if
10: end for
```

We define a *border node* as a node that has no neighbors whose distance in hops to the sink is greater than his own.

When all the nodes have received an ICM, the border nodes start broadcasting a *Border Message (BM)* which is a message that carries the adjacency list of the node that generated the message. The management of the BMs is performed in each node by storing the list of neighbors and the list of neighbors that have greater distance to the sink. So, when a BM is received, the receiving node verifies if the BM is coming from a node that is in list of neighbors that have greater distance to the sink; Line 3 of Algorithm 2 shows that validation. In such a case, the receiving node stores the ID of the BM's sender (Line 4 of Algorithm 2) and verifies if all it has received a BM from all its neighbors with greater distance to the sink, in order to broadcast the BM (Line 5 and 6 of Algorithm 2); otherwise, it will wait for the remaining BMs to arrive.

Once the sink receives all the BMs from its neighbors, the sink is ready to accept requests for routing.

Algorithm 2 Border Feedback

```
1: Border nodes broadcast BM
2: for u ∈ V do
3:     if NeighborHopBigger(u) ∈ SenderID(BM) then
4:         RxNeighborHopBigger(u).add(SenderID(BM))
5:         if RxNeighborHopBigger(u) == NeighborHopBigger(u) then
6:             Broadcast BM
7:         end if
8:     end if
9: end for
```

B. Request/Set Route Phase

When an event occurs, the node that become aware of the event sends a *Request Route Message (RRM)* to the sink by using the next node parameter that was setup during the configuration phase. After a small delay time, the node starts sending the sensed data, so we give a time to the sink to construct a routing tree. When the sink receives the RRM, it computes a route and sends a *Set Route Message (SRM)* to the nodes that are part of the route so that they update their next node parameter.

The SRM includes the *tree ID* parameter to handle multiple events, as explained in Section III-C. When a node receives an SRM, it validates if the tree ID of the SRM is greater than its tree ID, as shown in Line 2 of Algorithm 3, so the parameters of the node are updated (Lines 3-5 of Algorithm 3) and the SRM is transmitted to the following node in the routing tree (Line 6 of Algorithm 3).

Algorithm 3 SetRoute

```
1: for u ∈ R do                                  ▷ R is the set of nodes in the tree
2:     if TreeID(SRM) > TreeID(u) then
3:         HopsToSink(u) ← HopsToSink(SRM)
4:         NextHop(u) ← SenderID(SRM)
5:         TreeID(u) ← TreeID(SRM)
6:         u.send(SRM)                           ▷ to the next node in the tree
7:     end if
8: end for
```

C. Data transmission

The sink is in charge to compute improved routing trees, so when the sink finds a better solution than the current one, the sink sends an SRM to the network.

Meanwhile, more than one event can be detected so it may be possible that some RRMs are received at the sink while it is still processing the previous request. When an RRM arrives to the sink, the computing process at the sink is reset including the new node that requested the route.

In addition, it is also possible that an SRM travelling in the network overrides a configuration on a node that has already been configured with a better route. To handle that occurrence, nodes store the routing tree ID and the SRM has the new tree ID it is aiming to set, then a node will only overwrite its routing table if the SRM's tree ID is greater than the one recorded on the node, which can be seen in line 2 of Algorithm 3.

These three characteristics allow the routes to be changed according to the needs of the network, always improving the routing tree; hence, the routing trees generated by CER are dynamic. Furthermore, the data collection is independent of the routing configuration, so the relay nodes always retransmit the data packets using the last configured routing tree.

The network shown in Figure 2 illustrates how our proposal works. Given a routing tree that can be improved, as shown in Figure 2(a) the sink will keep computing routing trees with better cost. Once found, it will setup the network with the enhanced tree, as shown in Figure 2(b). Even more, a routing tree with size 7, as shown in Figure 2(c) is found so the network is reconfigured again.

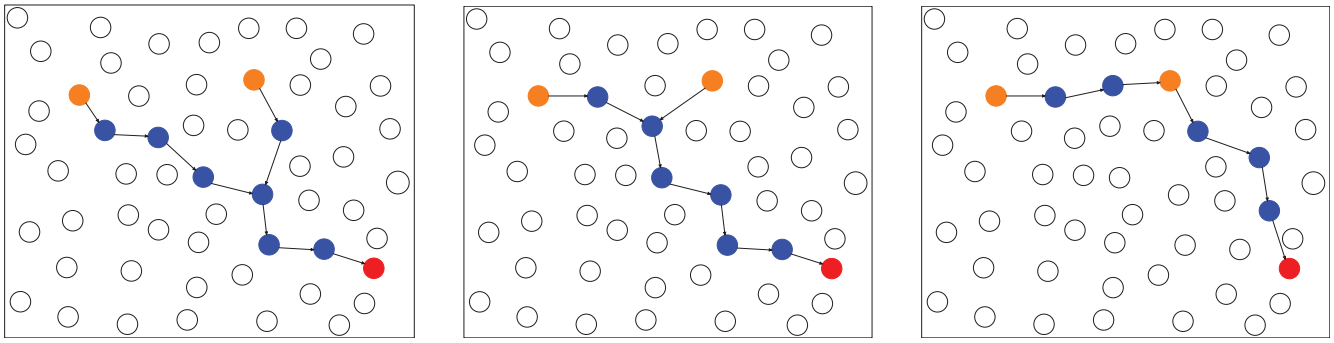
D. Event ending

When an event ends, the routing infrastructure no longer needs to consider the nodes that were reporting such an event. In fact, maintaining that same routing tree affects the overall performance of the network. For that reason, the node that was reporting the data from the culminated event send *Remove Message (RM)* to the sink using the existing routing tree. Upon receiving the RM, the sink resets the algorithm by taking off the node that sent message. After computing a routing tree, the sink sends an SRM to the corresponding nodes.

IV. Biased Random-Key Genetic Algorithm

A Genetic Algorithm (GA) is a metaheuristic, inspired on the evolution mechanism, used for optimization problems. A *feasible solution* is a solution that complies all the restrictions of an optimization problem. A *chromosome* encodes a feasible solution. A chromosome is modeled as an array of numbers or a string. Each element of the chromosome receives the name of *allele*. The *fitness value* of a chromosome is the value of the objective function of the solution the chromosome represents. The *decoder* is the function that associates a chromosome with a solution and computes its fitness value. A set of chromosomes is called a *population*.

A GA starts by generating a random population, then *evolves* it generating a new one. The new *generation* is the result of *combining* and *mutating* the chromosomes of the previous one. The combination or *crossover* of two chromosomes can be done, for instance, by randomly selecting an index, then taking the alleles from the first chromosome before the index



(a) Routing tree with size 9 (b) Routing tree with size 8 (c) Routing tree with size 7

Figure 2. When the sink finds a better routing tree, it delivers control packets to the nodes that are part of the new routing tree so that they update their routing table, establishing the new routing tree. The proposed solution keeps improving the solution for the current nodes that reported themselves as terminal nodes and sets the network configuration.

and the alleles from the second chromosome after the index resulting in a new chromosome. Mutation may be done by randomly changing some elements of the chromosome. The algorithm must be aware that when combining two chromosomes the new chromosome might not represent a feasible solution, so special care must be taken at this step.

Bean [12] proposed the *Random Key Genetic Algorithm (RKGA)* which takes a uniformly distributed random number in the range of $[0, 1]$ that is used as a key to codify the chromosomes as an array of these random numbers. The advantage of the RKGA is that, by using the random numbers as sort keys, at the time of making the crossover between two chromosomes, the resulting chromosome always encodes into a feasible solution.

Gonçalves and Resende [13] introduced the *Biased Random Key Genetic Algorithm*. A BRKGA ranks the chromosomes in a generation and selects the subset of chromosomes with the best fitness, which is denoted as the *elite set*. Then, in order to combine two chromosomes, we choose one chromosome from the elite set and the other one from the remaining chromosomes. In contrast with a normal Genetic Algorithm, BRKGA introduces mutants as a set of random chromosomes instead of performing mutations.

The initial population is a random set of p chromosomes. The following generations are generated as follows. The elite set, which is a subset of p_e chromosomes with better fitness, is directly copied into the following generation; this is called an elitist strategy. The *mutants*, is a set of p_m random chromosomes introduced into the following generation; their function is to avoid getting stuck into a local minimum. The $p - p_e - p_m$ remaining chromosomes come from the uniform crossover of an elite chromosome and a non-elite one with probability $\rho_e > 0.5$ of inherit each allele from its elite father. Figure 3 depicts this process.

The algorithm finishes after a given number of generations or when the reached solution is not significantly improved after a while. Since each chromosome is independent, the calculation to get its fitness value can be performed in a parallelized fashion. The API we use [14] let us define the number of threads for parallel decoding. Furthermore, the API offers the advantage of setting a number of independent populations which allows us to have a broader search.

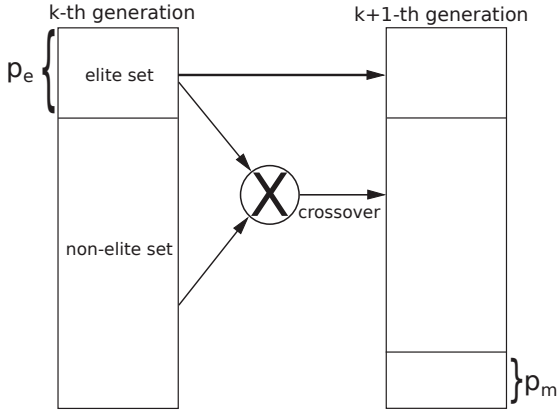


Figure 3. The BRKGA generates a new generation by directly copying the p_e chromosomes from the elite set; then generating p_m random chromosomes and finally combining an elite and a non-elite chromosome for the remaining $p - p_e - p_m$ chromosomes.

The parameters used in our algorithm are shown in Table I.

Table I
BRKGA PARAMETER SETTINGS

Parameter	Description	Value
p	Population Size	$2n$
p_e	Elite Set Size	10%
p_m	Number of Mutants	10%
ρ_e	Elite Crossover probability	0.6
K	Number of independent populations	3
MAXT	Number of threads for parallel decoding	4
MAXGEN	Maximum number of generations	100

A. Decoder

The decoder receives as input a chromosome and returns its fitness value. Each chromosome is an array of m random numbers in $[0, 1]$, where m is the number of edges in the network. Each random key, i.e. an allele, represents an edge in the graph. For each chromosome, a variation of the Kruskal's

algorithm [15] is executed. First, we perform a sort based on the random numbers assigned to each edge. Then, in a greedy fashion, we iterate over the sorted edges and add the current edge to the solution only if it does not form a cycle; thus forming a tree. Finally, we prune the tree by taking out the nodes that are not necessary, i.e., the ones not present in any path from the sink to any terminal node. The fitness value of the chromosome is the number of edges in this resulting tree.

When the first event occurs, the solution is the shortest path between the sink and the terminal node; when subsequent terminal nodes request for a routing tree, be either because of a new event appeared or because it also perceived the same event, the sink calculates the shortest path between the requesting node and the existing routing tree and it is codified into a chromosome that is considered part of the first generation of the BRKGA. In that way, the initial solution found by BRKGA is not much different than the current solution; hence generating fewer set messages and making BRKGA converge faster.

At the notification of an event ending, the BRKGA removes the notifying node from the set of terminal nodes. The pruning will erase the unneeded nodes from the solution.

The reason why we chose to use a BRKGA is, as mentioned above, it allows parallel decoding taking advantage of today's hardware. In addition, the algorithm will always keep improving the routing tree until, eventually, reaching the optimal tree. In case we do not have a hardware restriction we could let the algorithm run without stopping conditions.

V. Performance Evaluation

In this section we present the comparisons of our proposal made against SPT, DAARP and DDAARP. We evaluate the algorithm by using the simulator SinalGo version v.0.75.3 [16]. In all results, curves represent confidence intervals for 95 percent of confidence for 33 different instances. Table II shows the scenario parameters used in the simulation. According to the metric, some parameters will vary as described in each subsection. The first event starts at time 2.000s and the following events start at a uniformly distributed random time between $[2.000, 4.000]$ seconds. The events occur in random positions. For each simulation in which the number of nodes is varied, the sensor field dimension is adjusted to keep the density the same. We consider the network density as the relation $n\pi r_c^2/A$ where A is the area of the sensor field, r_c the communication radius and n the number of nodes. Sensors nodes are also randomly distributed.

The following metrics were used for the evaluation:

- **Overhead**: is the quantity of packets needed to setup the routing tree. These include the packets used in the Configuration Phase and the packets used at Request/Set Phase.
- **Number of transmissions**: are the totality of transmitted packets, that is, the overhead plus the data packets.
- **Tree cost**: is the number of Steiner Nodes in the routing tree.

Table II
General parameters used for the simulation

Parameter	Value
Sink node	1
Network size	1024
Communication radius (m)	80
# of events	3
Events radius (m)	50
Event duration ($hours$)	3
Loss probability (%)	0
Simulation duration ($hours$)	4
Notification interval (sec)	60
Sensor area (m^2)	700×700

- **Network Timelife**: is the time, in seconds, when a first node spent all its energy leaving it unable to function anymore.

A. Overhead

In this simulation scenario, we present the overhead as function of the number of events for network sizes of 256, 512, 1024 and 2048. Figure 4 shows that for setting up the routing tree, CER needs fewer packets than DAARP and DDAARP, about 39% and 41%, correspondingly. That is due to the fact that DAARP stores in each node a parameter that measures the distance to the routing tree and has to be updated each time a new event arrives, flooding many nodes with control packets. DDAARP needs control packets for each node that was selected as cluster-head.

In contrast, CER uses the shortest path to the sink for requesting a route and then only spends control packets on the nodes that are part of the routing tree. SPT has a constant overhead since it only performs the configuration once.

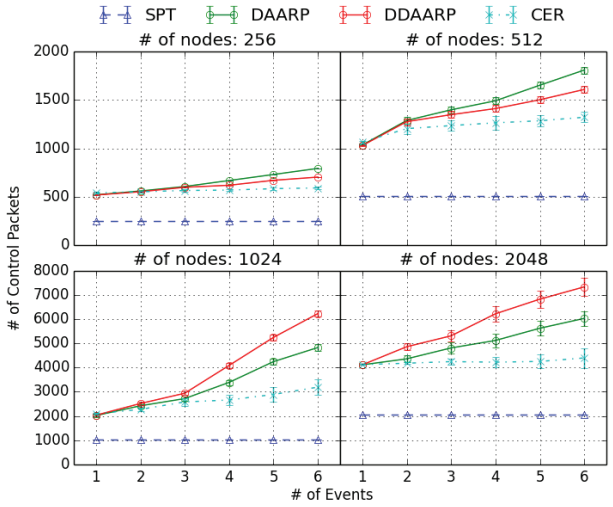


Figure 4. Overhead

As long as the network scales in the number of nodes, the difference gets bigger between DDAARP and CER; due to the fact that DDAARP needs to update many nodes in the network and CER only needs to update the ones that are part of the new routing tree. We can see that the growth rate in the

overhead for CER related to the number of events is very low, almost keeping constant, so our proposal has a good response when scaling the network.

Despite of the fact that SPT presents a low overhead, the routing trees generated by SPT have poor performance. The following metrics show that the data packets and the tree size are much bigger when compared to our results.

B. Number of transmissions

In this simulation scenario, we present the number of transmissions as function of the number of events for network sizes of 256, 512, 1024 and 2048. Figure 5 shows that CER generates fewer data packets; thus performing a better aggregation rate. Notice the difference between the algorithms is in the order of thousands of packets; hence having a positive impact on the batteries life. The impact of reducing the number of transmissions can be seen in energy consumption of the network, presented later.

For instance, for 6 events, CER has an improve of approximately 21% compared to DDAARP; 25% compared to DAARP and 33% compared to SPT. Depending on time the network operates; these savings on the packets will endure the networks life.

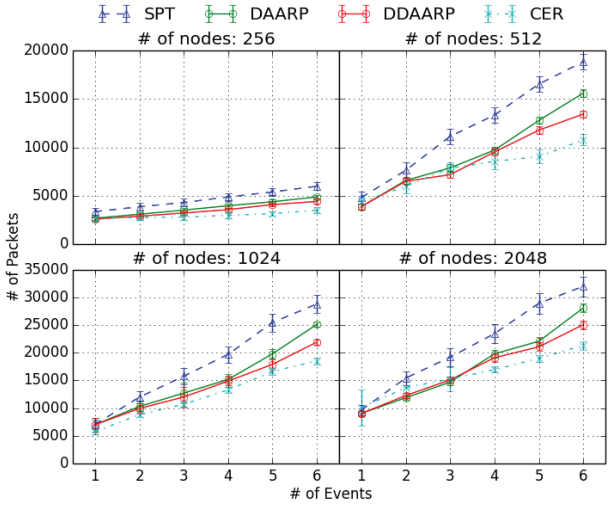


Figure 5. Datapackets

This metric also shows that CER, despite of generating more overhead than SPT, reduces the number of transmission for the same instances, consequently performing better data aggregation. This fact can be explained by the quality of the generated routing trees.

C. Tree Cost

In this simulation scenario, we present the size of the routing tree as function of the number of events for network sizes of 256, 512, 1024 and 2048. Because of CER keeps improving the routing tree; the tree cost presented here is the cost of the last tree computed by the sink using the BRKGA. Figure 6 shows that our proposal outperforms SPT, DAARP and DDAARP. This is due to the fact that all of them are heuristics that may be suitable for some kind of instances; however, in general there may exist instances on which they perform really bad. On the other hand, CER uses BRKGA to perform a broader search, independent of the instance. As discussed in Section IV, the BRKGA has many features to achieve a broad search and avoid failing to local minimum, so it is expected to have better routing trees.

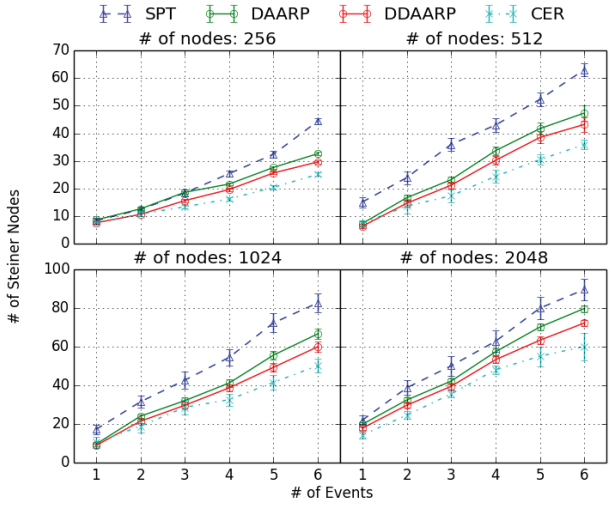


Figure 6. Tree Cost

For 6 events, CER is 37%, 25% and 17% more efficient than SPT, DAARP and DDAARP, respectively. This proves that the BRKGA is finding better quality trees and CER is configuring the network with those trees, having a positive impact on the energy consumption. As discussed earlier, CER generates more overhead than SPT, but this is countered by the good quality of the routing trees, which finally leads to a higher data aggregation rate.

D. Energy consumption

In this section we compare the total energy consumption. Figure 7 represents a map for the battery usage on the sensor field for instances of 1024 nodes. The color scale represents the mean of the battery usage. Table III shows the stats of the energy map. It is considered that each node spends a constant quantity of energy for each transmission and reception for the communication radius specified in Table II.

CER expends a total energy, i.e. the sum of the energy used by all the nodes, of 1991.03J, which represents an improvement of 16% when compared to DDAARP, 24% when compared to DAARP and 32% when compared to SPT. The second column of the table shows the mean of the energy spent by each sensor for a simulation time of 4 hours. This result reflects what was expected, a minor battery usage of the network mainly because of the good quality of the routing trees generated by the BRKGA.

Observe that Figure 7 present some isolated points. The reason why this happens is that this result presents the mean of instances with differing positions of nodes and events.

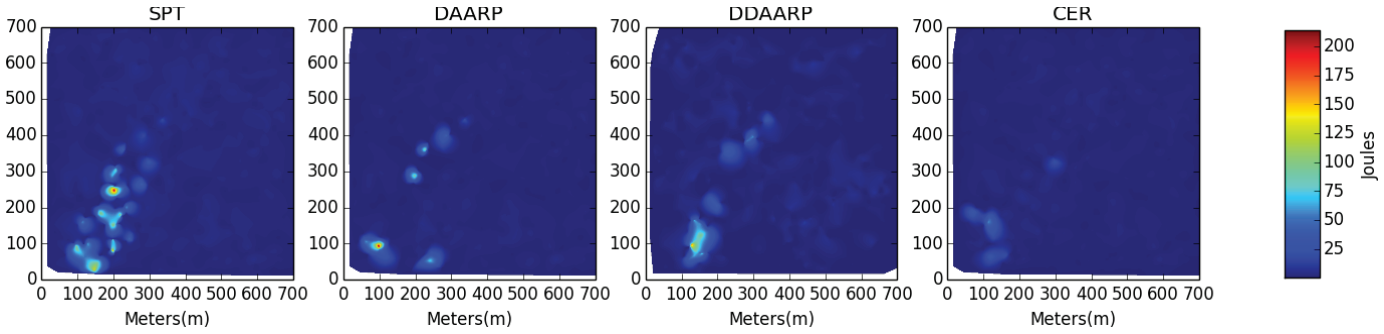


Figure 7. Mean of energy consumption for 1024 nodes

Table III
ENERGY MAP STATS

Algorithm	Total battery usage (J)	Battery usage mean (J)
SPT	2913.19	2.84
DAARP	2609.94	2.54
DDAARP	2356.40	2.30
CER	1991.03	1.94

E. Network Time Life

In this section we present the network timelife as a function of the data rate which is varied from 1 to 9. The size of the network, number of events and the other parameters are the ones presented in Table II. Figure 8 shows that CER has a greater timelife than other approaches, hence extending the network's timelife. In this metric we are just presenting the time on which appears the first node without energy, without evaluating the mechanisms each algorithm has to handle the disconnected node. With that in mind, it is clear that CER spends the node's energy in a more homogeneous way. For instance, for a data rate of nine, CER extends the network timelife by 28% and 100% compared to SPT and DAARP respectively.

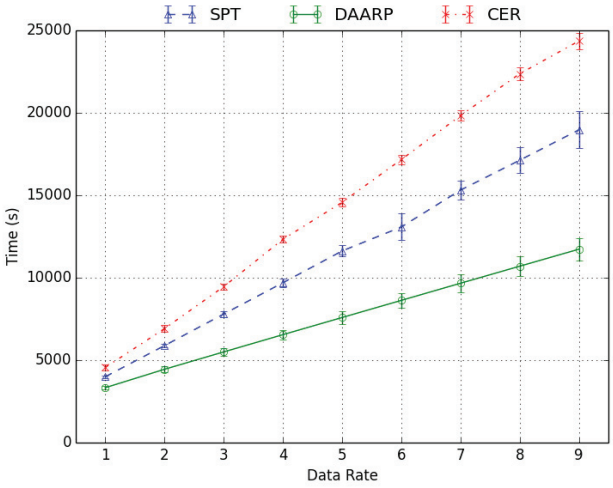


Figure 8. Network Time Life

VI. CONCLUSIONS AND FUTURE WORK

The main advantage of the presented solution combined with BRKGA is that the routing tree is closer to the optimal than existing approaches, consequently, leading to better routing paths. In addition, the algorithm is designed to avoid falling into local minimum which tends to extend the search domain getting better solutions. Also, our solution is independent of how the trees are computed. In fact, one could possibly develop other variants of the BRKGA proposed here or even consider other heuristics that improves the solution found over time and use it with our framework.

As future work, we are going to consider the battery use for building the routing tree. Therefore, the routing tree would change when a node has a considerable energy consumption, replacing it with another node that has more energy available. In all the presented schemes, we consider the nodes do not change their position. We also plan to consider scenarios where the nodes are moving, thus constantly changing the network structure.

ACKNOWLEDGMENT

This research was partially supported by grant #2013/21744-8, São Paulo Research Foundation (FAPESP) and grant #311499/2014-7, National Council for Scientific and Technological Development (CNPq).

REFERENCES

[1] M. P. urii, Z. Tafa, G. Dimi, and V. Milutinovi, "A survey of military applications of wireless sensor networks," in *2012 Mediterranean Conference on Embedded Computing (MECO)*, June 2012, pp. 196–199.

[2] X. Lu, S. Wang, W. Li, P. Jiang, and C. Zhang, "Development of a wsn based real time energy monitoring platform for industrial applications," in *Computer Supported Cooperative Work in Design (CSCWD), 2015 IEEE 19th International Conference on*, May 2015, pp. 337–342.

[3] "Environmental parameters monitoring in precision agriculture using wireless sensor networks," *Journal of Cleaner Production*, vol. 88, pp. 297 – 307, 2015.

[4] N. A. Pantazis, S. A. Nikolidakis, and D. D. Vergados, "Energy-efficient routing protocols in wireless sensor networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 551–591, Second 2013.

[5] J. N. Al-Karaki, R. Ul-Mustafa, and A. E. Kamal, "Data aggregation in wireless sensor networks - exact and approximate algorithms," in *High Performance Switching and Routing, 2004. HPSR. 2004 Workshop on*, 2004, pp. 241–245.

[6] B. Krishnamachari, D. Estrin, and S. Wicker, "The impact of data aggregation in wireless sensor networks," in *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, 2002, pp. 575–578.

[7] E. Nakamura, H. de Oliveira, L. Pontello, and A. Loureiro, "On demand role assignment for event-detection in sensor networks," in *Computers and Communications, 2006. ISCC '06. Proceedings. 11th IEEE Symposium on*, June 2006, pp. 941–947.

[8] L. A. Villas, A. Boukerche, R. B. Araujo, and A. A. Loureiro, "A reliable and data aggregation aware routing protocol for wireless sensor networks," in *Proceedings of the 12th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM Ó9. New York, NY, USA: ACM, 2009, pp. 245–252.

[9] L. Villas, A. Boukerche, R. de Araujo, and A. Loureiro, "Highly dynamic routing protocol for data aggregation in sensor networks," in *Computers and Communications (ISCC), 2010 IEEE Symposium on*, June 2010, pp. 496–502.

[10] E. Lalla-Ruiz, C. Expósito-Izquierdo, B. Melián-Batista, and J. M. Moreno-Vega, "A hybrid biased random key genetic algorithm for the quadratic assignment problem," *Information Processing Letters*, vol. 116, no. 8, pp. 513 – 520, 2016.

[11] J. S. Brando, T. F. Noronha, M. G. C. Resende, and C. C. Ribeiro, "A biased random-key genetic algorithm for single-round divisible load scheduling," *International Transactions in Operational Research*, vol. 22, no. 5, pp. 823–839, 2015.

[12] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization." *INFORMS Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.

[13] J. F. Gonçalves and M. G. Resende, "Biased random-key genetic algorithms for combinatorial optimization," *Journal of Heuristics*, vol. 17, no. 5, pp. 487–525, Oct. 2011.

[14] R. F. Toso and M. G. C. Resende, "A c++ application programming interface for biased random-key genetic algorithms," *Optimization Methods and Software*, vol. 30, no. 1, pp. 81–93, 2015.

[15] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[16] (2008) Sinalgo. simulator for network algorithms. Distributed Computing Group - ETH-Zurich. Last visited in October, 2008.

Fast Hybrid Network Reconfiguration for Large-Scale Lossless Interconnection Networks

Evangelos Tasoulas*, Ernst Gunnar Gran*, Tor Skeie*‡ and Bjørn Dag Johnsen†
*Simula Research Laboratory ‡University of Oslo †Oracle Corporation
{vangelis, ernstgr, tskeie}@simula.no bjorn-dag.johnsen@oracle.com

***Abstract*—Reconfiguration of high performance lossless interconnection networks is a cumbersome and time-consuming task. For that reason reconfiguration in large networks are typically limited to situations where it is absolutely necessary, for instance when severe faults occur. On the contrary, due to the shared and dynamic nature of modern cloud infrastructures, performance-driven reconfigurations are necessary to ensure efficient utilization of resources. In this work we present a scheme that allows for fast reconfigurations by limiting the task to subparts of the network that can benefit from a local reconfiguration. Moreover, our method is able to use different routing algorithms for different sub-parts within the same subnet. We also present a Fat-Tree routing algorithm that reconfigures a network given a user-provided node ordering. Hardware experiments and large scale simulation results show that we are able to significantly reduce reconfiguration times from 50% to as much as 98.7% for very large topologies, while improving performance.**

I. Introduction

High Performance Computing (HPC) clusters are massively parallel systems that consist of thousands of nodes and millions of cores. Traditionally, such systems are associated with the scientific community needs to run complex and high granularity computations. However, with the emergence of the cloud computing paradigm and Big-Data analytics, the computer science society tends to agree that there will be a convergence of *HPC* and *Big-Data*, with the *Cloud* being the vehicle for delivering the associated services to a broader audience [1], [2]. Large conventional HPC clusters are environments usually shared between users that run diversified, but predictable workloads. When exposed to the cloud and the more dynamic *pay-as-you-go* model however, the workload and utilization of the system can become very unpredictable, leading to the need for performance optimizations during runtime.

One of the components that can be tuned and reconfigured in order to improve performance is the underlying interconnection network. The interconnection network is a critical part in massively parallel architectures due to the intensive communication between nodes. As such, high performance network technologies that typically employ lossless layer-two flow control are used, as these technologies provide significantly better performance [3], [4]. Nevertheless, the performance comes at a cost of added complexity and management cost, and reconfiguring the network can be challenging [5]. Since packets are not getting dropped in lossless networks, deadlocks may occur if loops are allowed to form by the routing function. A Subnet Manager (SM) software is committed to administer the network. Among other tasks, this SM is responsible to compute deadlock-free communication paths between nodes in the network, and distribute the corresponding Linear Forwarding Tables (LFTs) to the switches. When a reconfiguration is needed, the SM must recalculate a new set of deadlock-free routes. During the transition phase, however, when distributing the new LFTs, a new routing function R_{new} coexists with the old routing function R_{old}. Although R_{old} and R_{new} are both deadlock-free, the combination of both might be not [6]. Moreover, the path computation is the costlier phase of a reconfiguration and can take up to several minutes, depending on the topology and the chosen routing function [7], introducing an obstacle that renders the reconfiguration to an extravagant operation that is avoided unless severe faults occur. In the case of faults, the reconfiguration is kept minimal in order to reestablish deadlock-free connectivity quickly, at the cost of degrading the performance [8].

In this paper we focus on the challenges of performance-driven reconfiguration in large-scale lossless networks. We introduce a hybrid reconfiguration scheme that allows for very fast partial network reconfiguration with different routing algorithms of choice in different subparts of the network. It is shown that our partial reconfigurations can be orders of magnitude faster than the initial full configuration, thus, we make it possible to consider performance-driven reconfigurations in lossless networks. The proposed scheme takes advantage of the fact that large HPC systems and clouds are shared by multiple tenants running isolated tasks. In such scenarios tenant inter-communication is not allowed [9], [10], thus the workload deployment and placement scheduler should try to avoid fragmentation to ensure efficient resource utilization [11], [12]. That is, the majority of the traffic per tenant can be contained within consolidated subparts of the network, subparts we can reconfigure in order to improve the overall performance. We use the Fat-Tree topology [13] and the Fat-Tree routing algorithm [14], [15] to demonstrate our work. We show that we are able to successfully reconfigure and improve performance within sub-trees by using a custom Fat-Tree routing algorithm that uses a provided node ordering to reconfigure the network. When we want to reconfigure the whole network, we use the default Fat-Tree routing algorithm, effectively exhibiting the combination of two different routing algorithms for different use-cases in a single subnet.

Our work is based on InfiniBand (IB) [16]. IB is a lossless interconnection network technology offering high bandwidth and low latency, thus making it very well suited for HPC and

978-1-5090-3216-7/16 $31.00 ©2016 IEEE

communication intensive workloads. IB is the most popular interconnect in the TOP500 supercomputers list, accelerating 40.8% of the systems in the list as of June 2016 [17].

The rest of the paper is organized as follows: Section II presents necessary background information, followed by related work in Section III. In Section IV we explain the working principles of our Fast Hybrid Reconfiguration, as well as our custom Fat-Tree routing algorithm that enables reconfiguration based on user-provided node-ordering. We evaluate a prototype implementation both on real hardware and by using simulations in Section V. In Section VI we conclude.

II. Background

A. InfiniBand Addressing Schemes

IB uses three different types of addresses [16]. First is the 16 bits Local Identifier (LID) which is a layer-two address. At least one unique LID is assigned to each *Host Channel Adapter* (HCA) port and each switch by the SM. The LIDs are used to route traffic within a subnet. Since the LID is 16 bits long, 65536 unique address combinations can be made, of which only 49151 (0x0001-0xBFFF) can be used as unicast addresses. Consequently, the number of available unicast addresses defines the maximum size of an IB subnet. Second is the 64 bits Global Unique Identifier (GUID) assigned by the manufacturer to each device (e.g. HCAs and switches) and each HCA port. Finally, the 128 bits Global Identifier (GID) is a valid IPv6 layer-three unicast address, and at least one GID is assigned to each HCA port and each switch. The GID is formed by combining a globally unique 64 bits prefix, assigned by the fabric administrator, and the corresponding GUID.

B. InfiniBand Subnet Management

IB is one of the first industry standard specifications implementing Software Defined Networking (SDN) principles, i.e. separating the control and the data plane [18]. An SM entity is running in one of the nodes in an IB fabric, and is responsible to discover and administer the subnet [16]. The SM assigns LID addresses to the HCA ports and switches, calculates deadlock free routes between all possible communicating pairs, and distributes the corresponding LFTs to each switch in the fabric. IB employees destination-based forwarding, so the information distributed in the LFTs is a mapping of destination LIDs to corresponding output ports used for forwarding at the switches. Once the LFTs have been distributed the network is operational, while the SM periodically sweeps the fabric for faults/changes and serves as a path-characteristics resolution service. A reconfiguration can be triggered either on demand by the fabric administrator, or forced when a fault or change is detected by a sweep. When the size of the fabric grows, the number of possible communication pairs increase polynomially. Consequently, the path computation increases polynomially as well, and depending on the topology and routing algorithm, the path computation can be in the order of several minutes [7]. OpenFabrics' OpenSM[1] is the most popular SM used on IB subnets, and the one that this work is based on.

[1]https://www.openfabrics.org/

C. The Fat-Tree Topology

The Fat-Tree is a scalable hierarchical network topology [19], [20], easy to build using commodity switches placed on different levels of the hierarchy [21]. The main idea behind the Fat-Trees is to employ *fatter* links between nodes, with more available bandwidth, towards the roots of the topology. The fatter links help to avoid congestion in the upper-level switches of the topology, and the bisection bandwidth is maintained. Different variations of Fat-Trees are presented in the literature, including *k*-ary-*n*-trees [19], Extended Generalized Fat-Trees (XGFTs) [20], Parallel Ports Generalized Fat-Trees (PGFTs) and Real Life Fat-Trees (RLFTs) [14].

A *k*-ary-*n*-tree [19] is an n level Fat-Tree with k^n end nodes and $n \cdot k^{n-1}$ switches, each with $2k$ ports. Each switch has an equal number of up and down connections in the tree. The XGFT Fat-Tree extends the *k*-ary-*n*-trees by allowing both different number of up and down connections for the switches, and different number of connections at each level in the tree. The PGFT definition further broadens the XGFT topologies and permits multiple connections between switches. A large variety of topologies can be defined using XGFTs and PGFTs. However, for practical purposes, RLFTs, a restricted version of PGFTs, are introduced to define Fat-Trees commonly found in today's HPC clusters [15]. An RLFT uses the same port-count switches at all levels in the Fat-Tree.

D. Fat-Tree Routing

The Fat-Tree routing algorithm (FTree) is a topology-aware routing algorithm for Fat-Tree topologies. For details on the FTree principles the reader may consult [14], [15], [22]. In this section we are going to briefly explain how the routing algorithm has been implemented in OpenSM; FTree first discovers the network topology and each switch is marked with a tuple that identifies its location in the topology. Each tuple is a vector of values in the form of $(l, a_h, ..., a_1)$, where l represents the level where the switch is located. The a_h represents the switch index within the top-most sub-tree, and recursively the digits a_{h-1} until a_1 represent the index of the sub-tree within that first sub-tree and so on. For a Fat-Tree with n levels, the *root*-level (topmost or core) switches are located in level $l = 0$, whereas the leaf switches (where nodes are connected to), are located in level $l = n - 1$. The tuple assignment for an example *2*-ary-*4*-tree is shown in Fig. 1.

Once the tuples have been assigned, FTree iterates through each leaf-switch in an ascending tuple order, and for each downward switch port where nodes are connected in an ascending port-order the algorithm routes the selected nodes based on their LID. In Fig. 2 we show different phases of how node a is routed. Switches in Fig. 2 are marked with numbers from $1-12$. FTree keeps port-usage counters for balancing the routes and starts by traversing the fabric upwards from the least loaded port while choosing the routes downwards, as shown in Fig. 2(a) with the red and green arrows respectively. In the first iteration all port counters are zero, so the first available upward port is chosen. For each level up, the newly reached switch (switch 5 in Fig. 2(a)) is selected as the switch to route

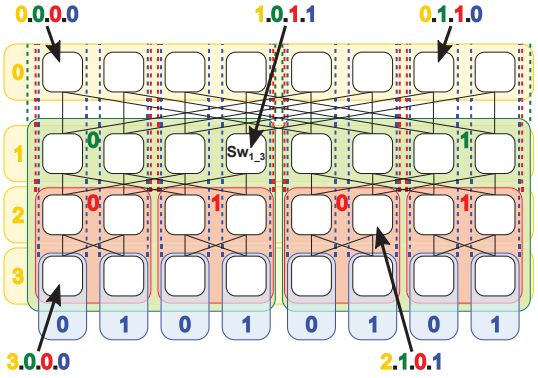


Fig. 1. k-ary-n-tree switch tuples assignment, where $k = 2, n = 4$. The nodes that are located at level 4 are omitted from this figure.

all the traffic downwards towards the selected node (node a), from the incoming port which through the switch was reached. Then the algorithm traverses the fabric downwards and assigns routes upwards towards that switch in a similar way as shown in Fig. 2(b). The same recursive operation continues until route entries for the selected node have been added to all of the necessary switches in the fabric, as depicted in Fig. 2(c) and 2(d). Next the algorithm will continue with the second node, node b in our case, as shown in 2(e), and so on until all nodes have been routed. With the resulting routing in Fig. 2(e), if node f sends a packet towards a, path $P1$ through switches $3 \rightarrow 7 \rightarrow 9 \rightarrow 5 \rightarrow 1 \rightarrow a$ will be used, while path $P2$ through switches $3 \rightarrow 8 \rightarrow 10 \rightarrow 6 \rightarrow 1 \rightarrow b$ will be used if f sends a packet to b. Note that in Fig. 2(d) the routing towards node a has been completed, but there are some *blank* switches without routes towards node a; the switches $6, 8, 10, 11, 12$. In reality, FTree add routes in these *blank* switches as well. If a packet towards a arrives for example in switch 12, this switch knows that it has to forward the received packet down towards switch 6, while switch 6 knows that the received packet from 12 has to be forwarded to switch 1 to reach its destination a. However, the switches in the lower levels will never forward traffic towards node a to switch 12 because the routes upward will always push the packets towards switch 9. Note that the use of a single root switch per destination node counters the growth of wide congestion trees [23].

III. Related Work

When a lossless network is reconfigured, routes have to be recalculated and distributed to all switches, while avoiding deadlocks. Note that the coexistence of two deadlock free routing functions, R_{old} and R_{new}, during the transition phase from the old to the new one, might not be deadlock free [6]. Zafar et al. [24] discuss the tools and applicable methods on the IB architecture (IBA), that would allow the implementation of the *Double Scheme* [25] reconfiguration method. *Double Scheme* is using Virtual Lanes (VLs) to separate the new and the old routing functions. Lysne et al. [26] use a token that is propagated through the network to mark a reconfiguration event. Before the token arrives at a switch, traffic is routed with the old routing algorithm. After the token has arrived and been forwarded through the output ports of the switch, the traffic is flowing with the new routing algorithm. The *Skyline* by Lysne et al. [27], speeds up the reconfiguration process by providing a method for identifying the minimum part of the network that needs to be reconfigured. Sem-Jacobsen et al. [8] use the channel dependency graph to create a channel list that is rearranged when traffic needs to be rerouted. The rearranging is happening in such a way, that no deadlocks can occur. Robles-Gómez et al. [28] use close up*/down* graphs to compute a new routing algorithm which is close to the old one, and guarantees that the combination of old and new routing during transition does not allow deadlocks to be introduced. Bermúdez et al. [29] are concerned with the long time it takes to compute optimal routing tables in large networks and the corresponding delay in the subnet becoming operational. They use some quickly calculated, but not optimal, provisional routes, and calculate the optimal routes *offline*. Since the provisional and the optimal routes are calculated based on the same acyclic graph, deadlock freedom is guaranteed. Zahid et al. [30] compares the old and the new routing in an attempt to minimize the reconfiguration cost by limiting the number of modifications. The result is less LFTs being distributed. The path-computation utilizes a meta-base generated during the initial path-computation phase in order to speed-up subsequent reconfigurations.

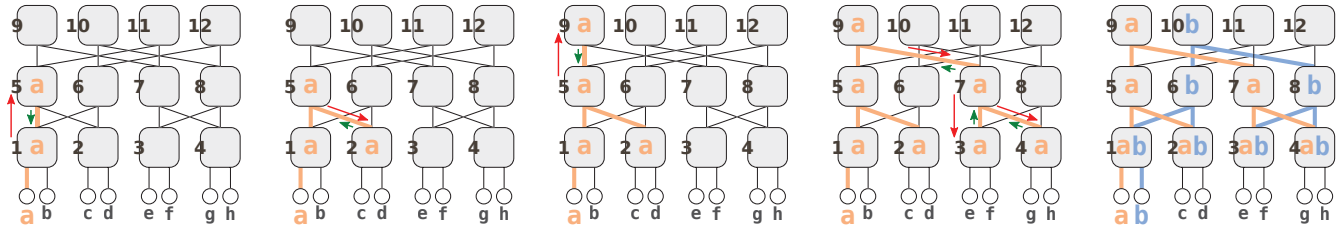
The above works either target quick, but not optimized, reconfiguration due to faults, or reconfigure the whole network based on a single routing algorithm. Due to the long path-computation time required by a full reconfiguration, offline calculation of paths with subsequent deployment is suggested by some. This paper differs by proposing a fast hybrid reconfiguration method that could be run often and online, for optimizing smaller parts of a shared network with the option to use multiple routing algorithms within a subnet.

IV. Fast Hybrid Reconfiguration Scheme

Our proposed reconfiguration scheme is based on the fact that HPC systems and cloud environments are shared by multiple tenants that run isolated tasks, i.e. tenant inter-communication is not allowed [9], [10]. To achieve better resource utilization, the workload deployment or virtual machine placement scheduler tries to avoid resource fragmentation to the extent possible. Consequently, per-tenant workloads are mapped onto physical machines that are close-by with regards to physical network connectivity, in order to avoid unnecessary network traffic and cross-tenant network interference [11], [12]. For Fat-Tree topologies with more than two levels[2], this means that the per-tenant traffic can usually be contained within a sub-tree of the multi-level Fat-Tree, as highlighted in Fig. 3.

Note that the traffic in the highlighted sub-tree in Fig. 3 is not flowing to and from the rest of the network. That is, we can apply a partial reconfiguration and optimize locally

[2]For two-level Fat-Trees, even when using the largest port-count switches available to build the topology, reconfiguring the whole fabric is still fast enough not to pose any real reconfiguration challenges [7]. Thus, in this paper we only consider larger Fat-Trees.



(a) Route down by going up (b) Route up by going down (c) Route down by going up (d) Route up by going down (e) a and b nodes routed

Fig. 2. Fat-Tree (FTree) Routing Phases

within the sub-tree based on the internal traffic pattern only. By applying such a partial reconfiguration, we effectively treat the reconfiguration as a Fat-Tree with less levels, and as we show in the evaluation Section V, the path-computation and overall reconfiguration cost can be substantially reduced (*Fast Reconfiguration*). In effect, performance-driven reconfiguration becomes attractive even in shared and highly dynamic environments. Moreover, when applying partial reconfiguration, we only need to alter the forwarding entries of the nodes within the sub-tree. Given that the initial routing algorithm used to route the fabric was FTree (Section II-D) or similar, that guarantees deadlock freedom by applying a variant of *up/down* routing without using Virtual Lanes [22], we can use any best-fit routing algorithm to reroute the given sub-tree as isolated (*Hybrid Reconfiguration*).

Note that once a sub-tree of a Fat-Tree is reconfigured, the connectivity between all end nodes, even those outside of the reconfigured sub-tree, is still maintained. As explained in Section II-D, all switches *know* where to forward traffic towards any destination. That is, every switch S has a valid forwarding entry in the LFT for every destination x, even if other nodes will never actually forward packets destined for x through S. Now, consider the case in Fig. 4 where the sub-tree highlighted with the dashed box has been reconfigured. The initial routing selected switch 5 to route traffic downwards towards node a and switch 6 to use towards node b (Fig. 2(e)). After the reconfiguration of the sub-tree, switch 5 is now used to route traffic towards node b and switch 6 towards node a, as presented in Fig. 4. In this case, if nodes c and d, located inside the sub-tree, send traffic towards nodes a or b, the newly calculated paths will be used. However, if the nodes e, f, g, h, located outside of the sub-tree, send traffic to a and b, the old paths will be used; the traffic towards a and b will enter the sub-tree at the switches 5 and 6, respectively, and not the other way around. Note that this behaviour external to the sub-tree could disturb the purpose of the sub-tree reconfiguration, e.g. by interfering with any sub-tree internal load balancing. Nonetheless, with no or very limited communication crossing the sub-tree boundary, such interference is of minor concern.

A. Choosing the Subset that will be Reconfigured

To apply our partial reconfiguration, we first need to choose all the nodes and switches in a sub-tree that have to be reconfigured. The method we use to identify the sub-tree uses the switch-tuples that have been discussed in Section II-D as a basis for comparison, and selects all nodes and switches in the sub-tree that needs to be reconfigured. The selection and consideration of all nodes in the sub-tree is necessary. Otherwise we may end up with imbalanced routing. The selection process of all entities in a sub-tree goes through the following steps:

1) The administrator (or an automated solution that monitors the fabric utilization) provides a list of nodes that must participate in the reconfiguration.
2) The tuples of the leaf switches of the nodes from step 1 are compared and the common ancestor sub-tree selected.
3) All the switches that belong to the sub-tree that was selected in step 2 will be marked for reconfiguration.
4) From the list of switches in step 3, the leaf switches will be picked and all of the nodes connected to the picked leaf switches will participate in the reconfiguration process.
5) Last, a routing algorithm has to calculate a new set of routes only for the nodes selected in step 4, and distribute the LFTs only to the switches selected in step 3.

B. Fat-Tree Routing with Custom Node-Ordering

As shown in [31], in multistage switch topologies like Fat-Trees, the effective bisection bandwidth is usually less than the theoretical bisection bandwidth for different traffic patterns. The reason is that depending on which node pairs have been selected for communication, there might be links that are shared in the upward direction. An example is illustrated in Fig. 5. In Fig. 5 nodes communicate within a two-level sub-part of a three-level Fat-Tree globally routed with the FTree routing algorithm. FTree routing chooses the switches with the corresponding color to route traffic downwards to the nodes with the same color. Although this sub-tree has a full theoretical bisection bandwidth[3], the effective bisection bandwidth in the illustrated communication pattern where nodes b, c and d send traffic to nodes e, i and m respectively, is 1/3 of the full bandwidth. All the destinations are routed through the same switch in the second level, switch 5, and the thick dashed link pointed by the arrow is shared by all three flows and becomes the bottleneck, although there are enough empty

[3]If there are enough links to avoid link sharing and obtain maximum speed when we split the network into two parts and each node from the first part communicates with only one node from the second part, the topology is said to have full bisection bandwidth.

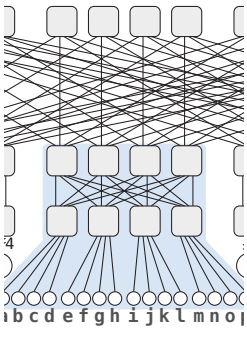


Fig. 3. A three-level Fat-Tree where workload from a single tenant is mapped within a two-level sub-tree.

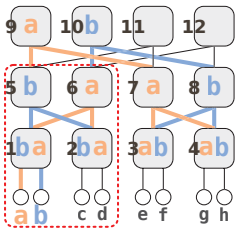


Fig. 4. Fat-Tree Routing multipathing after reconfiguring highlighted sub-tree.

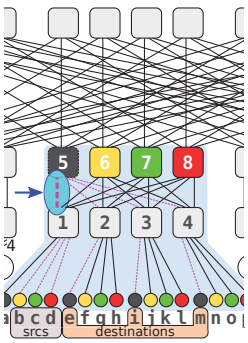


Fig. 5. Known pattern that degrades performance due to upward direction link sharing when source nodes *b*, *c*, *d* send traffic to destinations *e*, *i*, *m*.

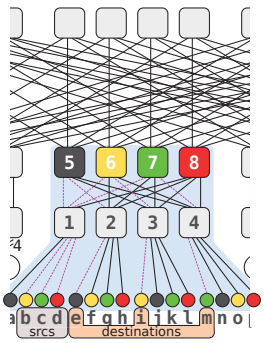


Fig. 6. NoFTree routing with node order e, i, m.

links to avoid link sharing and provide full bandwidth. Zahavi proposes and proves in [15] that if the MPI node ordering follows the node order that the FTree routing algorithm uses to route the nodes, full bisection bandwidth can be obtained for most of the MPI collective operations. Yet in dynamic cloud environments the provider may not be aware of the workload executed by different tenants, and cannot force users to use specific node orderings for communication. Furthermore, the users may need to run non-MPI-based programs as well. To allow for flexible reconfigurations that are not always bound to the same routing order that is based in the port order, we propose a new Fat-Tree routing algorithm, *NoFTree*, that uses a user-defined *N*ode *o*rdering to route a *F*at-*Tree* network. As we show in the Evaluation Section V having the ability to route a network with a given node order can be very effective, and one strategy to obtain better performance is to route nodes in the order of the amount of traffic each node receives. A simple way to determine the receiving traffic per node is to read the IB port counters [16]. In such a way the administrator doesn't have to know details about the jobs executed by tenants.

NoFTree is used in this work in the context of our Fast Hybrid Reconfiguration Scheme, and routes a sub-tree after the switches and nodes have been selected as described in Section IV-A. NoFTree works as follows:

1) An ordered list of nodes to be routed is provided by the user or by a monitoring solution.
2) NoFTree re-orders the nodes per leaf-switch. Then each ordered node is placed in the $n \ \% \ max\_nodes\_per\_leaf\_sw + 1$ slot to be routed in the given leaf-switch, where n is the global position of the node in the re-ordered list of nodes.
3) Remaining nodes that are connected to each leaf-switch, but not present in the provided node ordering list are filling the remaining leaf-switch routing slots based on the port order that nodes are connected to. If no port ordering is provided by the user, NoFTree will work exactly as the FTree routing algorithm.
4) NoFTree iterates through each leaf-switch again and routes each node based on the node order that has been constructed throughout the previous steps.

Fig. 6 illustrates how NoFTree can improve the performance in the example communication presented in Fig. 5 if the node order of the nodes that receive traffic is provided to the routing algorithm. In this case the node order is e, i, m. Since no node from leaf-switch 1 has been provided in the node ordering, nodes connected to switch 1 are routed based on the port order. Node e is the first node in the global node ordering and the first node to be ordered in leaf switch 2, so node e becomes the first node to be routed in switch 2 (routed downwards from switch 5). The rest of the nodes, f, g, h, are following the port order. Then the algorithm moves to the 3rd leaf switch where node i from the provided node ordering is connected. Node i is the second node in the global node ordering and the first node to be ordered in switch 3, so node i becomes the second node to be routed in switch 3, as explained on step 2 (routed downwards from switch 6). The nodes connected to switch 4 are routed in the same fashion. The resulting routing chooses the switches with the corresponding color to route traffic downwards to the nodes with the same color in Fig. 6, and in this scenario we can see a performance gain of 300% since there is no upward link sharing anymore.

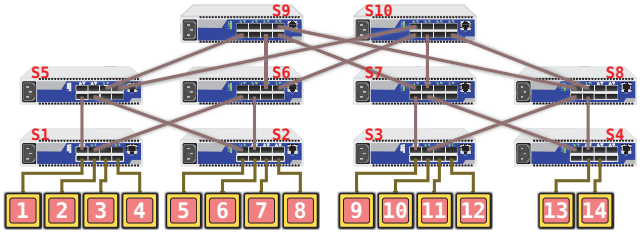
V. Evaluation

For the evaluation of our *Fast Hybrid Reconfiguration* we implemented both our subset-choosing method (IV-A) and NoFTree (IV-B) based on upstream OpenSM. We used our testbed with 10 IB QDR (32Gbps max effective speed) switches and 14 Compute Nodes (CNs) to evaluate our implementation on real hardware, and we used *ibsim* and *ORCS* simulator [32] to evaluate our methods in large scale subnets.

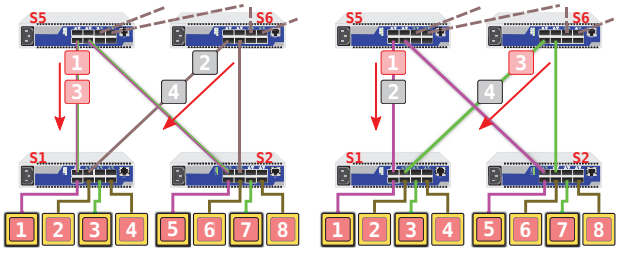
A. Experiments on Real Hardware

With our testbed we were able to build the PFGT[4] $(3;\ 4, 2, 4;\ 2, 2, 1;\ 2, 1, 1)$ that is lacking two CNs from the rightmost leaf switch as demonstrated in Fig. 7(a). We routed

[4]PGFTs are defined by the tuple: $(h; m_h, ..., m_1; w_h, ..., w_1; p_h, ..., p_1)$ where h is the number of levels in the tree; m_i is the number of lower level nodes connected to nodes on level i; w_i is the number of upper level nodes connected to nodes on level $i-1$ and p_i is the number of parallel links connecting nodes in level i and $i-1$ [15].



(a) Complete topology



(b) Before refonfiguration (c) After refonfiguration

Fig. 7. Experiments on real hardware.

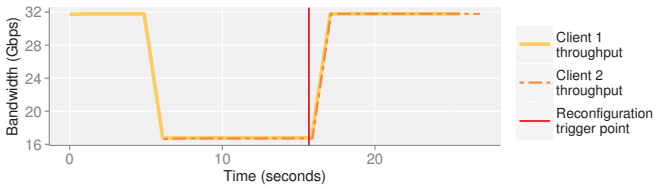


Fig. 8. Perftest running on topologies in Fig. 7(b) before the reconfiguration is triggered and Fig. 7(c) after a reconfiguration with NoFTree has been triggered.

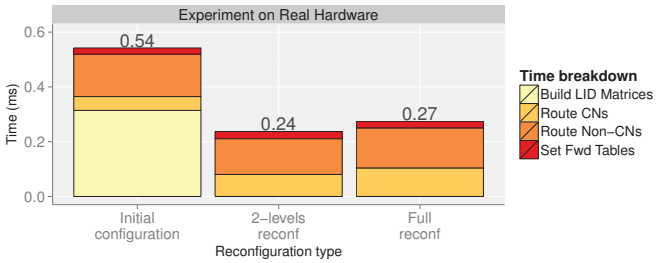


Fig. 9. Reconfiguration times for the topology in Fig. 7(a)

Topology Description	Num Nodes /Switches		Topology Description	Num Nodes /Switches
8-3-1-half	512/192		12-3-2-half	3456/432
8-3-1-full	1024/320		12-3-2-full	6912/720
8-3-2-half	1024/192		16-3-1-half	4096/768
8-3-2-full	2048/320		16-3-1-full	8192/1280
8-3-4-half	2048/192		16-3-2-half	8192/768
8-3-4-full	4096/320		16-3-2-full	16384/1280
12-3-1-half	1728/432		18-3-1-half	5832/972
12-3-1-full	3456/720		18-3-1-full	11664/1620

TABLE I
NUMBER OF NODES/SWITCHES IN THE SIMULATED TOPOLOGIES.

the topology with FTree and focused on the lower left sub-tree for our demonstration. We had two flows: $CN5$ sending traffic to $CN1$ and $CN7$ sending traffic to $CN3$ using the *perftest* utility. As one can see in Fig. 7(b), because FTree routes downwards both $CN1$ and $CN3$ from switch $S5$, our hypothesis is that the two flows should be shared and operate at half speed (~16Gbps per flow). If we reconfigure only the sub-tree with NoFTree and the correct node ordering we expect to get maximum speed (~32Gbps per flow) for both flows by routing downwards $CN1$ from $S5$ and $CN3$ from $S6$ as shown in 7(c). Moreover, since we only focus on a sub-tree we expect to get the reconfiguration faster than what if we reconfigured the whole subnet. We validate our hypothesis with the results presented in Fig. 8 and Fig. 9. In Fig. 8 we plotted the measured throughput per flow as measured from $CN5$ and $CN7$ (labeled as $Client1$ and $Client2$ respectively in Fig. 8). First $CN5$ starts sending traffic to $CN1$ at full speed. Around the 6th second and before we reconfigure with NoFTree, $CN7$ starts sending traffic to $CN3$ and the speed drops to half, until the moment we trigger the reconfiguration (marked with the solid vertical line) where we see that both flows jump back up to maximum speed. In Fig. 9 we present a stacked barplot with the times needed for OpenSM to complete different phases of the (re)configuration for the topology in Fig. 7(a). The left bar shows the time it took for the initial configuration. The middle bar shows the time it took to reconfigure only the lower left two-levels sub-tree from the topology in Fig. 7(a) with our Fast Hybrid Reconfiguration and NoFTree. The right bar shows the time needed to reconfigure the whole network with our Fast Hybrid Reconfiguration and FTree. An observation here is that most of the time in the initial FTree configuration is spent to build the necessary data structures (time marked as *Build LID Matrices* in the barplot), and every time a full reconfiguration is triggered with FTree, FTree tears down and rebuilds these data structures. With our method, even when we reconfigure the whole topology with FTree we carefully modify the data structures instead of rebuilding them from scratch, thus, we slightly increase the time to *Route CNs* in this small network, but overall, we reduce the reconfiguration time considerably. As such, we even reduce the time it takes to reconfigure the whole topology by 50%. However, the highlight of our method is when reconfiguring the sub-part of the topology. As illustrated by the middle bar in Fig. 9 the time it takes to route the CNs is reduced, and we reduce the time to reconfigure the network even further. An important note is that the experiments we did on real-hardware is just a proof of concept, and since the network we used is very small, our reconfiguration method doesn't make much of a difference. The simulations in the next section use much larger networks, that are typically used in big data centers, to show the significance of being able to reconfigure a topology partially with different routing algorithms.

B. Simulations

Ibsim was used to emulate all of the topologies presented in Table I, and OpenSM was used to calculate routes, and re-route the network with our fast reconfiguration method. The results are shown in Fig. 10, and the reconfiguration procedure and barplot description is the same as described in Section V-A. When comparing the results of the large simulated topologies

with the results from the real hardware experiments, it is noticeable that when the topology grows larger, reconfiguring only a sub-part of the network leads to considerable savings, and most of the time is spent to set the forwarding tables. In particular, for the largest network topology we tested, the *18-3-1-full*, the savings from our partial reconfiguration when compared with the initial configuration is 98.7%, and for a full reconfiguration 56.7%.

ORCS simulator was also used to test NoFTree routing algorithm when reconfiguring only a sub-part of the network, and demonstrate that partial reconfiguration can be beneficial if some bulk isolated communication is contained within a sub-tree of a Fat-Tree. ORCS simulator was enhanced significantly for this work, and all contributions have been pushed back to the project. For the ORCS simulations we implemented a new *receivers*[5] pattern. The receivers pattern will choose a number of receivers and will have other non-receiver nodes sending traffic towards the receivers with a user-provided chance. If a sender *decided* to not send traffic towards a receiver node, based on a second chance the sender will decide to either stay idle or send traffic to a non-receiver node. We ran the simulator with 1040 iterations for each unique case, and we picked hundreds of random sets of receivers while increased the number of receivers per switch and the number of switches where we distribute receiver nodes over time for different topologies. The same simulations were executed twice: once when the network was routed with the default FTree routing, and once after we reconfigured with NoFTree. In Fig. 11. we present the worst case and average case scenario results for different topologies. The bars indicate the percentage of max bandwidth achieved by the simulations. As one can observe, in all cases we achieved better results compared to the original FTree routing when the network was reconfigured with our partial reconfiguration method while using NoFTree, and for the worst case scenario we can see speed-ups of up to ∼25%.

VI. Conclusion

In this work we presented a reconfiguration scheme that allows for fast partial reconfigurations in sub-parts of a lossless network. Our aim is to make performance-driven reconfigurations practically usable in shared HPC systems and dynamic cloud environments, where multiple tenants share the infrastructure but run isolated tasks. When the bulk communication for different isolated tasks is contained within a sub-part of the network, our fast reconfiguration method optimize locally and speed up the reconfiguration process up to 98.7% with different routing algorithms of choice. We also presented a Fat-Tree-based routing algorithm, NoFTree, that is able to route a network given a user-provided node ordering. We used NoFTree to reconfigure sub-trees in Fat-Tree topologies, while we used the original Fat-Tree routing algorithm to reconfigure the complete topology when necessary, effectively demonstrating usage of multiple routing algorithms within a single subnet.

[5]The receivers pattern was selected as it is a common pattern (e.g. in server-client network architectures the servers are *receiver* nodes) that is easy to identify in an abstract way, for example, by reading port counters.

Acknowledgements

We would like to acknowledge Mellanox Technologies for providing us with IB hardware, and the Norwegian Research Council for funding the ERAC project (nr. 213283/O70).

References

[1] Satoshi Matsuoka, Hitoshi Sato, Osamu Tatebe, Michihiro Koibuchi, Ikki Fujiwara, Shuji Suzuki, Masanori Kakuta, Takashi Ishida, Yutaka Akiyama, Toyotaro Suzumura, Koji Ueno, Hiroki Kanezashi, and Takemasa Miyoshi, "Extreme big data (ebd): Next generation big data infrastructure technologies towards yottabyte/year," *Supercomputing frontiers and innovations*, vol. 1, no. 2, 2014.

[2] Asim Roy, Plamen Angelov, Adel Alimi, Kumar Venayagamoorthy, Theodore Trafalis, Jorge L. Reyes-Ortiz, Luca Oneto, and Davide Anguita, "Inns conference on big data 2015 program san francisco, ca, usa 8-10 august 2015 big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf," *Procedia Computer Science*, vol. 53, pp. 121 – 130, 2015.

[3] Sven-Arne Reinemo, Tor Skeie, and Manoj K Wadekar, "Ethernet for high-performance data centers: On the new ieee datacenter bridging standards," *IEEE micro*, vol. 30, no. 4, pp. 42–51, 2010.

[4] Jerome Vienne, Jitong Chen, Md Wasi-Ur-Rahman, Nusrat S Islam, Hari Subramoni, and Dhabaleswar K Panda, "Performance analysis and evaluation of infiniband fdr and 40gige roce on hpc and cloud computing systems," in *2012 IEEE 20th Annual Symposium on High-Performance Interconnects*. IEEE, 2012, pp. 48–55.

[5] Daniel Lüdtke and Dietmar Tutsch, "Lossless static vs. dynamic reconfiguration of interconnection networks in parallel and distributed computer systems," in *Proceedings of the 2007 Summer Computer Simulation Conference*, San Diego, CA, USA, 2007, SCSC '07, pp. 717–724, Society for Computer Simulation International.

[6] J. Duato, "A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 8, pp. 841–854, Aug 1996.

[7] Evangelos Tasoulas, Ernst Gunnar Gran, Bjorn Dag Johnsen, Kyrre Begnum, and Tor Skeie, "Towards the InfiniBand SR-IOV vSwitch Architecture," in *IEEE International Conference on Cluster Computing (CLUSTER), 2015*, Sept 2015, pp. 371–380.

[8] Frank Olaf Sem-Jacobsen and Olav Lysne, "Topology Agnostic Dynamic Quick Reconfiguration for Large-Scale Interconnection Networks," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 228–235.

[9] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen, and Tor Skeie, "Partition-Aware Routing to Improve Network Isolation in InfiniBand Based Multi-tenant Clusters," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2015.*, 2015, pp. 189–198.

[10] Eitan Zahavi, Alex Shpiner, Ori Rottenstreich, Avinoam Kolodny, and Isaac Keslassy, "Links as a Service (LaaS): Feeling Alone in the Shared Cloud," *arXiv preprint arXiv:1509.07395*, 2015.

[11] Weiwei Fang, Xiangmin Liang, Shengxin Li, Luca Chiaraviglio, and Naixue Xiong, "Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Computer Networks*, vol. 57, no. 1, pp. 179 – 196, 2013.

[12] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[13] Charles E Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[14] Eitan Zahavi, "D-Mod-K routing providing non-blocking traffic for shift permutations on real life fat trees," *CCIT Report 776, Technion*, 2010.

[15] E. Zahavi, "Fat-tree routing and node ordering providing contention free traffic for MPI global collectives," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1423–1432, 2012.

[16] InfiniBand Trade Association, "InfiniBand Architecture General Specifications 1.3," 2015.

[17] TOP500, "Top500.org," http://www.top500.org/, 2016.

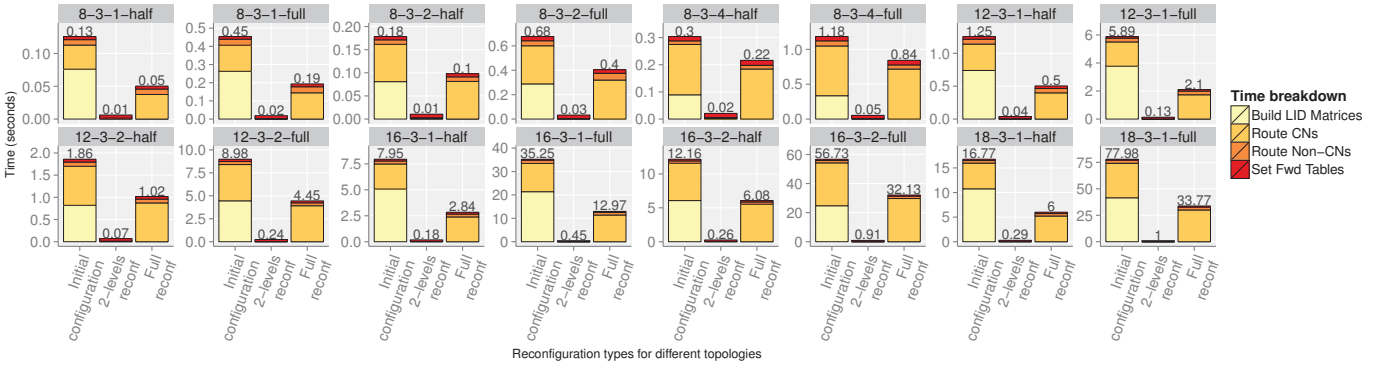
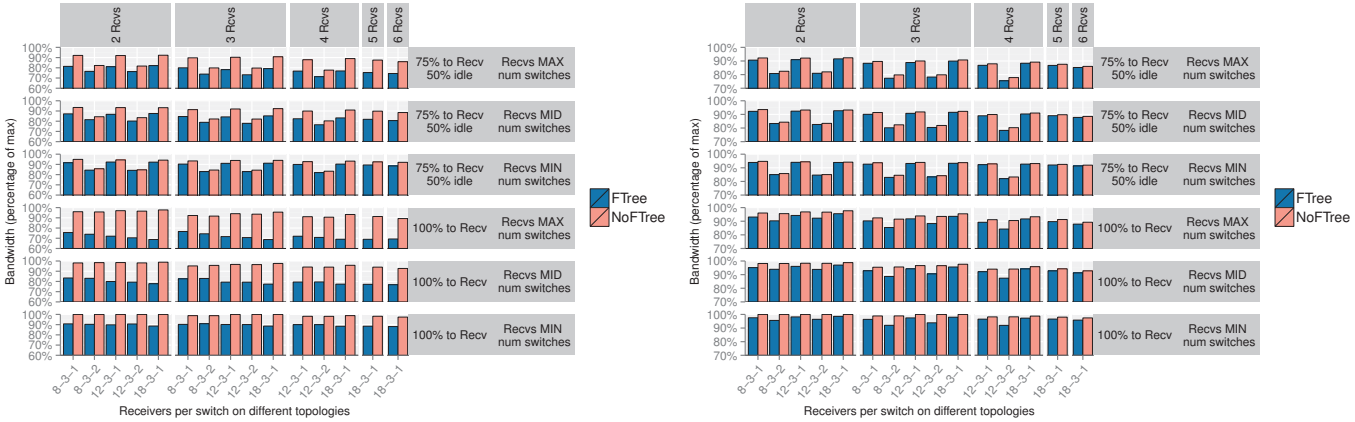


Fig. 10. Reconfiguration times for simulated topologies.



(a) Worst case scenario

(b) Average case

Fig. 11. ORCS simulation results (Higher bars indicate better performance).

[18] M. K. Shin, K. H. Nam, and H. J. Kim, "Software-defined networking (sdn): A reference architecture and open apis," in *2012 International Conference on ICT Convergence (ICTC)*, Oct 2012, pp. 360–361.

[19] Fabrizio Petrini and Marco Vanneschi, "k-ary n-trees: High performance networks for massively parallel architectures," in *Proceedings of the 11th International Parallel Processing Symposium, 1997.* IEEE, 1997, pp. 87–93.

[20] Sabine R Ohring, Maximilian Ibel, Sajal K Das, and Mohan J Kumar, "On generalized fat trees," in *Proceedings of the 9th International Parallel Processing Symposium, 1995.* IEEE, 1995, pp. 37–44.

[21] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*. ACM, 2008, vol. 38, pp. 63–74.

[22] Eitan Zahavi, Gregory Johnson, Darren J Kerbyson, and Michael Lang, "Optimized InfiniBand fat-tree routing for shift all-to-all communication patterns," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 217–231, 2010.

[23] Ernst Gunnar Gran, *Congestion Management in Lossless Interconnection Networks*, phd, Faculty of Mathematics and Natural Sciences, University of Oslo, March 2014.

[24] Bilal Zafar, Timothy M Pinkston, Aurelio Bermúdez, and Jose Duato, "Deadlock-Free Dynamic Reconfiguration Over InfiniBand™ Networks," *Parallel Algorithms and Applications*, vol. 19, no. 2-3, pp. 127–143, 2004.

[25] Ruoming Pang, Timothy Mark Pinkston, and José Duato, "The Double Scheme: Deadlock-free Dynamic Reconfiguration of Cut-Through Networks," in *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. IEEE, 2000, pp. 439–448.

[26] Olav Lysne, José Miguel Montañana,Timothy Mark Pinkston, José Duato, Tor Skeie, and José Flich, "Simple Deadlock-Free Dynamic Network Reconfiguration," in *High Performance Computing-HiPC 2004*, pp. 504–515. Springer, 2005.

[27] Olav Lysne and José Duato, "Fast Dynamic Reconfiguration in Irregular Networks," in *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. IEEE, 2000, pp. 449–458.

[28] Antonio Robles-Gómez, Aurelio Bermúdez, Rafael Casado, and Åshild Grønstad Solheim, "Deadlock-Free Dynamic Network Reconfiguration Based on Close Up*/Down* Graphs," in *Euro-Par 2008–Parallel Processing*, pp. 940–949. Springer, 2008.

[29] Aurelio Bermúdez, Rafael Casado, Francisco J Quiles, and Jose Duato, "Use of Provisional Routes to Speed-up Change Assimilation in InfiniBand Networks," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 186.

[30] Feroz Zahid, Ernst Gunnar Gran, Bartosz Bogdański, Bjørn Dag Johnsen, Tor Skeie, and Evangelos Tasoulas, "Compact network reconfiguration in fat-trees," *The Journal of Supercomputing*, pp. 1–30, 2016.

[31] T. Hoefler, T. Schneider, and A. Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *2008 IEEE International Conference on Cluster Computing*, Sept 2008, pp. 116–125.

[32] Timo Schneider, Torsten Hoefler, and Andrew Lumsdaine, "ORCS: An oblivious routing congestion simulator," *Indiana University, Computer Science Department, Tech. Rep*, 2009.

An Offset Based Global Sleeping Schedule for Self-Organizing Wireless Sensor Networks

Stephanie Imelda Pella, Prakash Veeraraghavan, Somnath Ghosh
Department of Comp. Science and Information Technology
La Trobe University
Melbourne, Australia
S.Pella, P.Veera, S.Ghosh @ latrobe.edu.au

Abstract— **In wireless sensor networks (WSNs), conserving the nodes' energy is one the main motivations in designing the medium access control (MAC) layer protocols. A common approach is to allow the nodes to turn their radio modules off periodically according to certain schedules. The nodes that operate on a common schedule and located in a common physical area form a virtual cluster. The nodes in the cluster borders have more active period to maintain the connectivity between clusters, thus, have shorter life spans. This work proposes a scheme that enables cluster merging in a distributed environment to eliminate the problem. The proposed scheme uses the schedule offset, the time difference between the starts of the active periods in two schedules, as the criteria for deciding the direction of the merging. We evaluate the performance of the proposed protocol using a mathematical estimation and simulation. The result shows the proposed scheme has up to 50 times shorter convergence time and save up to 90% more energy during the merging process compared to the existing global schedule protocols.**

Keywords— Sensor Network; Energy Efficiency; Duty-Cycle; Global Sleeping Schedule

I. INTRODUCTION

Energy conservation is one of the central issues in Wireless Sensor Network (WSN). To prolong the network life, medium access control (MAC) protocols in WSN are designed to be energy efficient. The most common approach to conserve energy is the use of the duty cycle scheme. The scheme allows the nodes to turn their radio components off and operate in a sleeping mode to preserve their energy. Periodically the nodes need to 'wake up' by switching on their transceiver to check the channel activity or transmitting packets. Each node maintains a schedule that determines when it needs to sleep and wake up.

The duty cycle-based protocols can be classified into the synchronized duty cycle protocols [1-4] and the asynchronous duty cycle protocols. The synchronous scheme requires the nodes to synchronize their schedules with their neighbors, instead, nodes in the asynchronous scheme choose their schedules independent to their neighbors' schedules. Asynchronous duty cycle protocols outperform the synchronous ones in term of energy saving, however, they yield a much lower throughput [5].

In the synchronous duty cycle-based MAC protocols, the nodes may randomly create their schedules in the initialization stage if they fail to discover an existing schedule, hence, introducing the existence of more than one schedule in the network. Neighboring nodes that operate on a common schedule form a virtual cluster. The nodes located near the border of two or more clusters need to operate on the active mode of all the bordered clusters to act as the gateways for the clusters. This causes the nodes to exhaust their energy much sooner than the rest of the network.

Studies in [6-8] shows that the non-uniform energy depletion rates of the nodes adversely impact the network lifespan and performance. In the worst case, it fragments the network. Since each border node commonly acts as a gateway between clusters, when it exhausts its energy and is forced to leave the network, one or more cluster could be isolated from the rest of the network. This, as a result, shortens the useful time of the network.

In this study, we propose an energy-efficient scheme to achieve a single sleeping schedule in WSNs. In our proposed scheme, in discovering another cluster, a border node uses the difference between the starting of the active periods in the two clusters to decide whether or not its cluster needs to merge to the newly discovered cluster. The merging decision is made independently, but consistently, for all the nodes in a cluster. The simulation shows that our proposed scheme outperform the other existing global schedule protocols [6-8] in term of merging time, energy consumption and control packet overhead.

The rest of the paper is organized as follows. In section 2, we introduce the basic mechanism of our schemes such as the offset of two schedules and virtual frame. In section 3, we present the cluster merging algorithm and in section 4, we analyze the performance of the proposed protocol trough simulations. Finally, section 5 summarizes the paper.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

II. Basic Offset Based Global Schedule Scheme

In a global schedule protocol, a node that discovers more than one schedule uses a set of criteria to choose which schedule it will implement. This section presents a scheme that uses the offset of two schedules in determining the winning schedule.

The synchronous duty cycle-based protocols, such as S-MAC[1, 2], T-MAC[3], and DS-MAC[4], divides the nodes operational time into frames. A frame constitutes of an active/wake-up time and a sleeping time as shown in Fig. 1.. Nodes that do not participate in data transmission turn their radio modules off during sleeping time. In each cycle, which is a group of 10 frames, each node randomly chooses a SYNC frame, during which it broadcasts its SYNC to announce its schedule. In every ten cycles, nodes randomly pick a cycle as their synchronization cycle. During its synchronization cycle, a node stays active for the duration of the entire cycle to ensure it receives any sent SYNC in its neighborhood.

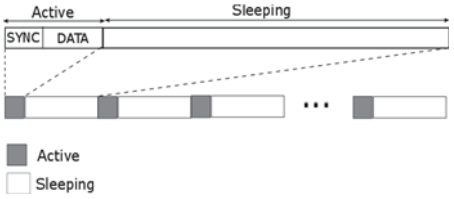


Fig. 1 A Frame in Synchronized Duty Cycle Based Protocols

The proposed protocol requires the nodes to calculate the offset of its schedule and the newly founded schedule when it receives a SYNC advertising a schedule of a different cluster. In equation (1), d_{S1S2} is the schedule offset between S_1 and S_2 as seen by the nodes in cluster S_1, t_{S1} and t_{S2} are the durations of time until the starts of the next frames in schedule S_1 and S_2 respectively, and T is the length of a frame.

$$d_{S_1S_2} = (t_{S_2} - t_{S_1}) \, mod \, T \qquad (1)$$

The offset of two schedules S_1 and S_2, is the minimum duration of the start of a frame in S_2 precedes the start of a frame in S_1 as shown in Fig. 2. Nodes in cluster S_1 need to merge with cluster S_2 if schedule S_2 precedes S_1, i.e. the offset of S_1 and S_2 is more than half of the frame length ($d_{s1s2} > ½$ T).

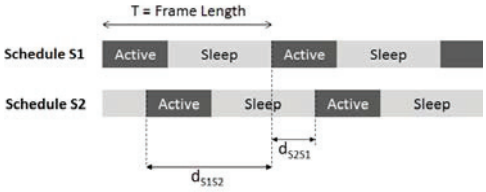


Fig. 2 The Offset of Two Schedules

The use of schedule offset as a winning criterion overcomes two drawbacks in previous proposed global schedule protocols. Firstly, it avoids a big control overhead resulted from the using of schedule age as winning criteria as proposed in [6] since the nodes does not need additional information for calculating the offset. And, secondly, it eliminates the problem of having different schedules with the same schedule ID in a rechargeable sensor network, unlike the use of ID as winning criterion[7, 8]. However, it creates a problem when the offset is exactly half of the frame length ($d_{s1s2} = ½$ T) as neither of the schedules precedes the other one.

In order to solve the problem we introduce a virtual frame that consists of two frames as shown in Fig. 3.

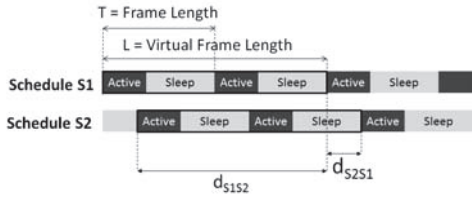


Fig. 3 A Virtual Frame

In discovering a new schedule, a node calculates the offset of its schedule and the newly received ones using equation (2).

$$d_{S_1S_2} = (t_{S_2} - t_{S_1}) mod \, L \qquad (2)$$

In this scheme, if the offset of two schedules S1 and S2 is half of the virtual frame, the nodes in the two clusters essentially wake up and sleep at the same time, hence the clusters do not need to merge.

Since the propagation delay of SYNC packets affects the way a node sees the start of the next frame of a schedule, and consequently the offset of two schedules, we need to consider the maximum propagation delay of the packets. This could be obtained by calculating the maximum propagation path in the network, for example by using Hilbert's space filling curve algorithm. Let δ_{max} be the maximum propagation delay in the network, when a node follows the rules listed in TABLE 1 in receiving SYNC from another cluster

TABLE 1 Merging Rules

Offset Value	Action
[½ L + δ_{max} , L]	Merge to the newly discovered cluster
[0, ½ L - δ_{max}]	Broadcast SYNC to notify the discovered cluster of the schedule of the node's cluster.
[½ L - δ_{max} ,½ L + δ_{max}]	Do not need to merge

III. The Cluster Merging Algorithm

A. Synchronization Control Packet Formats

In the proposed protocol, we use two control packets for schedule synchronization, namely, SYNC and SYNC-M packets. Similar to the mechanism proposed in S-MAC [1, 2], each node periodically sends SYNC for announcing its schedule. When nodes in cluster S_1 decides to merge into cluster S_2, cluster S_1 is called the merging cluster and cluster S_2

is called the destination cluster. The nodes in the merging cluster and in the destination cluster broadcast SYNC-Ms to notify their neighborhood of the merging.

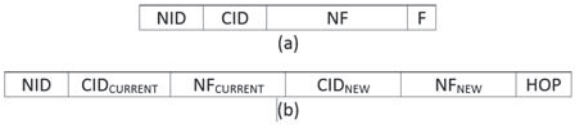


Fig. 4 Control Packets (a) SYNC (b) SYNC-M

In Fig. 4, a SYNC packet consists of the ID of the node that sends the packet (NID), cluster ID (CID), the duration until next virtual frame (NF) and a one-bit flag (F). F is set to 1 when the cluster is in the merging process. A SYNC-M packet consists of Node ID, the current cluster ID, the duration until next frame in the current cluster, the newly received cluster ID, the duration until the next frame in the newly received cluster and the number of hops (HOP). HOP starts at 1 and is increased every time the packet rebroadcasted.

B. The Merging Scheme

Each node in the network obeys the following steps in choosing its schedule on receiving a SYNC:

1) If the node receives the SYNC before it has chosen its schedule, it adopts the advertised schedule and announce this schedule in its SYNC.

2) If the node receives the SYNC advertising a schedule of a different cluster after it has chosen a schedule, it calculates the offset (d) between its schedule and the newly received schedule and takes an action based on the merging rules in TABLE 1. There are two cases to consider:

a) If the node decides to merge to the newly discovered cluster, it creates and broadcast a SYNC-M to notify the members of its current and destination cluster of its decision. It then sets a timer twait. At the end of twait, if there is no interruption (detailed in step 3), it merges to the destination cluster.

b) If the node does not decide to merge to the newly discovered cluster, it broadcasts a SYNC advertising its schedule during the next active time of the other cluster.

3) If the node goes with the condition in step 2(a), during the waiting period t_{wait}, it ignores all received SYNCs announcing the other clusters and operates on both schedules. If the node receives a SYNC-M

a) If the SYNC-M packet has the same cluster id, and advertises that another cluster, with a bigger cluster ID than CIDNEW decides to merge with its current cluster, it cancels its timer and redo step 2.

b) Otherwise, the node merges with the destination cluster at the end of twait.

On receiving a SYNC-M packet, the nodes in both merging and destination clusters rebroadcast the SYNC-M. Any node that hears the SYNC-M sets t_{wait} timer. During t_{wait}, the nodes in the destination cluster ignore all the SYNC packets from any other cluster and the nodes in the cluster other than the merging and the destination cluster do not start a merging process with the merging cluster. At the end of t_{wait}, the nodes in the merging cluster merge with the destination cluster.

C. Upper Bound of Time and Energy Spent in a Merging Process

The duration of waiting time, t_{wait}, is twice of the maximum time needed for the SYNC-M to propagate to another furthest end of the cluster. In every hop the packet has traveled, we exclude that hop in calculating the waiting time. Each node in the network uses equation (3) to calculate its specific waiting time t_{wait}, where N is the maximum number of hops in a network, HOP is the number of hops the packet has been traveled , T is the duration of a frame and L is the duration a virtual frame.

$$t_{wait} = 2 * (N - HOP) * T = (N - HOP) * L \tag{3}$$

After receiving a SYNC-M, a node needs to wait until the next frame to rebroadcast the packet. Therefore, the upper bound of the time needed to propagate the packet to the whole cluster equals to the maximum number of hops between two furthest nodes in the cluster times the duration of a frame. We then compare it to convergence times of the other two existing global schedule protocol, S-MACL[7, 8]and GSA[6], where the node merge individually instead of a cluster at a time. This means that each node needs to re-discover another cluster before starting the merging process. Since a node can only discover another cluster during the synchronization cluster (which happen once every 100 frames, according to S-MAC [1, 2]). This causes the convergence time in both SMAC-L and GSA is up to 50 times bigger than the one in the proposed protocol with cluster merging scheme.

$$\begin{aligned} t_{conv_{offset}} &= O[N * L] \\ t_{conv\_other} &= O[N * 10C] = O[50 * N * L] \end{aligned} \tag{4}$$

During the merging process, nodes exchange their control packets including the periodic SYNCs (in the existing protocols and our proposed protocol) and SYNC-Ms (in our proposed protocols). The energy spent for sending the control packet during the merging process is shown in equation (5), where [SYNC] and [$SYNC_M$] are the size of SYNC and $SYNC_M$ respectively in bit and e is the energy needed to send one bit.

$$\begin{aligned} O[E_{offset}] &= \frac{1}{5L} O[t_{conv_{offset}}] * [SYNC] * e + [SYNC_M] * e \\ O[E_{other}] &= \frac{1}{5L} E[t_{conv_{other}}] * [SYNC] * e \end{aligned} \tag{5}$$

IV. The Cluster Merging Algorithm

A. The Simulation Environment

To validate our proposed protocol, we run extensive simulations in MATLAB 2014 environment. We use the parameters that are used in the performance evaluation of S-MAC protocol in [2]. The sizes of SYNC and SYNC-M packets are asumed to be 2 and 4 bytes based on the information contained in the packets.

To investigate the worst scenario in the network, we consider a chain topology network as shown in Fig. 5. We compare the performance of our proposed protocol with the performance of ID based global schedule protocol (SMAC-L[7, 8]).

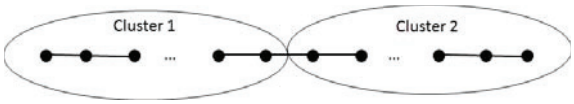


Fig. 5 Chain Topology

B. Results and Discussion

Fig. 6 shows the convergence time (i.e. the merging time) of two clusters. It is the duration between the first time a node in the merging cluster receives the SYNC of the destination cluster and the time the merging concludes.

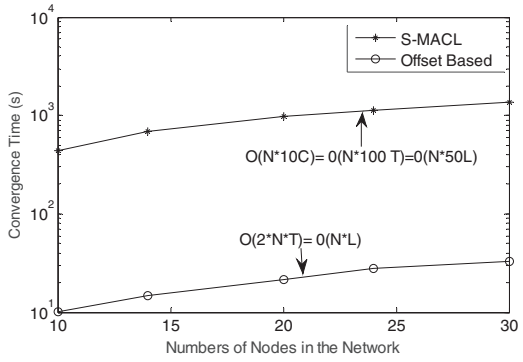


Fig. 6 Convergence Time in Chain Topology Network

The solid lines represent the simulation result and the dashed lines represent the upper bound of the convergence time as shown in equation (4). The result shows that our proposed protocol has much smaller convergence time (roughly 50 times smaller) compared to S-MACL protocol.

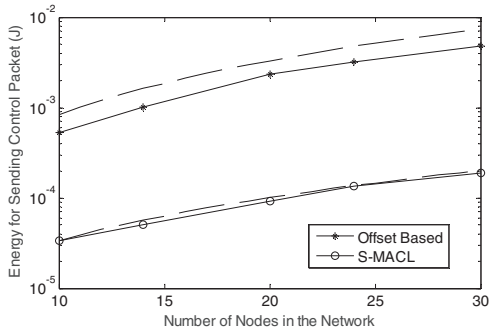


Fig. 7 Energy Spent during Convergence Time Chain Topology

In Fig. 7, we plot the energy spent by the network for sending control packet during a merging process. Similar to the previous figure, the solid lines represent the simulation result and the dashed lines represent the expected value as shown in equation (5). The result shows that even though the offset based protocol has a bigger control packet size compared to S-MACL, due to the smaller convergence time, during the merging process it spends less energy than the id-based protocol does. On average, in a chain network, the energy spent for sending the control packets in our proposed protocol is about 10 % of the one in S-MACL.

V. Summary

This paper has presented the development of a global sleeping schedule protocols in for a self-organizing WSN. We proposed the use of the offset between two schedules as winning criteria in the merging process to overcome the problems resulted from the use of schedule ID and schedule age for the winning criteria as proposed by the existing global schedule protocols. We proposed a cluster merging algorithm, in which, each time a border node discover a new cluster and decides to join the cluster, it propagates its decision to its entire cluster. We measure the performance of the proposed protocol in term of the convergence time and the energy spent during the merging process in a . The result shows that the merging time in our proposed protocol is up to 50 times less the existing global schedule protocol (S-MAC L [52]). Moreover, during the merging process, our proposed protocol saves up to 90% more energy than S-MACL.

[1] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, 2002, pp. 1567-1576.

[2] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," Networking, IEEE/ACM Transactions on, vol. 12, pp. 493-506, 2004.

[3] T. Van Dam and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in Proceedings of the 1st international conference on Embedded networked sensor systems, 2003, pp. 171-180.

[4] P. Lin, C. Qiao, and X. Wang, "Medium access control with a dynamic duty cycle for sensor networks," in Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE, 2004, pp. 1534-1539.

[5] P. Huang, L. Xiao, S. Soltani, M. W. Mutka, and N. Xi, "The evolution of MAC protocols in wireless sensor networks: A survey," IEEE Communications Surveys & Tutorials, vol. 15, pp. 101-120, 2013.

[6] Y. Li, W. Ye, and J. Heidemann, "Energy and latency control in low duty cycle MAC protocols," in Wireless Communications and Networking Conference, 2005 IEEE, 2005, pp. 676-682.

[7] S. Ghosh, P. Veeraraghavan, S. Singh, and L. Zhang, "Performance of a wireless sensor network mac protocol with a global sleep schedule," International Journal of Multimedia and Ubiquitous Engineering, vol. 4, pp. 99-114, 2009.

[8] L. Zhang, S. Ghosh, P. Veeraraghavan, and S. Singh, "An energy efficient wireless sensor MAC protocol with global sleeping schedule," in Computer Science and its Applications, 2008. CSA'08. International Symposium on, 2008, pp. 303-308.

Label Encoding Algorithm for MPLS Segment Routing

Rabah Guedrez* Olivier Dugeon* Samer Lahoud† Géraldine Texier‡
*Orange Labs, Lannion, France,
rabah.guedrez@orange.com, olivier.dugeon@orange.com
†*IRISA/Université de Rennes 1/Adopnet Team, Rennes, France, samer.lahoud@irisa.fr
‡IRISA/Télécom Bretagne/Adopnet Team, Rennes, France, geraldine.texier@telecom-bretagne.eu

***Abstract*—Segment Routing is a new architecture that leverages the source routing mechanism to enhance packet forwarding in networks. It is designed to operate over either an MPLS (SR-MPLS) or an IPv6 control plane. SR-MPLS encodes a path as a stack of labels inserted in the packet header by the ingress node. This overhead may violate the Maximum SID Depth (MSD), the equipment hardware limitation which indicates the maximum number of labels an ingress node can push onto the packet header. Currently, the MSD varies from 3 to 5 depending on the equipment manufacturer. Therefore, the MSD value considerably limits the number of paths that can be implemented with SR-MPLS, leading to an inefficient network resource utilization and possibly to congestion. We propose and analyze SR-LEA, an algorithm for an efficient path label encoding that takes advantage of the existing IGP shortest paths in the network. The output of SR-LEA is the minimum label stack to express SR-MPLS paths according to the MSD constraint. Therefore, SR-LEA substantially slackens the impact of MSD and restores the path diversity that MSD forbids in the network.**

***Index Terms*—Segment Routing, MPLS, traffic engineering.**

I. Introduction

Segment Routing (SR) is a new architecture standardized by IETF SPRING working group [1]. It can be instantiated over two existing data plane MPLS (SR-MPLS) [2] and IPv6 (SR-IPv6). In SR packet are forwarded using the source routing mechanism: the path the packet has to go through is encoded in its header. SR-MPLS is the central focus of the IETF working groups, mainly because of the important implications of service providers (SPs). The major advantage of SR is to eliminate the per-flow states from the SP's core routers. In fact, a path is directly usable by any router; no prior setup/signalization is required, unlike MPLS-TE where a tunnel has to be signaled and maintained using protocols such as the Resource Reservation Protocol Traffic Engineering (RSVP-TE). In SR, only the ingress node has to maintain per-flow states. Plus, SR architecture extends already deployed IGP protocols (OSPF, IS-IS and BGP Link State) to exchange SR information, revoking the need for a label distribution protocol such as LDP or RSVP-TE for SR-MPLS.

A SR Path (SRP) is encoded as list of segments identifiers (SIDs), each SID associated with a data plane forwarding instruction *e.g.*, forward the packet down the IGP shortest path or forward to a specific exit interface.

978-1-5090-3216-7/16/$31.00

In SR-MPLS, a SID is represented by a 20-bit label. The SID is processed using the three standard MPLS operations POP, PUSH, and SWAP. A SRP is encoded as a stack of labels that the ingress router pushes onto the packet header. In fact, pushing more than one was supported since the early version of MPLS standards, the label stack has been used for multiple use cases: hierarchical tunnels, Layer 2 Virtual Private Network (L2VPN), and Layer 3 VPN. However, those use cases require a small number of labels, for example, a scenario of L2VPN or L3VPN requires only simultaneously two labels: the tunnel's label and VPN's label. To take full advantage of SR's potential, a router has to be able to push a larger number of labels. Unfortunately, current hardware suffers from physical limitation of the number of labels that can be used simultaneously [3].

In fact, in order to achieve wire-speed packet processing, hardware vendors use Application-specific integrated circuit (ASIC)s. They are designed to perform specific tasks very efficiently compared to general purpose processors. Consequently, they are limited in the size and the type of the operations they can perform. For example, the PUSH operation is implemented using dedicated ASICs that limit the number of labels they can push onto the packet header, this limitation in SR is known as the Maximum SID Depth (MSD). Therefore, an efficient label encoding able to reduce the labels stack size is essential to alleviate the MSD impact. In addition, reducing the label stack saves space and enables to carry other types of labels.

In this paper, we propose two label encoding algorithms for SR-MPLS paths. Both algorithms compute the minimum number of labels to express a SRP. We evaluate their performances over several real-world network topologies. The results are presented in term of the average number of labels to express a set network paths. In addition, we study their efficiency in alleviating the impact of the MSD limitation.

II. Related works

In [4], Giorgetti *et al.* propose two SRP encoding algorithms that produce two label stacks of the equal size for the same SRP. Both algorithms use only Node-SIDs to encode a SRP. Unfortunately, in some cases, the resulting label stack may not correspond to the initial path. In fact, the proposed algorithms work well in a network where the shortest path between two

neighbors is their direct common link, however in reality, a network administrator may attribute higher costs to particular links resulting in the direct link not being the shortest path between two nodes. In [5], the network graph is augmented by inserting a virtual link for every pair of nodes in addition to the existing physical links. The virtual links represent the Equal-Cost Multiple Paths (ECMP)s between two nodes. The path computation is performed over this new graph. The proposed label encoding algorithm replaces a virtual link by the tail's end node Node-SID whilst the physical link is replaced by an Adj-SID. This proposition suffers mainly from scalability issues due to the size of the new graph (a 1000 nodes graph results in a new graph with approximately half million links).

III. Segment Identification (SID)

A SRP can be encoded using any combination of SIDs (*i.e.*, local or global) as long as the nodes that the packet traverses own a forwarding instruction to reach the egress node. In SR-MPLS, a node advertises a Segment Routing Global Bloc (SRGB). The SRGB is the range of labels allocated for SR (*e.g.*, [1000, 2000]). A global SID takes its value within the SRGB (*e.g.*, 1100), all the SR nodes install a forwarding instruction associated with each global SIDs. A local SID takes its value outside the SRGB (*e.g.*, 3000); it is advertised in the SR domain but only the node advertising it possesses an associated forwarding instruction. In this work, we focus on two SIDs types: Node-SID and Adj-SID, we do not consider other SID types such as service SID, BGP peering SIDs, etc.

We propose two SRP encoding algorithms to produce a label stack composed of two SID types: Node-SID and Adj-SID. Each SID has a pre-installed forwarding plane instruction associated with it. A**Node-SID** is a label associated with the SR node *i.e.*, attached to the loopback address. When a SR node receives a packet with a Node-SID as a top label, it forwards it on the IGP shortest path to reach the node that owns that Node-SID, the node owning the Node-SID pops the label before inspecting the next label in the stack. An **Adjacency SID** (Adj-SID) is the label attached to an IGP adjacency *i.e.*, the interface to reach the neighbor router. It is used to enforce packet forwarding through a specific exit interface. By default, an Adj-SID is advertised as a local segment, it can also be advertised as global if desired.

IV. Segment Routing path Encoding

The SRP length varies depending on the network diameter, QoS requirements, and network resources availability. Accordingly, the label stack to express a SRP can be very big, resulting in a label stack size that may exceed the ingress MSD. This not only inhibits the use of such paths, but also the use of other label types. Therefore, an efficient encoding algorithm is required to minimize the size of the label stack.

A source routed path is said to be strict if all the links and nodes the packet will go through are listed in the its header. On the contrary, a path is said to be loose if its header contains only a subset of the links and nodes the packet will pass through. SRPs can be expressed exclusively with Node-SIDs, local Adj-SIDs, Global Adj-SIDs or a combination of those SID types. In this paper, a SRP is strict if it is encoded using only Adj-SIDs, loose otherwise. A SRP encoded with local Adj-SID is a strict path, because the Adj-SIDs are local to the nodes advertising them have the forwarding entries associated with them. Therefore, the packet has to go through only the nodes that own the Adj-SIDs. A SRP encoded exclusively with Node-SID or a combination of Node-SIDs and Adj-SIDs is a loose path. Two successive Node-SIDs in the label stack can be separated by one or more network nodes. The label stack expresses the initial path in the current state of the network. However, if the IGP metric between two SR nodes changes, the label stack will not represent the initial path anymore. A SRP encoded with global Adj-SIDs can be either strict or loose: it is strict if all the links that the packet has to go through are listed in the label stack, loose otherwise.

We propose two SRP path encoding algorithms that computes label stack as a combination of Node-SIDs and Adj-SIDs. In current SR deployments, Adj-SIDs are advertised as local segments. In that case, we use the SR paths Label Encoding Algorithm (SR-LEA) to compute the minimum label stack. However, as stated in the standards, Adj-SIDs can also be advertised as global segments. Therefore, the minimum label stack is computed using the SR-LEA-A Algorithm.

Let us consider the topology detailed in Fig. 1. All the nodes allocate the same SRGB: [1000, 2000]. The computed path to satisfy the Quality of Service (QoS) requirements for the traffic sent by CE1 to CE2 is *P*: $PE1 \rightarrow P2 \rightarrow P3 \rightarrow P7 \rightarrow P6 \rightarrow PE5$, encoded and pushed as a stack of labels onto flow's packets by $PE1$.

A. Strict Encoding

A strict encoding of the SRP is the worst case scenario, as it generates the maximum label stack to encode a SRP. Two approaches may be applied. The first one uses exclusively Node-SIDs to encode a SRP by replacing each node in the SRP by its Node-SID. This approach suffers from the same problem as in [4] and is valid only if the shortest path is expressed by direct links between all the neighbors in the path. For example, a strict encoding of path *P* results in the following label stack: {*Node-SID PE1, Node-SID P2, Node-SID P3, Node-SID P7, Node-SID P6, Node-SID PE5*}. This label stack does not express path *P*. The packets at $P3$ will not be sent to $P7$ over the direct link because it is not the shortest path. The second one uses exclusively Adj-SIDs to encode a SRP. At each node the exit interface is replaced with the associated Adj-SID, this produces a label stack that corresponds to requested path. As shown in Fig. 2, a strict encoding of path *P* results in the following label stack: {*Adj-SID PE1-P2, Adj-SID P2-P3, Adj-SID P3-P7, Adj-SID P7-P6, Adj-SID P6-PE5*} $= [5012, 5023, 5037, 5076, 5065]$. Each node pops the Adj-SID that it owns before forwarding the packet through the chosen interface.

Strict encoding can be essential to accomplish certain tasks such as Operations, Administration, and Maintenance (OAM). However, a SP network can be composed of hundreds or even

thousands of nodes. Using strict encoding especially for long paths is not always possible as it violates the MSD constraint, also it adds a considerable overhead to packets.

B. SR-LEA Algorithm

We propose the SR-LEA algorithm to compute the minimum label stack to encode a SRP when the Adj-SID are advertised as local segments. Its input is the initial path expressed as a list of IP addresses manually or computed by a centralized entity such as a Software Defined Network (SDN) controller [6] or by a Path Computation Element (PCE)[7]. SR-LEA makes use of existing IGP shortest paths, which are installed as forwarding instructions by the SR-MPLS control plane. The resulting label stack is a combination of Node-SIDs and local Adj-SIDs. It represents exactly the initially computed path in the current state of the network.

SR-LEA has two main steps detailed by the pseudocode in Algorithm 1. First, the SRP is spliced into a succession of shortest paths (subpaths), the container A holds the SRP splices, whereas B will hold the potential SRP splice. Second, each subpath composed of three or more nodes is replaced by its tail's end node Node-SID, whilst each two nodes subpath is replaced by the Adj-SID between those two nodes. The best case is that the requested SRP follows the shortest path SPF. Consequently, SR-LEA outputs a label stack composed of one label: the egress node's Node-SID.

In order to encode the path *P* using SR-LEA, we follow the two steps of the algorithm. First, the subpaths that compose the path *P* are computed and saved in A: $\{(PE1, P2, P3), (P3, P7), (P7, P6, PE5)\}$. Finally each subpath in A is replaced with the appropriate SID. $\{PE1, P2, P3\}$ is composed of three nodes, it is replaced by $P3$'s Node-SID (1003). $(P3, P7)$ is composed of two nodes and is replaced by the Adj-SID $P3$-$P7$ (5037). $\{P7, P6, PE5\}$ is composed of three nodes. Therefore, it is replaced by $PE5$'s Node-SID (1005).

The resulting label stack is [1003, 5037, 1005]. As shown in Fig. 3, a packet follows the IGP shortest path to reach $P3$ using label 1003 (*i.e.*, P3's Node-SID). At P3, the Adj-SID 5037 is used to enforce the packet through the link *P3-P7*. At $P7$, label 1005 (*i.e.*, PE5's Node-SID) is used to forward the packet down the IGP shortest path to reach $PE5$. At PE5, label 1005 is popped and the IP packet is forwarded to $CE2$.

C. SR-LEA-A

In the segment routing architecture, it is possible to advertise an adjacency (*i.e.*, an interface) as a global segment, rather than advertising it as a local segment. Accordingly, the adjacency becomes routable in the SR domain. In comparison to the local Adj-SID, all the SR nodes forward the packet using the IGP shortest path to reach the node that advertises the global Adj-SID, then the node that owns the adjacency forwards the packet to the exit interface associated with the global Adj-SID. To take advantage of this possibility, we propose SR-LEA with global Adj-SIDs (SR-LEA-A). When Adj-SIDs are advertised as global segments it is the SR-LEA-A that computes the

Algorithm 1 Efficient Label Encoding algorithm

INPUT: The SRP expressed as a list of IP addresses
OUTPUT: labelStack the SRP minimum label stack.
Initialization:
G: Graph of the network topology
$A = \{\ \}$: Holds the list of the SRP subpaths.
$B = [\ ]$: Used to construct a subpath, when no IP addresses can be added it is moved to A.
$SPF = Dijkstra(SRP[1],\ SRP[end])$: The shortest path between the source and destination of the SRP.
$labelStack = [\ ]$

STEP 1: Computation of the SRP subpaths.

$i = 1$: Points to the current node of the SRP.
$k = length(SRP)$: Points to the last node of the candidate subpath.
while $i <= length(SRP)$ **do**
  $push(B,\ SRP[i])$
  if $i == length(SRP)$ **then**
    $push(A,\ B)$
  else if $B \nsubseteq SPF$ **then**
    if $length(B) == 2$ **then**
      if $k > i$ **then**
        $k--$
        $B = B[1]$
        $SPF = Dijkstra(G,\ B[1],\ SRP[k])$
        $continue$
      else
        $push(A,\ B)$
        $B = B[end]$
        $SPF = Djikstra(G,\ B[1],\ SRP[k])$
      end if
    else
      $push(A,\ B[1 : end - 1])$
      $SPF = Djikstra(G,\ B[end - 1],\ SRP[k])$
      $B = [\ ]$
      $i--$
      $continue$
    end if
  end if
  $i++$
  $k = length(SRP)$
end while

STEP 2: The construction of the label stack.

for $i \leftarrow 1$ To $Size(A)$ **do**
  if $length(A[i]) > 2$ **then**
    $push(labelStack,\ NodeSID(A[i][end]))$
  else
    $push(labelStack,\ AdjSID(A[i]))$
  end if
end for

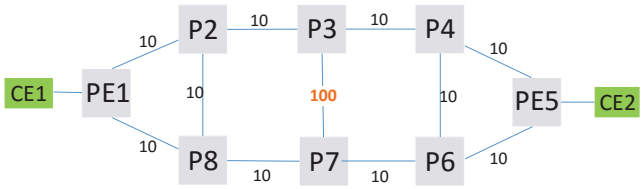


Fig. 1: Reference network topology, all the links costs is 10 except the link P3-P7 is attributed cost 100.

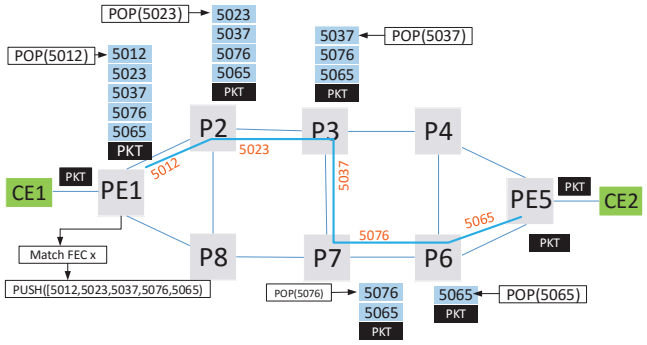


Fig. 2: The SRP to connect CE1 and CE2 is expressed as a label stack using the strict algorithm.

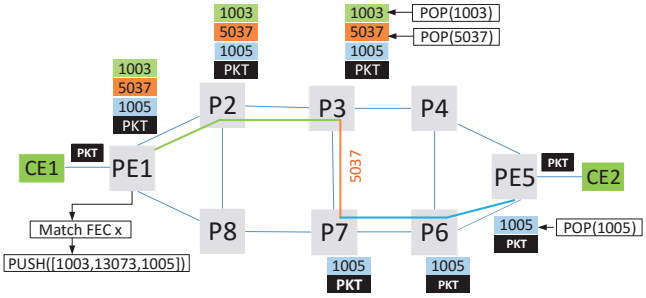


Fig. 3: The SRP to connect CE1 and CE2 is expressed as a label stack computed using the SR-LEA algorithm.

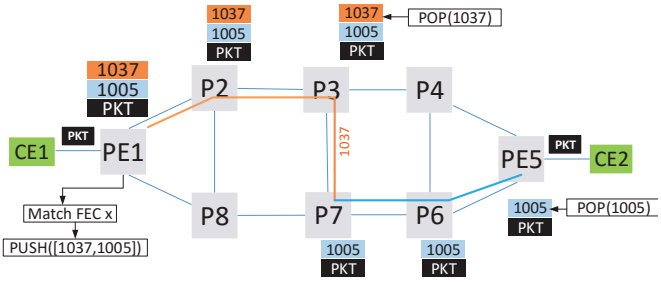


Fig. 4: The SRP to connect CE1 and CE2 is expressed as a label stack computed using the SR-LEA-A algorithm.

minimum label stack. In SR-LEA-A, we suppose that the Adj-SIDs are advertised as global segments, the resulting label stack is either smaller or equal to the SR-LEA's one. Both algorithms share step 1 detailed in Algorithm 1. In SR-LEA-A, as detailed by the pseudocode in Algorithm 2: a subpath of $size \geqslant 3$ followed by one of $size = 2$ are encoded using one label: the global Adj-SID between the last node in the first path and the first node in the second one. Compared to SR-LEA, two labels are used to encode the two subpaths.

Algorithm 2 Efficient Label Encoding with global Adj-SIDs

STEP 1 Same as for SR-LEA

STEP 2

for $i \leftarrow 1$ To $Size(A)$ **do**
  if $length(A[i]) > 2$ **then**
    if $length(A[i+1]) == 2$ **then**
      $push(labelStack, GlobalAdjSID($
      $A[i][end],\ A[i+1][1]))$
      $p+ = 2$
      $continue$
    end if
    $push(labelStack, NodeSID(A[i][end]))$
  else
    $push(labelStack, AdjSID(A[i]))$
  end if
end for

In the example described in Fig. 4, P3 advertises its adjacency with P7 as the global SID 1037, the list A contains the following subpaths: $\{(PE1, P2, P3), (P3, P7), (P7, P6, PE5)\}$. Accordingly, the two subpaths $\{(PE1, P2, P3), (P3, P7)\}$ are encoded using the global Adj-SID $P3 - P7 : 1037$. Consequently, the label stack for the path P is [1037, 1004]. At PE1 and P2, based on 1037 the packet is forwarded down the shortest path to reach P3. At P3, the top label 1037 is popped and the packet forwarded through the interface that connects P3 to P7. At P7, based on the $PE5$'s Node-SID (*i.e.*, 1005) the packet is forwarded through the shortest path to reach $PE5$.

V. Simulation Results

In order to better evaluate the performance of the proposed algorithms, we experimented on several SNDlib network topologies [8]. To get a representative set of paths, for each topology, we consider a sample bandwidth demand matrix D. As detailed in Table I, we solve the multicommodity flow problem.The result is the optimal set of paths to satisfy the demand matrix. The paths are then encoded using the strict Adj-SID, SR-LEA and SR-LEA-A.

Both of our algorithms compute the minimum label stack to express a SRP, SR-LEA when the Adj-SIDs are local segments and SR-LEA-A when they are global. The comparison is made between the strict encoding, the SR-LEA and the SR-LEA-A algorithms. For each topology, using the three encoding

Topology	# vertices	# edges	# demands
Geant	22	36	431
Albilene	12	18	131
Brain	161	166	9045
Germany50	50	80	1270
Nobel-germany	17	26	248

TABLE I: Parameters for each topology

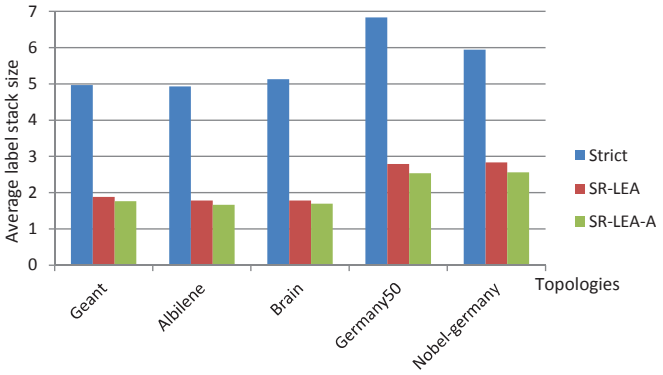


Fig. 5: Comparison of the average label stack size generated using a strict encoding, SR-LEA and SR-LEA-A algorithms.

algorithms detailed previously, we compute the average label stack size and the percentage of network paths encoded with a label stack $size \leq MSD$.

Fig. 5 illustrates the per-topology average label stack size variation depending on the topology and the encoding algorithm. We observe that the strict encoding always produces a large label stack. This was expected because no optimization on the label stack size is performed, rather a one to one mapping of the physical links to the label stack. We note that for some paths the label stack reaches up to 14 labels. SR-LEA reduces the size of the label stack by 52% to 65% compared to the strict encoding; the observed gain varies depending on the network design and diameter. SR-LEA-A gives the best results. Notably, compared to the strict encoding, the average label stack size is reduced by 57% to 67%.

The MSD is a local characteristic of a router, it varies from one equipment vendor to another. In an architecture where the path computation is delegated by the SR node to a centralized entity such as a SDN controller or a PCE [3]. This limitation makes long paths in the network unusable and forces the network traffic to follow only short paths which cause inefficient traffic distribution or worse: network congestion. For this study, we fixed the MSD to 5 labels, which is the value announced currently by the major equipment vendors.

Fig. 6, illustrates the variation of the percentage of the useable paths in each topology. With a strict encoding, the percentage of useable paths can be very low *e.g.*, 37% for $Germany50$ topology. Using SR-LEA, increases considerably the amount of useable paths *e.g.*, from 37% to 97% for $Germany50$ topology. However, encoding the label stack using SR-LEA-A gives the best results, as it increases the number of usable paths from 37% to 99%, a gain of 2% to 4% more than SR-LEA. We expect the difference to be more

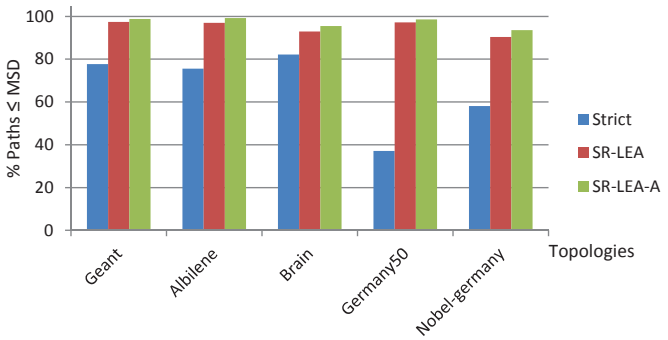


Fig. 6: Paths with a label stack size $\leq MSD$ $(MSD = 5)$.

considerable on topologies with bigger diameters.

The proposed algorithms are very efficient to reduce the label stack size, but also they limit considerably the impact of the MSD limitation. However, they do not eliminate the MSD problem, as we still have paths that can not be expressed with a label stack smaller than the MSD.

VI. CONCLUSION

In this work, we proposed two SR-MPLS paths label encoding algorithms, namely SR-LEA and SR-LEA-A. Both algorithms compute the minimum label stack to express a segment routing path. Their performance has been evaluated over real topologies. In addition, we prove that they are efficient in alleviating the impact of the MSD. For future work, a PCE implementation of the proposed algorithms is underdevelopment. We are considering the possibility to use the two algorithms to encode Topology Independent Loop-Free Alternate (TI-LFA) Fast Reroute post-convergence paths.

REFERENCES

[1] C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," Internet Engineering Task Force, Internet-Draft draft-ietf-spring-segment-routing-09, Jul. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-spring-segment-routing-09

[2] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[3] S. Sivabalan, J. Medved, C. Filsfils, V. Lopez, J. Tantsura, W. Henderickx, E. Crabbe, and J. Hardwick, "PCEP Extensions for Segment Routing," Internet Engineering Task Force, Internet-Draft draft-ietf-pce-segment-routing-07, Mar. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-pce-segment-routing-07

[4] A. Giorgetti, P. Castoldi, F. Cugini, J. Nijhof, F. Lazzeri, and G. Bruno, "Path encoding in segment routing," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[5] F. Lazzeri, G. Bruno, J. Nijhof, A. Giorgetti, and P. Castoldi, "Efficient label encoding in segment-routing enabled optical networks," in *Optical Network Design and Modeling (ONDM), 2015 International Conference on*. IEEE, 2015, pp. 34–38.

[6] A. Sgambelluri, A. Giorgetti, F. Cugini, G. Bruno, F. Lazzeri, and P. Castoldi, "First demonstration of sdn-based segment routing in multi-layer networks," in *Optical Fiber Communications Conference and Exhibition (OFC), 2015*. IEEE, 2015, pp. 1–3.

[7] A. Sgambelluri, F. Paolucci, A. Giorgetti, F. Cugini, and P. Castoldi, "Sdn and pce implementations for segment routing," in *Networks and Optical Communications-(NOC), 2015 20th European Conference on*. IEEE, 2015, pp. 1–4.

[8] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010. [Online]. Available: http://www3.interscience.wiley.com/journal/122653325/abstract

Named Data Networking for Tactical Communication Environments

M. Tamer Refaei, Sean Ha, Zac Cavallero, Creighton Hager
The MITRE Corporation

***Abstract*—Tactical communication environments are characterized by frequent disruptions, intermittent connectivity, and low bandwidth (DIL) due to the unfavorable, unfriendly, and sometimes very dynamic network conditions. Nevertheless, there is still an expectation that tactical applications can communicate seamlessly despite the challenging communication environment. In this paper, we propose and evaluate the use of Named Data Networking (NDN) in tactical communication environments. NDN has inherent functionalities that makes it more robust in a DIL environment in comparison to IP. In this work, we consider a notional tactical environment and compare network performance under IP versus NDN. In our evaluation, we consider different levels of network disruption and different data dissemination mechanisms. Our results show that NDN can provide significant improvement gains in network performance that is superior to IP, under similar network conditions.**

I. Introduction

The IP communication model has been shown to perform poorly in DIL environments, like Military tactical networks [1]. By design, IP assumes a conversational communication model, which requires the creation and maintenance of end-to-end routes between senders and receivers. In a DIL environment, these assumptions do not hold. End-to-end connectivity may not exist at all or may not exist long enough to form reliable end-to-end routes. In this paper, we consider the use of a data-centric communication model for DIL environments. In particular, we use Named Data Networking (NDN) [2]. NDN is a proposed architecture for the next generation Internet that shifts the network focus away from addressing and onto data.

NDN's objective is to move away from IP's host-centric model and towards a more dynamic and intelligent distribution network. This is accomplished through in-network caching, data naming, stateful forwarding and securing content. These built in functionalities can enhance network efficiency and robustness in DIL environments. Caching in NDN is a built-in network service (as opposed to an overlay service in Content Distribution Networks). Data requested by a consumer can be delivered from the closest cache as opposed to from its original producer. This brings content closer to the consumer, improves network performance, and enables robustness in the face of disruption. Moreover, NDN looks at separating data from its physical network location by assigning unique names to data. A client requests a specific piece of data by name instead of specifying a content provider address where the data resides. This eliminates the need to maintain end-to-end routes, which can be a significant source of overhead in DIL environments (considering the low-bandwidth and frequently disrupted links). Since NDN is data centric, the routing and forwarding functionalities in NDN are performed on data names rather than host-specific IP addresses. Forwarding in NDN is stateful, which enables NDN to perform much more intelligent forwarding decisions than IP (which is stateless). This functionality can be used to enable access control (only allow data with certain names to be forwarded), intelligent caching (cache and forward data based on its priority, which can be name-driven), as well as robustness to disruption (cache data based on known delay/disruption on the incoming link). Finally, NDN makes security a built-in service by having producers of data sign their data. This eliminates the need for securing channels, ensures that data cannot be forged, and securely attributes data to its source.

978-1-5090-3216-7/16/$31.00

In this paper, we consider the use of NDN in a DIL environment represented in a notional tactical communication environment. We compare the performance of IP to that of NDN under different levels of network disruption and using different types of data dissemination models (unicast and multicast). We show how NDN can provide significant robustness against network disruption in comparison to IP. We also show how NDN can mix the unicast and multicast data dissemination models to achieve localized robustness to disruption.

This paper is organized as follows. In section II, we provide an overview of NDN and a summary of related work. In section III, we describe our network environment, traffic model, and evaluation metrics. We discuss the results of our evaluation in section IV and the impact of the size of NDN's cache store on network performance. We conclude in section V.

II. Named Data Networks

NDN is a relatively new approach to disseminating data across a network. It provides an architectural approach to requesting data across a network without specifying a fixed network endpoint. There are three main network components in an NDN environment: producers, consumers, and forwarders. Producers of data objects are responsible for giving each a unique name that identifies it. A consumer requests any data object by its given name. Forwarders forward requests and data objects between consumers and producers. A data object can be cached within the network while being forwarded. This enables subsequent requests for the same data to be served from the network rather than the data's producer.

NDN implements two types of packets: *interest packets* and *data packets*. Consumers request data by name by sending *interest packets*. An *interest packet* includes the name of the data object requested and possibly some information for filtering against the data to match. A forwarder that receives an *interest packet*, and has a corresponding data object cached, can respond with a *data packet* that includes this data object. If not, it forwards the *interest packet* to other NDN forwarder(s) that may know how to get the data. If the data is not cached anywhere, the *interest packet* will eventually reach its producer, which generates a *data packet* in response. The *data packet* is routed along the reverse route traversed by the *interest packet*. Each forwarder along the path may cache the *data packet* in its local store. Caching allows subsequent requests of the same data object to be served from caches rather than by the data producer.

There has been a number of research efforts on resilient communication for DIL environments. Some approaches attempted to adapt IP protocols and services to disruption. A number of research efforts leveraged overlay protocols such as Disruption Tolerant Networking (DTN) [3], which is specifically designed to handle network disruption. There were also some efforts that explored the use of Information Centric Networks in DIL environments [4] [5] [6]. An NDN-based architecture was introduced for vehicular ad-hoc networks (VANETs) in [7], which shares some of the characteristics of DIL environments. Our primary focus in this paper is to evaluate NDN in particular in a DIL environment.

III. NDN vs. IP in Disrupted Environments

In this section, we compare NDN's performance against that of IP. We discuss the characteristics of the evaluation network environment in the following subsections.

A. Network Components

We utilized the notional tactical communication network that is shown in Figure 1. We emulated this network using CORE (Common Open Research Emulator) [8]. The topology includes the following network components:

- Two shore nodes and two ships (shown inside squares) that function as consumers of information.
- Five High Mobility Multipurpose Wheeled Vehicle (HMMWV) that function as producers. The vehicles will produce traffic and send it to the shore nodes.
- Two relay ships deployed close to the producers. They will function as static relays for traffic.
- Two UAVs that move in a circular path (shown as a dashed circle) to provide connectivity between the HMMWVs and the ships (and eventually, an end-to-end path between consumers and producers).

B. Network Connectivity

In the network environment used, the shore nodes and the relay ships are interconnected by a satellite. The data rate on all links to the satellite is 512Kbps and the delay is 250ms. To account for the environmental/adversarial effects typically experienced in tactical networks, the links from the satellite to the ship relay nodes have a 10% loss. The UAVs provide disrupted end-to-end connectivity between the HMMWVs and the ship relay nodes through their circular mobility path shown in Figure 1.

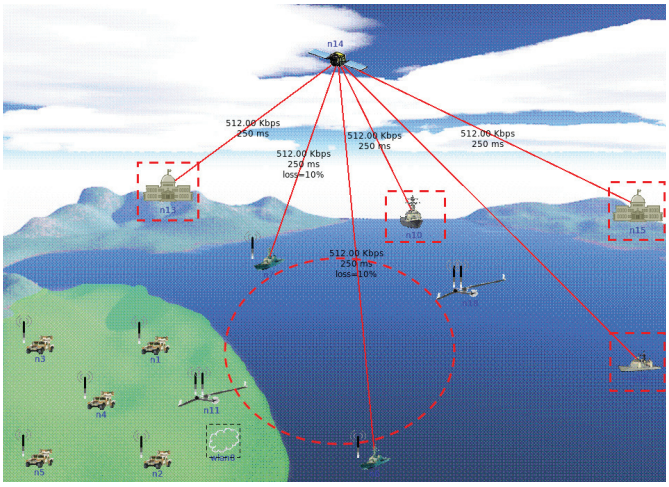


Fig. 1: Network Topology

To realize different levels of disruption, we use 2 different UAV mobility models.

- *Slightly Disrupted:* The UAVs are reliably connected to the relay ships but their layer 1 and 2 connectivity to the HMMWVs is disrupted 10% of the time. Note that the layer 3 connectivity is expected to be worse due to the time needed for routing to converge.
- *Moderately Disrupted:* The connectivity between the UAVs and both the relay ships and the HMMWVs is disrupted. Layer 1 and 2 connectivity between the relay ships and the HMMWVs through the UAVs is disrupted 70% of the time. Layer 3 end-to-end routes between consumers and producers never converge in this model.

The consumers were set up to have the satellite as their gateway. The rest of the nodes in the network run the Optimized Link State Routing (OLSR) [9] protocol. For multicast scenarios, the Simplified Multicast Forwarding (SMF) mechanism was used for forwarding [10].

C. Traffic Model

For the IP environments, we use Advanced Trivial File Transfer Protocol (ATFTP). Each HMMWV runs an ATFTP server that hosts 30 files (a total of 150 files), 15KBytes each. Each shore node runs an ATFTP client and attempts to get as many of the 150 files from the HMMWVs as possible in random order within 30 minutes. The ATFTP data chunk size was set to 1400 bytes.

For the NDN environments, we use the same 150 files but utilized NDN for retrieval. Each file is named uniquely as */ndn/node-name/file-number*. The NDN chunk size was set to 1400 bytes (i.e. 11 chunks per file).

For both the IP and the NDN environments, we utilized two data dissemination models:

TABLE I: Data Dissemination Configurations

	Model		Traffic	
Scenario	IP	NDN	Unicast	Multicast
SC.1	X		X	
SC.2	X			X
SC.3		X	X	
SC.4		X		X

- *Unicast:* In the IP environment, all ATFTP traffic is sent as unicast traffic. In the NDN environment, we utilized unicast transports (i.e. faces) between all the NDN routers.
- *Multicast:* Using multicast for dissemination of data eliminates the dependency on the state of layer 3 routing. In the IP environment, all ATFTP traffic is sent as multicast traffic. In the NDN environment, layer 2 and multicast transports were utilized in the disrupted portions of the network (between the relay nodes, UAVs, and the HMMWVs).

The list of all scenarios considered is shown in Table I.

D. Evaluation Metrics

We utilized 3 metrics to assess and compare the performance of all scenarios considered:

- *Delivery Delay:* This metric measures the delay of retrieving files from producers at the consumers.
- *Delivery Ratio:* This metric assesses the ratio of files that were received successfully at the consumers.
- *Link Utilization:* This metric assesses how much data was sent over some of the challenged/critical links within the network. This includes the links from the satellite to the relay ships (*Sat-R1, Sat-R2*), from the relay ships to the UAVs (*R1-UAV, R2-UAV*), and from the UAVs to the HMMWVs (*UAV1-HMV, UAV2-HMV*). We ignored both NDN *interest packets* and ATFTP control traffic in the assessment of link utilization since both result in roughly the same amount of traffic.

We ran each scenario 30 times, with each run having a duration of 30 minutes. All of the results discussed will be based on averages over the 30 runs.

IV. Evaluation

In this section, we introduce the results of our evaluation from the two scenarios discussed in section III-B.

A. Slightly Disrupted

In this connectivity environment, we only considered SC.1 and SC.3 since the network is relatively stable compared to the moderate mobility model. In SC.1, the delivery ratio was limited to 30%. The disruption in layer 1 and 2 was significant enough to reduce the ability of OLSR to establish stable end-to-end routes. End-to-end routes were possible about 5% of the time, which had a significant impact on the operation of ATFTP. SC.3 on the other hand had a 99.4% delivery ratio. NDN's in-network caching allows for serving data that was previously retrieved and cached, which improved robustness to disruption.

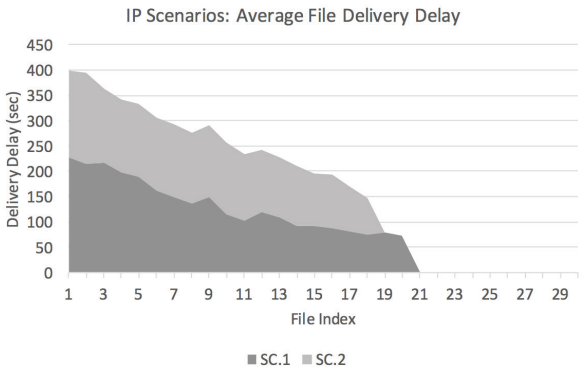


Fig. 2: SC.1 and SC.2 Delivery Delay

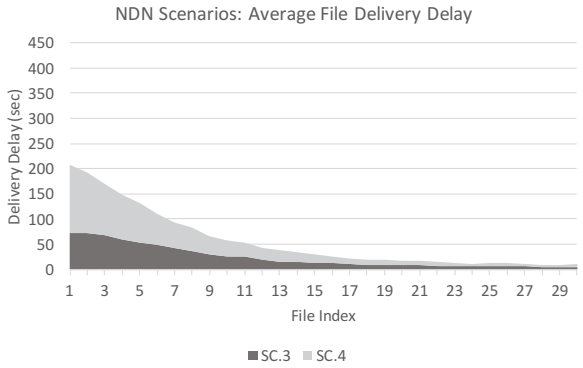


Fig. 3: SC.3 and SC.4 Delivery Delay

The delivery delay for SC.1 is shown in Figure 2. Each series corresponds to the average delivery delay of files as they get delivered at the consumers. Note that the series are shown as a stacked area graph and that for files that were never retrieved (within the 30 minute bound) the graph did not show any value (e.g. files with indices higher than 21). The figure shows an average of 135 seconds delivery delay (63 second standard deviation). The delivery delay for SC.3 (shown in Figure 3) averaged 23 seconds (22 second standard deviation). A good portion on the files (about $\frac{1}{3}$) were delivered from caches, which significantly reduced delivery delay. In fact, the delivery delay for the first $\frac{1}{3}$ of the files averaged 50 seconds with an 18 second standard deviation. For the second $\frac{1}{3}$, it dropped to 14 seconds with a 5 second standard deviation. For the final $\frac{1}{3}$, it was 6 seconds with a 1 second standard deviation.

B. Moderately Disrupted

In this connectivity environment, we only considered SC.2 and SC.4 since layer routes failed to converge. The delivery ratio was 30% for SC.2 and about 88% for SC.4. The average delay for SC.2 was 137 seconds, while SC.4 scored 33 seconds. Once again, the disrupted network was able to benefit from NDN's in-network caching. In addition, NDN's data

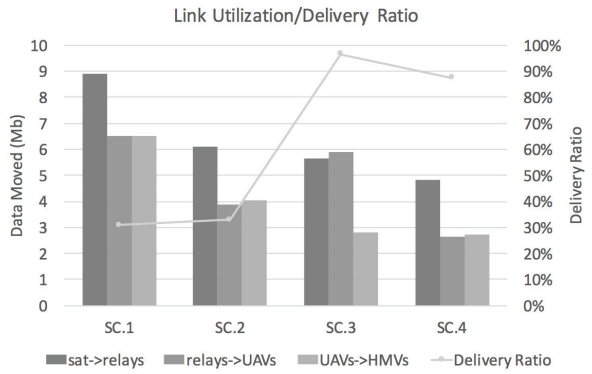


Fig. 4: Link Utilization/File Delivery Ratio

TABLE II: Impact of Cache Size

$\frac{CacheSize}{TotalDataSize}$	Avg. Delivery Delay (sec)	File Delivery Ratio
>100%	23	99.6%
50%	46	99.4 %
25%	64	80 %

centric model allowed for using unicast between NDN nodes in the relatively stable portion of the network (between shore, satellite, and ship relays) while using multicast in the disrupted portion (ship relays, UAV, and HMMWV nodes). This reduced link utilization as shown in Figure 4 without compromising delivery ratio.

Finally, Figure 4 shows the link utilization on some of the challenged links together with file delivery ratios. It is clear from the figure that using NDN reduces the amount of data moved across links (by fetching data from caches) without compromising file delivery ratio, which reflects NDN's efficiency. In the unicast scenarios, NDN resulted in reducing the data moved on the links between the satellite and the relay nodes by about 35%. In the multicast scenario, that reduction was about 40%. The delivery ratio in both was more than 88%, more than double the delivery ratio in the IP environments.

C. Impact of Cache Size

In the scenarios considered so far, we used a cache size of about 500MB, which is much larger than the total data generated (about 2.25MB). To assess the impact of the cache size on NDN's performance, we re-ran SC.3 with smaller cache sizes: half of the total data that is generated and a quarter of the data generated. The results (shown in Table II) show that even with the cache store size set to 25% of the total generated data, NDN still beats IP delivering 80% of all files within 30 minutes with an average delivery delay of 64 seconds per file.

V. Conclusion and Future Work

In this work, we evaluated the use of NDN in tactical communication networks. The in-network caching functionality of NDN can provide significant robustness against disruption in an incremental manner. This can be done by using unicast for data dissemination in the stable portions of the network and selectively utilizing multicast data dissemination for more disrupted portions of the network. In the future, we plan to explore the use of Software Defined Networks with NDN for tactical network environments where mobility is predictable or scheduled. In these environments, a priori knowledge of disruptions can be leveraged synchronously between NDN and SDN to improve resiliency to disruption.

Acknowledgment

This work was sponsored by the MITRE Technology Program as a MITRE Sponsored Research (MSR) Project.

References

[1] K. Scott et al. "Robust communications for disconnected, intermittent, low-bandwidth (DIL) environments". In: *Military Communications Conference, 2011 - MILCOM 2011*. Nov. 2011, pp. 1009–1014. DOI: 10.1109/MILCOM.2011.6127428.

[2] Lixia Zhang et al. "Named Data Networking". In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 66–73. ISSN: 0146-4833. DOI: 10.1145/2656877.2656887. URL: http://doi.acm.org/10.1145/2656877.2656887.

[3] Kevin Fall. "A Delay-tolerant Network Architecture for Challenged Internets". In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '03. Karlsruhe, Germany: ACM, 2003, pp. 27–34. ISBN: 1-58113-735-4. DOI: 10.1145/863955.863960. URL: http://doi.acm.org/10.1145/863955.863960.

[4] J. Seedorf et al. "The Benefit of Information Centric Networking for Enabling Communications in Disaster Scenarios". In: *2015 IEEE Globecom Workshops (GC Wkshps)*. Dec. 2015, pp. 1–7. DOI: 10.1109/GLOCOMW.2015.7414086.

[5] You Lu et al. "Information-centric delay-tolerant mobile ad-hoc networks". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE. 2014, pp. 428–433.

[6] S. Y. Oh, D. Lau, and M. Gerla. "Content Centric Networking in tactical and emergency MANETs". In: *Wireless Days (WD), 2010 IFIP*. Oct. 2010, pp. 1–5. DOI: 10.1109/WD.2010.5657708.

[7] Giulio Grassi et al. "ACM HotMobile 2013 poster: vehicular inter-networking via named data". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 17.3 (2013), pp. 23–24.

[8] *Common Open Research Emulator (CORE)*. URL: http://www.nrl.navy.mil/itd/ncs/products/core.

[9] *Optimized Link State Routing*. URL: https://tools.ietf.org/html/rfc7181.

[10] *Simplified Multicast Forwarding*. URL: https://tools.ietf.org/html/rfc6621.

Reducing the Latency-Tail of Short-Lived Flows: Adding Forward Error Correction in Data Centers

Klaus-Tycho Foerster, Demian Jaeger, David Stolz, and Roger Wattenhofer
ETH Zurich, Switzerland
{foklaus, jaegerde, stolzda, wattenhofer}@ethz.ch

***Abstract*—TCP handles packet loss in the network by retransmitting lost packets, which in turn increases latency. Many connections in data centers are short-lived and consist only of a few packets (e.g., RPCs). Such connections suffer disproportionately from packet retransmissions. We address this issue by introducing a new transport layer protocol called ATP: ATP uses ample forward error correction at the beginning of a connection, allowing short-lived flows to recover from packet loss without retransmissions – but at the same time not congesting long-lived flows. Our experiments show that in an environment with background traffic, the latency's 99th percentile can be reduced by a factor of almost 20 while being fair to other TCP connections.**

***Index Terms*—FEC, Latency Tail, TCP, Data Centers**

I. Introduction

When Microsoft Research examined the traffic of 1500 servers in a cluster for data mining [1], they found that while more than 50 % of all the flows did not last for more than 100 ms, they did not contribute to more than 1 % of the transferred data. Many flows were transmitting at a small rate, 50 % at 10 kB/s or less (cf. also [2], [3]). Even though such little flows do not contribute significantly to the overall traffic volume, they are important and especially affected by a lost packet. The added latency of an additional RTT due to the retransmission is proportionately larger for a small flow than a large one. The authors of [1] state: "*We do believe that TCP's inability to recover from even a few packet drops without resorting to timeouts in low bandwidth delay product settings is a fundamental problem that needs to be solved.*" [1]

Similarly, a review of Facebook's data center network architecture showed that a single HTTP request can result in dozens of cache and database lookups and almost 400 remote procedure calls (RPCs) [4]. We assume that the vast amount of small flows discovered is at least partially caused by RPCs.

As thus, we aim to reduce the latency of *small* flows within data centers by introducing a new transport layer protocol called *Another Transport Protocol (ATP)*. Our goal is to reduce the latency of small flows that is caused by packet loss. While TCP *always* needs to perform a retransmission in the case of a packet loss and the data transfer thus takes an additional Round-trip Time (RTT), our protocol uses Forward Error Correction (FEC) in order to avoid this latency.

Organisation of our short paper. We start by describing the design of ATP in Section II, before providing a positive performance evaluation of ATP in Section III. We then discuss related work in Section IV, and lastly conclude in Section V.

978-1-5090-3216-7/16/$31.00

II. Design of ATP: Another Transport Protocol

In this section, we will describe the two most important features of ATP, packet loss handling and congestion control.

A. ATP's Packet Loss Handling

ATP has three different measures to handle lost packets:

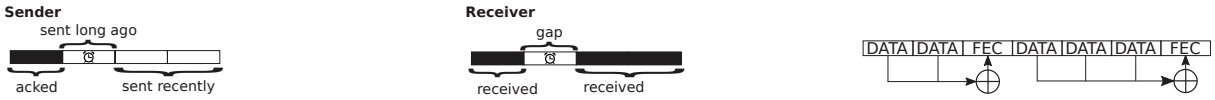
Sender Timeouts. A timeout on the sender side occurs if a byte of the stream does not get acknowledged within a specified time. The timeout duration depends on the RTT. If a byte times out, it will be retransmitted by the sender. The protocol tries to avoid these timeouts when possible, as such timeouts are comparatively large and so is the added latency.

Receiver Timeouts. If the receiver receives out-of-order packets, the received data stream will have gaps. If these gaps exist for too long, they will trigger a timeout and the receiver sends a retransmission request to the sender which then retransmits the missing data. Subfigures 1a and 1b illustrate the two different types of timeouts.

Forward Error Correction. In order to avoid timeouts even in the case of packet loss, the protocol uses a simple systematic block coding. Systematic codes contain the input data in the output. This leads to no decoding overhead if all data is received correctly. After sending multiple packets with plain data, the protocol will insert a FEC packet which contains the XOR of the previously sent data packets as illustrated in Subigure 1c. If one of these packets is lost, the receiver can restore the missing packet. These additional packets result in an overhead leading to higher bandwidth requirement for the same goodput. To minimize the overhead, the rate of the FEC is adjusted by the protocol. At the start of a connection the rate will be high, but it is decreased as the transfer speeds up and there is no loss.

B. Congestion Control

An important feature of a reliable transport layer protocol is its congestion control algorithm. If there is congestion in the network, the protocol must adjust its send rate. The basic principle is illustrated in Figure 2 and is the same for ATP and TCP. The window size is the amount of unacknowledged

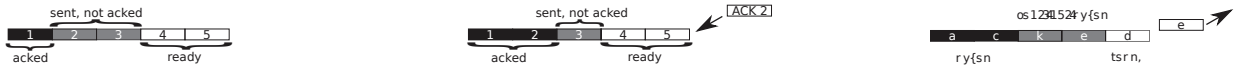


(a) If a packet does not get acknowledged within a certain time period, it will trigger a timeout and the sender will send it again.

(b) If a gap in the receiver stream persists for a specified timeout period, the receiver will ask the sender to resend the packet.

(c) The FEC packets contain the bitwise XOR of the plain data packets sent before.

Fig. 1. Timeouts can either happen at the sender or the receiver.



(a) Packet 1 is already acknowledged, while 2 and 3 are not. Even though the sender has additional data in its buffer, it is not allowed to send it.

(b) The sender now receives an acknowledgment for segment 2. It now has only one unacknowledged packet, while its window size is 2.

(c) The sender now sends packet 4, as there is still space in the window. After sending the fourth packet, all allowed packets have been sent.

Fig. 2. In this example the window size is the length of two packets. The sender is only allowed to send two packets and must wait until an acknowledgment of a packet is received before the next packet can be sent.

bytes the sender is allowed to transmit. Adjustment of this window allows control of the transmission speed.

ATP's Congestion Control. Our protocol increases its window based on a RTT without any congestion events. If during a whole RTT no such event occurs and at least some data gets acknowledged, the window will be increased by a constant value corresponding to the maximum packet size. Since this increase might not be fast enough for links with high latency, at the start of a new connection the window is increased with each newly acknowledged segment. However, in data centers the RTT is usually very small, and an increase of the RTT is most probably due to increased buffer latency. Since Another Transport Protocol (ATP) then increases its window size slower, the buffer will not fill up too fast. In the case of a lost packet, there are, as mentioned in Section II-A, three possible ways to get the data to the receiver: Either by using the FEC information, by sending a retransmission request to the sender, or if the sender experiences a timeout. If the receiver needs to use a FEC packet due to a lost packet, it will inform the sender about this event – the sender then decreases its window by a small factor. If the receiver sends a retransmission request or the sender has a local timeout, it will decrease its window by a larger factor.

As the window is increased, the protocol will simultaneously reduce the FEC rate – since the network is not in a congested state with losses. Conversely, if the window size is decreased, the FEC rate is increased.

We note that the specific implementation details of ATP (in *C*) are omitted in this short paper due to space constraints. The main difference of ATP to TCP is the included forward error correction, which is our central focus in this short paper.

III. Performance Evaluation

In this section, we first describe our testbed setup, before comparing the performance of ATP and TCP for small flows under various background traffic settings in Subsection III-A. We analyze a large number of connections, showing that ATP indeed reduces the tail latency of small flows. Lastly, we also briefly show the fairness of ATP to TCP in Subsection III-B.

Testbed Setup. We evaluated the implementation of ATP on a small testbed using up to four laptops and a single desktop machine, depicted in Figure 3. All machines ran on Ubuntu 15.10 and had a Gigabit Ethernet interface. In each experiment, the Ethernet pause frame functionality was disabled. When comparing TCP with ATP, all the hardware helpers of the NIC to support TCP were disabled.

ATP relies on a mechanism called *Random Early Detection (RED)*, which is a widely available in high performance switches used in data centers. RED detects if a queue in a switch gets larger and starts to drop packets before it is completely full. This results in a fairer packet drop distribution across multiple flows, and does usually not result in a drop of consecutive packets – an essential property for ATP. The testbed's switches (*Planet GSD-805*) do not support RED, hence the desktop machine is used to provide RED.

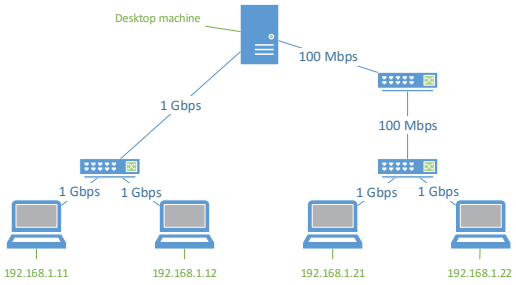
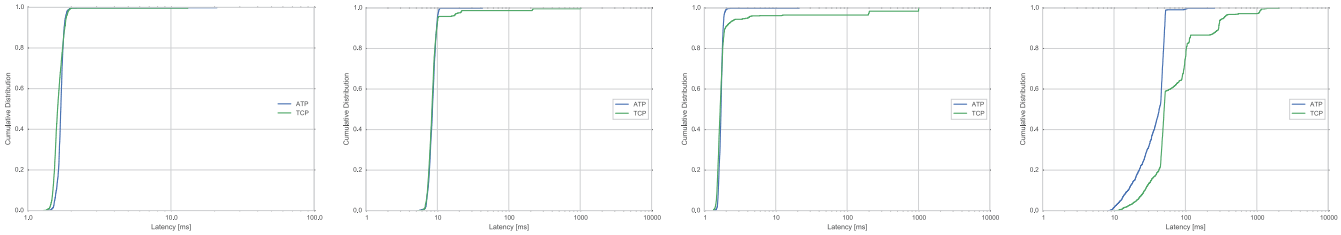


Fig. 3. Depiction of the testbed we used for our experiments. One 100 Mbps link is added between all tested connection for congestion control effects.

A. Comparison of ATP and TCP

The main goal of our design of ATP is to reduce the latency of small flows, in the presence of larger flows. In this subsection, we compare several small ATP connections to TCP on different background traffic scenarios. We will see that, especially in a slightly congested network, ATP often can omit retransmissions and thus additional latency.

For each background traffic scenario, 1000 connections were subsequently completed, each transferring 8.5 kB data from 192.168.1.11 to 192.168.1.21. Background traffic was sent permanently from 192.168.1.12 to 192.168.1.22. For each connection the time for its completion was measured. Note that for TCP connections, the time for the three-way handshake



(a) No other traffic in the network.

(b) Background traffic of 192.168.1.12 permanently sending a large stream of TCP data to 192.168.1.22.

(c) Link loss of 1 % on the Linux machine acting as a switch. The loss was introduced only in the direction in which the data was sent, and not in the acknowledgment direction.

(d) Background traffic of 192.168.1.12 permanently sending a large stream of UDP data to 192.168.1.22; Of all the UDP packets sent, 1.5 % were dropped in the network.

Fig. 4. Comparison of the cumulative distribution functions of the connection completion times of ATP and TCP in various settings of background traffic. ATP is depicted in blue, TCP in green. In these experiments, the sender sent a data stream of 8.5 kB 1000 times from 192.168.1.11 to 192.168.1.21 in the network depicted in Figure 3. While TCP slightly outperforms ATP in cases of no background traffic, the latency tail of the completion times grows considerably with heavy background traffic and data loss, letting ATP complete considerably faster than TCP. E.g., already in Subfigure 4c, ATP finishes a bit after 10ms, while TCP takes about 1000ms to complete for the last flows.

was subtracted from the measured time, in order to have a direct comparison to ATP, which does not perform such a handshake.

No Background Traffic. In the first experiment, there is no background traffic at all. The results shown in Subfigure 4a demonstrate that TCP and ATP behave very alike. The median and the mean of the flow duration is slightly lower for TCP than for ATP, which is expected since ATP transmits 2 additional `FEC` packets for each connection.

TCP Background Traffic. In the second experiment, there is background traffic originating from a large TCP stream. The results are displayed in Subfigure 4b. Since TCP implements a congestion control algorithm, the network will not be excessively congested. Nevertheless, due to the small buffer size set in the desktop machine (acting as a switch), some connections experience packet loss. ATP can reconstruct the stream on the receiver side and – apart from one single connection in the whole sample – does not need to retransmit data at all. This leads to a significantly lower 99th percentile of the flow duration. TCP's 99th percentile is at 210 ms while ATP's is at 11 ms. However, TCP's 95th percentile is only 2 % higher.

Lossy Link In Subfigure 4c the desktop machine introduces a loss rate of 1 %. In such a case ATP will usually be able to reconstruct the lost packets immediately. The 99th percentile is at 2 ms, which is the same as in the case of no loss at all. However, the 99th percentile of TCP is at over *one second*! The reason for this is mainly due to the problem TCP has when the last data packet is lost. The sender waits until a timeout occurs. ATP's last packet sent is a `FEC` packet, and the second last is the stream's last data packet. As long as both of these last packets are not lost in the network, the worst that can happen is a receiver timeout, which is significantly faster than a sender timeout.

UDP Background Traffic The massive injection of UDP packets into the network with a speed slightly above the maximum link capacity congests the network heavily. It results in a network that experiences 1.5 % packet loss on average. Additionally, all buffers are full, which makes retransmissions more expensive. In contrast to experiments with TCP background traffic, where the 95th percentile did not differ extremely between TCP and ATP, the 95th percentile in this experiment differs vastly: While with ATP 95 % of all flows finish within 50 ms, TCP needs 340 ms. The 99th percentile is at 53 ms for ATP and at 1134 ms for TCP. These results are shown in Subfigure 4d.

Tail Latency Subfigures 4a to 4d show the cumulative distribution of the latencies in different environments. Since the tails of the distributions are not clearly visible, Figure 5 shows the last percentiles. We see again, that in case there is congestion, or if packet loss occurs in the network, ATP can reduce the tail latency successfully.

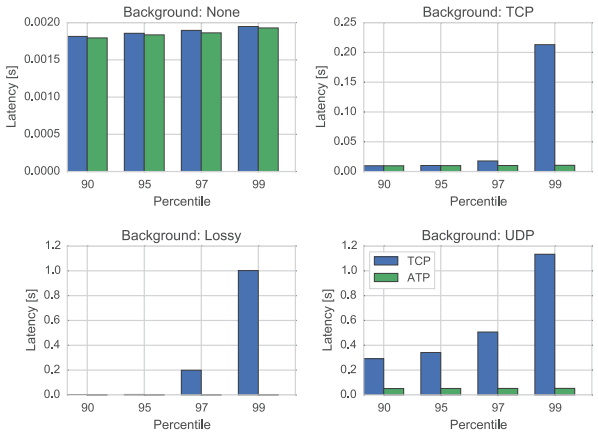


Fig. 5. Tail latencies of ATP and TCP compared in different background traffic scenarios. The different scenarios are described in Subsection III-A.

B. Fairness of Concurrent Connections

An important feature of a transport layer protocol is that it does not starve other connections. If two connections are started in parallel, both connections should use approximately the same bandwidth. By its similarity in design to TCP, ATP is fair to both ATP and TCP – which we also evaluated experimentally in our network testbed.

Due to to space constraints in this short paper, we just briefly review one experiment for TCP: In Figure 6, a TCP and an

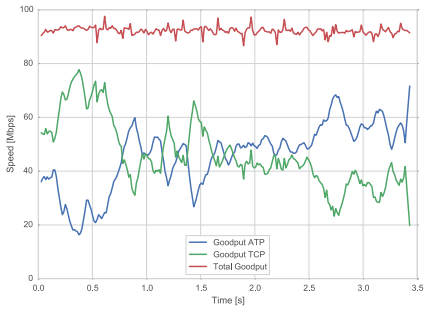


Fig. 6. An ATP and a TCP connection transfer 20 MB of data each. ATP from 192.168.1.11 to 192.168.1.21, TCP from 192.168.1.12 to 192.168.1.22.

ATP stream are started at the same time and both must transmit 20 MB of data. Looking at the whole connection duration the link is shared in a fair way: the TCP stream finishes after 3.43s, while it takes ATP 1.18 % longer to complete.

IV. Related Work

Adding forward error correction (FEC) at the link layer is a well studied concept in wireless networks, see, e.g., [5]. More closely related to our work is applying FEC on a packet level basis. The authors of [6], following an idea of [7], evaluated the effect of applying FEC on the IP and TCP interface by running simulations. If their new *intermediate layer* detects the loss of the next segment of the TCP stream, it waits some time for the arrival of an error correction packet and will rebuild the missing IP packet. Used between two layers, the transport layer protocol can not directly benefit from the additional information provided by the applied FEC[1]. Additionally, their solution adds latency, since the new intermediate layer waits for potential correction packets. This and other [8] early proposals focused on lossy links, rather than on data centers.

Recently, multiple projects have attempted to reduce the latency of flows within a data center. The HULL architecture [9] trades 10 % of the links' bandwidth for reduced latency. If the traffic load on a link arrives at 90 % of the total capacity, a NetFPGA sets the Explicit Congestion Notification (ECN) flag and TCP will slow down. This avoids the filling of the switch's queues and will thus reduce latency. Fastpass [10] lets each sender delegate control to a centralized arbiter which decides when a packet has to be sent and which paths it should take. This results in a single point of failure, or at least a non trivial handover from the primary to the secondary controller. Both of these suggestions need adaption of the intermediate network infrastructure itself, while our approach has the advantage of only requiring software adjustments at the endpoints.

Lastly, for a recent discussion of the impact of the latency tail, also beyond a data center setting, we refer to [11].

V. Conclusion and Outlook

We designed and implemented a transport layer protocol called ATP, which provides the same functionalities as TCP: A reliable transfer of a data stream within an IP network without congesting the network, and with fairness to other ATP, as well as TCP streams. The design goal was to reduce the latency tail of small flows within data centers.

The protocol uses a simple systematic block coding. After sending a variable number of data packets, a FEC packet is sent, containing the XOR of the previously sent data packets. This allows the immediate reconstruction of the stream, if the network dropped a packet and thus reduces the need for retransmissions. ATP uses a sliding window mechanism to control its send rate. Whenever a packet loss occurs, the send rate is decreased and the same time the FEC rate is increased.

In a small testbed, we evaluated the performance of ATP and verified that it behaves fairly to other connections. In a series of multiple small data transfers, ATP often outperforms TCP, since it is able to avoid retransmissions. In an environment which has TCP background traffic, ATP's 99th percentile of the flow completion time is at 11 ms, while TCP's is at 210 ms. The additional FEC information induces a small overhead on the network's load, but the latency decrease can improve the user experience. ATP needs no central controller, or changes to the intermediate hardware, and only relies on software adjustments on the end-hosts.

A next step of using packet based FEC information to shrink the latency of small flows would be to include ATP's functionality as a TCP extension. This would guarantee backward compatibility with hosts that do not support ATP. It is furthermore possible to augment ATP's implementation by a priority-aware congestion control algorithm.

[1]This can be addressed by using Explicit Congestion Notification (ECN) messages, as suggested by the authors.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments. Klaus-Tycho Foerster was partially supported by Microsoft Research.

References

[1] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Internet measurement conference*. ACM, 2009.

[2] A. Feldmann, J. Rexford, and R. Cáceres, "Efficient policies for carrying web traffic over flow-switched networks," *IEEE/ACM Trans. Netw.*, vol. 6, no. 6, pp. 673–685, Dec. 1998.

[3] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 50–61, 2011.

[4] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *IEEE Optical Interconnects Conf.* IEEE, 2013.

[5] B. Liu, D. Goeckel, and D. Towsley, "Tcp-cognizant adaptive forward error correction in wireless networks," in *GLOBECOM*. IEEE, 2002.

[6] H. Lundqvist and G. Karlsson, "TCP with end-to-end FEC," in *Communications, 2004 International Zurich Seminar on*. IEEE, 2004.

[7] C. Huitema, "The case for packet level fec," in *TC6 WG6.1/6.4 Fifth International Workshop on Protocols for High-Speed Networks V*, 1997.

[8] O. Tickoo, V. Subramanian, S. Kalyanaraman, and K. K. Ramakrishnan, "LT-TCP: end-to-end framework to improve TCP performance over networks with lossy channels," in *IWQoS*, 2005.

[9] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: trading a little bandwidth for ultra-low latency in the data center," in *NSDI*, 2012.

[10] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 307–318, 2015.

[11] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

The Cardinality-Constrained Paths Problem: Multicast Data Routing in Heterogeneous Communication Networks

Alvaro Velasquez[1], Piotr Wojciechowski[2], K. Subramani[3], Steven L. Drager[4], Sumit Kumar Jha[5]
Department of Computer Science, University of Central Florida, Orlando, FL[1,5]
LCSEE, West Virginia University, Morgantown, WV[2,3]
Information Directorate, Air Force Research Laboratory, Rome, NY[4]
{velasquez, jha}@cs.ucf.edu, pwojciec@mix.wvu.edu, ksmani@csee.wvu.edu, steven.drager@us.af.mil

***Abstract*—In this paper, we present two new problems and a theoretical framework that can be used to route information in heterogeneous communication networks. These problems are the cardinality-constrained and interval-constrained paths problems and they consist of finding paths in a network such that cardinality constraints on the number of nodes belonging to different sets of labels are satisfied. We propose a novel algorithm for finding said paths and demonstrate the effectiveness of our approach on networks of various sizes.**

I. Introduction

One big challenge in securing today's communication networks lies in their dynamic nature, with nodes constantly connecting to and disconnecting from other nodes. This is the case with mobile devices and increasingly so with vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) [5] communication, whose safety is a significant research focus of the United States Department of Transportation [7]. In these vehicular ad-hoc networks (VANETs), road-side units (RSUs) can send information to vehicles' on-board units (OBUs), which can then route information to other OBUs as a multi-hop network [1]. The topology of vehicle-to-vehicle and vehicle-to-infrastructure networks changes quickly due to the high speeds of moving vehicles, which causes nodes to connect to and disconnect from the VANET frequently [8]. The size of these VANETs can also be quite large in areas of heavy traffic and dense populations [9]. These challenges necessitate efficient routing algorithms.

In this paper, we solve the problem of finding paths in a heterogeneous network where there are upper-bound constraints on the number of nodes of each type that can be traversed. We call this the cardinality-constrained paths problem. We also explore the case where there are lower and upper bounds on the cardinality constraints. We call this the interval-constrained paths problem. Such constraints arise naturally in delay-tolerant networks (DTNs) where there is intermittent connectivity and interference or faults in the data are likely [10]. This is the case in VANETs and many ad-hoc networks. For dealing with these precarious domains, carry-and-forward routing schemes are often used [3]. These protocols store the message in intermediate nodes which then forward the message when a connection is available. The intermediate nodes can also act as verification nodes which ensure the integrity of the message being sent. Thus, placing a lower bound on the number of intermediate vertices is sensible for trustworthy communication.

We make the following contributions:

- The problems of finding cardinality-constrained paths (CCP) and interval-constrained paths (ICP) are introduced. These problems entail finding a path between two nodes in a network such that constraints on the number of nodes belonging to each label are satisfied. The CCP problem deals with upper bound constraints while the ICP problem pertains to upper and lower bounds.
- We establish the complexity of the CCP and ICP problems. more specifically, we show that they belong to the **NP-Hard** complexity class.
- For the CCP problem, we propose an efficient solution whose runtime is polynomial in the number of

Alvaro Velasquez acknowledges support from the National Science Foundation Graduate Research Fellowship Program (GRFP). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Air Force Research Laboratory or the National Science Foundation. This research was supported in part by the National Science Foundation through Award CCF-1305054. This work was supported by the Air Force Research Laboratory under US Air Force contract FA8750-16-3-6003. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. [4] Approved for public release: distribution unlimited, case 88ABW-2016-4285. Cleared August 31, 2016.

978-1-5090-3216-7/16/31.00

nodes in the network when the number of labels is fixed. For the ICP problem, a similar polynomial-time algorithm is proposed when the network of interest is a directed acyclic graph (DAG).

II. Problem Definition and Complexity

In order to define the cardinality-constrained path problem, we need to formalize the notion of labeled graphs. In untrusted communication networks, this may simply consist of a graph $G = (V, E)$, where V is the set of vertices and E is the graph's adjacency matrix, such that every vertex $v \in V$ belongs to either some set of trusted or authenticated nodes L_T or a set of untrusted nodes L_U. The general case can be formalized as follows. A labeled graph $G = (V, E, \{L_1, L_2, \ldots, L_p\}), |V| = n$, is a graph whose nodes belong to one of p disjoint labels $L_1, L_2, \ldots, L_p \subseteq V, L_i \cap L_j = \emptyset$, such that $L_1 \cup L_2 \cup \cdots \cup L_p = V$. A path $\pi^{s \to t} = \{s, v_{i_1}, v_{i_2}, \ldots, v_{i_k}, t\} \subseteq V$ from source node $s \in V$ to destination node $t \in V$ is an ordered set defined by the vertices in the path such that $(\pi_i^{s \to t}, \pi_{i+1}^{s \to t}) \in E$. We are interested in the number of nodes in the path that belong to each label. Let $L_i^\pi = \pi^{s \to t} \cap L_i$ denote the set of nodes belonging to the path that have label i.

In practice, these labels can be determined in many ways depending on the domain of the network. For network security purposes, authenticated nodes and compromised nodes could be two labels, with all other hosts belonging to a third label. In the vehicular domain, on-board units (OBUs) and road-side units (RSUs) have their respective labels, which could be augmented by adding labels to nodes that are vulnerable to malfunctions via structural damage or malicious hacking. There can also be labels denoting the performance of a node in terms of processing power or its reliability in terms of inherent fault-tolerance. In wireless sensor networks (WSNs) where resources are scarce [6], labels could be used to categorize different levels energy consumption or storage capacity.

Cardinality-Constrained Paths Problem

***Definition** 1:* (Cardinality-Constrained Path) Given integers $k_1, k_2, \ldots, k_p$, a labeled graph $G = (V, E, \{L_1, \ldots, L_p\})$, and source and destination nodes $s, t \in V$, the cardinality-constrained path problem consists of finding a path $\pi^{s \to t} = \{s, v_{i_1}, v_{i_2}, \ldots, v_{i_k}, t\}$ such that $|L_1^\pi| \leq k_1, |L_2^\pi| \leq k_2, \ldots, |L_p^\pi| \leq k_p$.

We show, by reduction from 3SAT, that the cardinality-constrained path problem is **NP-Hard**. Consider an instance Φ of 3SAT with n variables and m clauses. For each variable x_i in Φ, let k_{2i} be the number of clauses in which the literal x_i appears. Similarly, let k_{2i+1} be the number of clauses in which the literal $\neg x_i$ appears. Associated with the variable x_i, we create gadget VG_i shown in Figure 1.

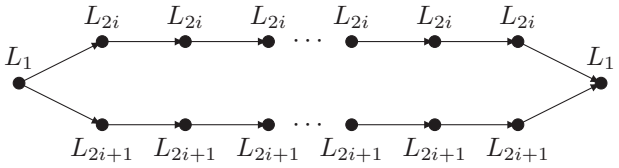


Fig. 1. Gadget VG_i for variable x_i. This gadget is a labeled graph $G = (V, E, \{L_1, L_{2i}, L_{2i+1}\})$

In this gadget, the number of vertices along the top path is k_{2i}. Each of these vertices is assigned the label L_{2i}. Likewise, the number of vertices along the bottom path is k_{2i+1}. Each of these vertices is assigned the label L_{2i+1}.

For each clause ϕ_j in Φ, we create the gadget CG_j shown in Figure 2 such that the labels of the three intermediate vertices depend on the literals in the clause. If x_i is in ϕ_j, then an intermediate vertex is assigned label L_{2i}. Conversely, if $\neg x_i$ is in ϕ_j, then an intermediate vertex is assigned label L_{2i+1}.

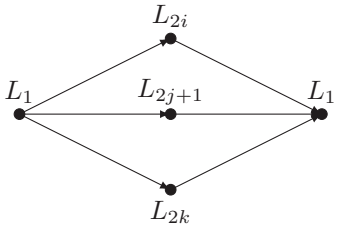


Fig. 2. Gadget for clause $(x_i, \neg x_j, x_k)$. This gadget is a labeled graph $G = (V, E, \{L_1, L_{2i}, L_{2j+1}, L_{2k}\})$.

We construct G by combining the gadgets as shown in Figure 3. Note that in G, there are $(m + n + 1)$ vertices labeled L_1. Also note that all these vertices must appear on any path from s to t. Thus, we set $k_1 = (m + n + 1)$ to ensure that this limit is always satisfied. Now we need to show that there is a path of the desired type in G if and only if Φ is satisfiable.

***Theorem** 1:* Cardinality-Constrained Path is **NP-Hard**

Proof: Using the construction above, we will show that Φ has a satisfying assignment if and only if there is a path $\pi^{s \to t}$ in G such that $|L_1^\pi| \leq k_1, \ldots, |L_p^\pi| \leq k_p$.

($\implies$) Suppose Φ is satisfiable. For each variable x_i assigned a value of **true** in the satisfying assignment, the path will traverse the bottom path of gadget VG_i. This uses k_{2i+1} vertices of label L_{2i+1}. Similarly, for each variable x_i assigned a value of **false**, the path will traverse the top path of VG_i. This uses k_{2i} vertices of label L_{2i}.

Fig. 3. Complete graph $G = (V, E, \{L_1, \ldots, L_{2i+1}, \ldots, L_{2n}, L_{2n+1}\})$ of the construction for reducing 3SAT to a cardinality-constrained path.

For each clause ϕ_j, the path will traverse an intermediate vertex of CG_j corresponding to a **true** literal. Let us focus on the variable x_i. If x_i is assigned a value of **true**, then the following conditions hold.

- We traverse k_{2i+1} vertices of label L_{2i+1} in VG_i.
- We do not traverse any vertices of label L_{2i+1} in the clause gadgets since $\neg x_i$ has a value of **false**.
- We traverse at most k_{2i} vertices of label L_{2i} since the literal x_i appears in only k_{2i} clauses.

Thus, the satisfying instance of Φ yields a path $\pi^{s\to t}$ with $|L_1^\pi| \leq k_1, \ldots, |L_{2i}^\pi| \leq k_{2i}, |L_{2i+1}^\pi| \leq k_{2i+1}, \ldots, |L_{2n}^\pi| \leq k_{2n}, |L_{2n+1}^\pi| \leq k_{2n+1}$.

($\Longleftarrow$) Assume G has a path $\pi^{s\to t}$ with $|L_1^\pi| \leq k_1, \ldots, |L_p^\pi| \leq k_p$. For each variable x_i in Φ, assign a value of **true** if $\pi^{s\to t}$ traverses the bottom of gadget VG_i. If $\pi^{s\to t}$ traverses the top of gadget VG_i, then assign a value of **false** to x_i.

For each clause ϕ_j in Φ, the following hold.

- If $\pi^{s\to t}$ traverses the intermediate vertex of CG_j corresponding to the literal x_i, then $\pi^{s\to t}$ cannot traverse the top path of gadget VG_i. Otherwise, $\pi^{s\to t}$ would contain $k_{2i}+1$ vertices of label L_{2i}, which violates the constraint $|L_{2i}| \leq k_{2i}$. Thus, x_i must be assigned a value of **true** and clause ϕ_j is satisfied.
- If $\pi^{s\to t}$ traverses the intermediate vertex of CG_j corresponding to the literal $\neg x_i$, then $\pi^{s\to t}$ cannot traverse the bottom path of gadget VG_i. Otherwise, $\pi^{s\to t}$ would contain $k_{2i+1}+1$ vertices of label $L_{2\cdot i+1}$, which violates the constraint $|L_{2i+1}| \leq k_{2i+1}$. Thus, x_i must be assigned a value of **false** and clause ϕ_j is satisfied.

It follows that a path $\pi^{s\to t}$ with $|L_1^\pi| \leq k_1, \ldots, |L_p^\pi| \leq k_p$ yields a satisfying assignment to Φ. □

Interval-Constrained Paths Problem

***Definition** 2:* (Interval-Constrained Path) Given integer pairs $(k_1, k_1'), (k_2, k_2'), \ldots, (k_p, k_p')$, a labeled graph $G = (V, E, \{L_1, \ldots, L_p\})$, and source and destination nodes $s, t \in V$, the interval-constrained path problem consists of finding a path $\pi^{s\to t} = \{s, v_{i_1}, v_{i_2}, \ldots, v_{i_k}, t\}$ such that $k_1 \leq |L_1^\pi| \leq k_1', k_2 \leq |L_2^\pi| \leq k_2', \ldots, k_p \leq |L_p^\pi| \leq k_p'$.

We demonstrate that this problem is **NP-Hard** in the simplest case where there is only a single label, i.e. $p = 1$. Naturally, the problem is at least as hard for $p > 1$.

***Theorem** 2:* Interval-Constrained Path is **NP-Hard**

Proof: We will prove this result through a reduction from the Hamiltonian path problem. Let $G = (V, E)$ be a directed graph with n vertices and m edges, and let s and t be vertices in $\mathbf{V}$. We construct $G' = (V', E', \{L_1\})$ as shown in Figure 4. Assign the label L_1 to all vertices in G', and set $k_1 = n$ and $k_1' = \lceil \frac{3n}{2} \rceil$. Then there is a Hamiltonian path in G if and only if there is a path $\pi^{s\to t}$ in G' such that $k_1 \leq |L_1^\pi| \leq k_1'$.

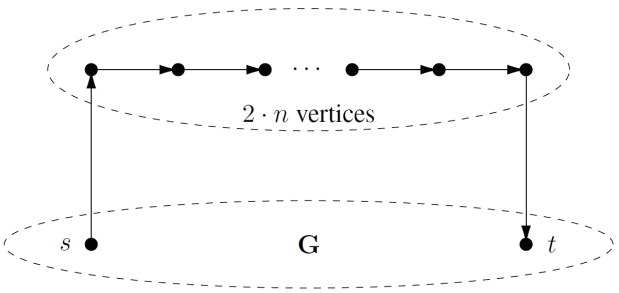


Fig. 4. Graph $G' = (V', E', \{L_1\})$ used in our construction.

($\Longrightarrow$) Assume that G has a Hamiltonian path $\pi^{s\to t}$ from s to t. Let π' be the corresponding path in G'. We have that $n \leq |L_1^{\pi'}| \leq \lceil \frac{3n}{2} \rceil$. Thus, G' satisfies the interval-constrained path problem.

($\Longleftarrow$) Assume that G' has a path $\pi^{s\to t}$ from s to t such that $k_1 \leq |L_1^\pi| \leq k_1'$. Note that the top path in G' consists of $(2 + 2n) > \lceil \frac{3\cdot n}{2} \rceil$ vertices. This path would violate constraint $|L_1^\pi| \leq k_1'$. Thus, $\pi^{s\to t}$ must consist of only vertices originally from G. Since $|L_1^\pi| \geq n$, $\pi^{s\to t}$ must use all n vertices in V. Thus, G must have a Hamiltonian path from s to t.

Since the Hamiltonian path problem is **NP-complete**, the interval-constrained path problem is **NP-hard**. Note that G' has $3n$ vertices. Thus the bounds chosen n and $\lceil \frac{3n}{2} \rceil$ are non-degenerate. □

III. Methodology

In this section, we define the dynamic programming procedures used to solve the cardinality-constrained paths (CCP) and the interval-constrained paths (ICP) problems. In terms of complexity, the main difference between the two problems is that CCP is fixed-parameter tractable when the number of labels is fixed. Meanwhile, we have shown that ICP remains **NP-Hard** even with this fixed parameter. Consequently, we propose a polynomial-time algorithm for solving CCP in the general case for a fixed label size. The extension proposed for ICP is also solvable in polynomial time, but is only applicable to directed acyclic graphs.

$$u_1^0(t_1, t_2, \ldots, t_{p-1}) = \begin{cases} 1 & \mathcal{I}_p(v_1) \\ 0 & \bigvee_{i=1}^{p-1} ((t_i \geq 1) \wedge (\mathcal{I}_i(v_1) = 1)) \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

$$u_j^0(t_1, t_2, \ldots, t_{p-1}) = \infty \qquad j = 2, 3, \ldots, |V|, t_i = 0, 1, \ldots, k_i \quad (2)$$

$$u_j^m(t_1, \ldots, t_{p-1}) = \min \left\{ u_j^{m-1}(t_1, \ldots, t_{p-1}), \min_{k:(v_k, v_j) \in E} \left\{ u_k^{m-1}\left(t_1 - \mathcal{I}_1(v_j), \ldots, t_{p-1} - \mathcal{I}_{p-1}(v_j)\right) + \mathcal{I}_p(v_j) \right\} \right\} \quad (3)$$

Cardinality-Constrained Paths

We have devised an algorithm that is exponential only in the number of labels p and polynomial in the number of vertices $|V|$ in the graph. Thus, our algorithm is tractable for a fixed number of labels. To the best of our knowledge, this is the first such algorithm to tackle this problem. Our proposed dynamic programming algorithm uses equations (1), (2), and (3) to populate the dynamic programming matrix, which is then used to infer the solution path through a standard backtrack search.

Let $u_j^m(t_1, t_2, \ldots, t_{p-1})$ denote the minimal number of p-labeled nodes in a path from the source node v_1 to v_j such that the number of k-labeled nodes is less than or equal to $t_k, 1 \leq k \leq p-1$, and the path consists of at most m edges. Then we can define a dynamic programming recurrence as shown in equations (1), (2), and (3), where $\mathcal{I}_k(v_i)$ is 1 if $v_i \in L_k$ and 0 otherwise. The initialization equation (1) states that the number of p-labeled nodes from the source node v_1 to itself using 0 edges is 1 if $v_1 \in L_p$, 0 if $v_1 \in L_i, i \neq p$ and its corresponding constraint parameter t_i is greater than or equal to 1, and ∞ otherwise. Similarly, the number of p-labeled nodes from v_1 to any other vertex v_j using 0 edges is initialized to ∞ (2) since no such path can exist. The recurrence (3) minimizes the cost (number of p-labeled nodes) of the path by verifying whether there is a path of smaller cost to v_j and iterating through all incoming edges $(v_k, v_j) \in E$ to determine whether the minimum-cost path to v_k will yield a minimum-cost path to v_j.

***Theorem** 3 (Optimal Substructure):* Let $\pi^{v_1 \to v_n} = \pi^{v_1 \to v_j} \cup \pi^{v_j \to v_n}$ be the path constructed from $(1), (2), (3)$. Then $\pi^{v_1 \to v_j}$ is the path from v_1 to v_j with the minimum number of p-labeled nodes such that constraints $|L_1^\pi| \leq k_1, \ldots, |L_{p-1}^\pi| \leq k_{p-1}$ are satisfied.

Proof: Suppose a path $\sigma^{v_1 \to v_j}$ from v_1 to v_j exists such that $|L_p^\sigma| < u_j^{|\sigma^{v_1 \to v_j}|-1}(t_1, \ldots, t_{p-1})$ and $|L_1^\sigma| \leq t_1, \ldots, |L_{p-1}^\sigma| \leq t_{p-1}$. Then there exists some path $\sigma' \subset \sigma^{v_1 \to v_j}, \sigma' \cap \pi^{v_1 \to v_j} = \emptyset$ not contained in $\pi^{v_1 \to v_j}$. Let v_α denote the last vertex in path σ' and let $v^* \in \sigma^{v_1 \to v_j} \cap \pi^{v_1 \to v_j}$ be the vertex connecting σ' to $\pi^{v_1 \to v_j}$ (i.e. $(v_\alpha, v^*) \in E$). $\pi^{v_1 \to v^*} \subseteq \pi^{v_1 \to v_j}$ is the path from v_1 to v^* contained in $\pi^{v_1 \to v_j}$. Similarly, let $\sigma^{v_1 \to \sigma_1'}, \sigma^{\sigma_1' \to v^*} \subset \sigma^{v_1 \to v_j}$ represent sub-paths of $\sigma^{v_1 \to v_j}$. Two cases arise:

- $|\sigma^{v_1 \to \sigma_1'} \cup \sigma^{\sigma_1' \to v^*}| < |\pi^{v_1 \to v^*}|$: When $m = |\sigma^{v_1 \to \sigma_1'} \cup \sigma^{\sigma_1' \to v^*}|$, the first parameter in the outer minimization of (3) will set $u_j^m(t_1, \ldots, t_{p-1}) \leq |L_p^\sigma|$. This result will then propagate to larger m.
- $|\sigma^{v_1 \to \sigma_1'} \cup \sigma^{\sigma_1' \to v^*}| \geq |\pi^{v_1 \to v^*}|$: Since $(v_\alpha, v^*) \in E$, the inner minimization in (3) will check the value $u_\alpha^m(t_1, \ldots, t_{p-1})$ when $m = |\sigma^{v_1 \to \sigma_1'} \cup \sigma^{\sigma_1' \to v^*}|$ and set $u_j^m(t_1, \ldots, t_{p-1}) \leq |L_p^\sigma|$.

It follows that no such path $\sigma^{v_1 \to v_j}$ with $|L_p^\sigma| < u_j^{|\pi^{v_1 \to v_j}|-1}(t_1, \ldots, t_{p-1})$ exists. □

It follows from Theorem 3 that our algorithm returns the path from v_1 to v_n with the minimum number of p-labeled nodes such that the constraints $|L_1^\pi| \leq k_1, \ldots, |L_p^\pi| \leq k_p$ are met. The proof of optimal substructure in Theorem 4 is similar and is therefore omitted.

Note that (3) iterates through paths of length m. The maximum length of a path in $G = (V, E, \{L_1, \ldots, L_p\})$ is $|V| - 1$ and there can be no path of length $k_1 + k_2 + \cdots + k_p$ that satisfies $|L_1^\pi| \leq k_1, \ldots, |L_p^\pi| \leq k_p$ since one of the constraints would be violated. Thus, $m = 0, 1, \ldots, (\sum_{i=1}^p k_i) - 1$, leading to $\sum_{i=1}^p k_i \leq |V| - 1$ iterations in (3). In each iteration, (3) must also process every vertex and check the incoming edges for said vertex, yielding $|V| d_{avg}$ iterations, where d_{avg} is the average degree of the vertices in G. Furthermore, for every constraint k_i ($i \neq p$), we must iterate from 0 to k_i, yielding $\Pi_{i=1}^{p-1}(k_i + 1)$ iterations. This results in a runtime complexity of $\mathcal{O}\left(|V| d_{avg} \left(\sum_{i=1}^p k_i\right) \left(\Pi_{i=1}^{p-1} k_i\right)\right)$.

Interval-Constrained Paths

Given a labeled graph $G = (V, E, \{L_1, \ldots, L_p\})$ and constraint pairs $(k_1, k_1'), \ldots, (k_p, k_p')$, we make the assumption that there is some $k_i = 0$. This assumption allows us to minimize over the set L_i using a similar procedure to the one used to solve the cardinality-constrained paths problem. Without loss of generality, we assume that $k_p = 0$. Thus, the problem that we solve is a special class of the interval-constrained paths problem.

In order to satisfy the interval constraints introduced by the ICP problem, we add a simple modification to the pre-

vious approach. The only change is to equation (1), which is now replaced by (4). This change to the initialization equation causes the constraint parameters $t_1, \ldots, t_{p-1}$ to be met exactly as opposed to as upper bounds. Let $\widehat{u}_j^m(t_1, t_2, \ldots, t_{p-1})$ denote the minimum number of p-labeled nodes in a path from the source node v_1 to v_j such that the number of k-labeled nodes is **exactly** $t_k, 1 \leq k \leq p-1$, and the path consists of at most m edges. The values of $\widehat{u}_j^m(t_1, t_2, \ldots, t_{p-1})$ can be defined by equations $(2), (3), (4)$. The addition of (4) causes the entries in the resulting dynamic programming matrix to match the constraint parameters $t_1, \ldots, t_{p-1}$ with exactitude. It is easy to see that the runtime of the algorithm given by $(2), (3), (4)$ is $\mathcal{O}\left(|V| d_{avg} \left(\sum_{i=1}^{p} k_i'\right) \left(\Pi_{i=1}^{p-1} k_i'\right)\right)$. That is, the runtime is the same as that of $(1), (2), (3)$ used to solve the constrained-paths problem.

$$\widehat{u}_1^0(t_1, \ldots, t_{p-1}) = \begin{cases} 1 & (\mathcal{I}_p(v_1) = 1) \wedge \bigwedge_{i=1}^{p-1}(t_i = 0) \\ 0 & \bigvee_{i=1}^{p-1}\left(t_i = 1 \wedge \mathcal{I}_i(v_1) = 1 \wedge \bigwedge_{j \neq i} t_j = 0\right) \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

The solution to the interval-constrained paths (ICP) problem given by $(2), (3), (4)$ yields a simple path when the network of interest is a directed acyclic graph. Otherwise, the solution may yield a path with a cycle as a solution. This is to be expected due to the complexity of the problem even when the number of labels is fixed. Therefore, this problem is believed to be intractable to solve in the general case.

***Theorem** 4 (Optimal Substructure):* Let $\pi^{v_1 \to v_n} = \pi^{v_1 \to v_j} \cup \pi^{v_j \to v_n}$ be the path constructed from $(2), (3), (4)$. Then $\pi^{v_1 \to v_j}$ is the path from v_1 to v_j with the minimum number of p-labeled nodes such that constraints $|L_1^\pi| = k_1, \ldots, |L_{p-1}^\pi| = k_{p-1}$ are satisfied.

IV. Experimental Results

The average of length of a path in random networks with varying degrees of connectivity has been studied in [4]. This length follows the equation $\lambda(n, k) = (ln(n) - \gamma)/ln(k) + 1/2$, where n is the number of vertices, γ is the Euler-Mascheroni constant, and k is the average degree of nodes in the graph. For large networks with one million nodes, this yields an average path length of 10 for low-connectivity networks ($k = 4$) and 5 for the case of high-connectivity ($k = 20$). We use $\lambda(n, 4)$ and $\lambda(n, 20)$ to define the constraints used in our runtime results (Fig. 5). These small path lengths have been evidenced in real complex networks. In an analysis of the Facebook social network with 721 million user nodes, it was found that the average path length between two nodes was just 4.74

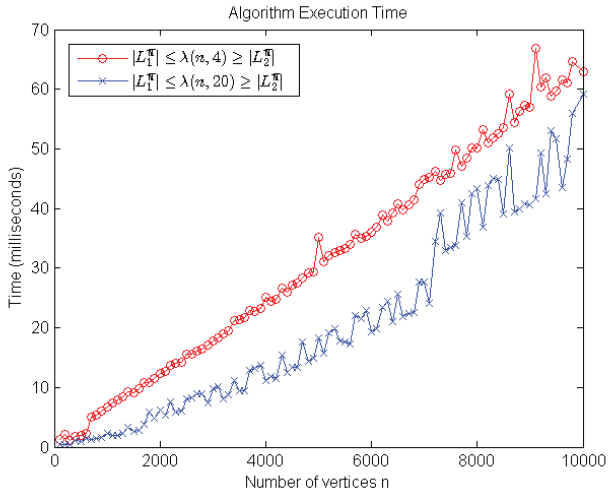


Fig. 5. Running time, in milliseconds, of our algorithm on randomized networks of various sizes. We look at networks whose average degree and path length are $k = 4$ and $\lambda(n, 4)$, respectively, and $k = 20, \lambda(n, 20)$, where n is the number of nodes in the network.

[2]. This leads us to believe that cardinality-constrained paths in $G = (V, E, \{L_1, \ldots, L_p\})$ have the property $|\pi^{v_1 \to v_n}| \leq \sum_{i=1}^{p} k_i << |V|$ in the general case.

References

[1] Saif Al-Sultan, Moath M Al-Doori, Ali H Al-Bayatti, and Hussien Zedan. A comprehensive survey on vehicular ad hoc network. *Journal of network and computer applications*, 37:380–392, 2014.

[2] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *Proceedings of the 4th Annual ACM Web Science Conference*, pages 33–42. ACM, 2012.

[3] Yuh-Shyan Chen and Yun-Wei Lin. A mobicast routing protocol with carry-and-forward in vehicular ad hoc networks. *International Journal of Communication Systems*, 27(10):1416–1440, 2014.

[4] Agata Fronczak, Piotr Fronczak, and Janusz A Hołyst. Average path length in random networks. *Physical Review E*, 70(5):056110, 2004.

[5] John Harding, Gregory Powell, Rebecca Yoon, Joshua Fikentscher, Charlene Doyle, Dana Sade, Mike Lukuc, Jim Simons, and Jing Wang. Vehicle-to-vehicle communications: Readiness of v2v technology for application. Technical report, 2014.

[6] Muhammad Adeel Mahmood, Winston KG Seah, and Ian Welch. Reliability in wireless sensor networks: A survey and challenges ahead. *Computer Networks*, 79:166–187, 2015.

[7] Intelligent Transportation Systems Joint Program Office. Vehicle-to-infrastructure (v2i) communications for safety. http://www.its.dot.gov/factsheets/pdf/JPO-022%20V2ISAFETY%20V5.5.2%20F.pdf.

[8] Mukesh Saini, Abdulhameed Alelaiwi, and Abdulmotaleb El Saddik. How close are we to realizing a pragmatic vanet solution? a meta-survey. *ACM Computing Surveys (CSUR)*, 48(2):29, 2015.

[9] Yasser Toor, Paul Muhlethaler, Anis Laouiti, and Arnaud De La Fortelle. Vehicle ad hoc networks: applications and related technical issues. *IEEE communications surveys & tutorials*, 10(3):74–88, 2008.

[10] Athanasios V Vasilakos, Yan Zhang, and Thrasyvoulos Spyropoulos. *Delay tolerant networks: Protocols and applications*. CRC press, 2016.

A Network Service Design and Deployment Process for NFV Systems

Sadaf Mustafiz*, Francis Palma*, Maria Toeroe† and Ferhat Khendek*
*Department of Electrical and Computer Engineering
Concordia University, Montreal, Canada.
Email: {sadaf.mustafiz, ferhat.khendek}@concordia.ca, f_palma@encs.concordia.ca
†Ericsson Inc., Montreal, Canada
Email: maria.toeroe@ericsson.com

***Abstract*—Recently, the paradigm of Network Functions Virtualisation (NFV) has emerged for the rapid provisioning and management of network services. It is based on the cloud paradigm and the virtualisation technology. The European Telecommunications Standards Institute (ETSI) has been actively defining the NFV framework, which includes several functional blocks for network service provisioning and management. The interfaces and the roles of these functional blocks are being defined as well as the artifacts they manipulate. However, the workflow defining the relations and dependencies between these different blocks as well as the successive processing of the artifacts throughout this workflow have not been specified. This is the purpose of this paper where we define an NFV standard compliant process for network service design and deployment. The process starts from the tenant network service requirements all the way to the network service deployment.**

I. Introduction

Network Functions Virtualisation (NFV) is an emerging paradigm that builds on cloud computing [1] and the virtualisation technology to eliminate the drawbacks of traditional physical network infrastructure [2], [3] and enables rapid provisioning of network services (NSs). There is no need for the deployment of a wide range of network equipments, which requires substantial capital and operating expenses. In contrast, NFV exploits the virtualisation technology and cloud resources, and remodels the physical devices into virtual entities known as Virtualised Network Functions (VNFs) implemented as software packages [4].

The NFV reference architectural framework standardised by ETSI [3] and adopted by TOSCA [5], defines several functional blocks, *i.e.*, *actors*, playing different roles in the various phases of NS and VNF lifecycle management, from on-boarding to operations and monitoring. The current ETSI standard [6] specifies the NFV reference framework, its functional blocks, their roles, their interfaces, and some NS and VNF-related operational flows. An NS and VNF deployment and management process is implied from these functional blocks, interfaces, and the operational flows, however the workflow as such is not defined.

In this paper we propose such a workflow, one possible chaining of the different functions and operations defined in the standard. These functions and operations are grouped into activities. The relations and dependencies between these activities as well as the dependencies and the relations between the NFV reference framework functional blocks are explicitly defined in the workflow. The proposed workflow is compliant with the NFV reference framework as it obeys to the standard definitions of the functional blocks, roles, and interfaces. Moreover, this process goes beyond the reference framework and proposes the automatic generation of a new NS Descriptor (NSD) and other artifacts from the tenant's NS requirements and the catalogue of VNF Descriptors (VNFDs) upon tenant request. This workflow is a first step toward the necessary automation of network service design and deployment process for NFV systems. To achieve this, we (1) investigate and identify the different functions and operations in relation to the functional blocks; (2) identify and capture various interactions among the functional blocks; (3) identify the limitations and open issues in the NFV reference framework; (4) determine how the different artifacts are refined throughout the process; (5) fill the gaps and define a concrete process, with high-level activities for grouping together operations and functions defined in the standard, that is compliant to the NFV reference framework.

This paper is structured as follows: Section II gives a brief background on the NFV reference framework, its functional blocks, and various NFV artifacts. Section III outlines the underlying process as inferred from the standard, and identifies its limitations and open issues. Section IV presents the NS design and deployment workflow and its Process Model (PM). Section V discusses possible variations in the proposed process and how it can be adapted and/or extended. In Section VI, we review the related work. Finally, Section VII concludes with some future work.

II. Background

This section provides a brief introduction to the various functional blocks in the NFV reference architecture and its artifacts as proposed in the ETSI standard [3].

A. Main Functional Blocks in the NFV Architecture

As shown in Figure 1, the main functional module in the architecture is the NFV Management and Orchestration (NFV-MANO), which is in charge of deployment, management,

978-1-5090-3216-7/16/$31.00

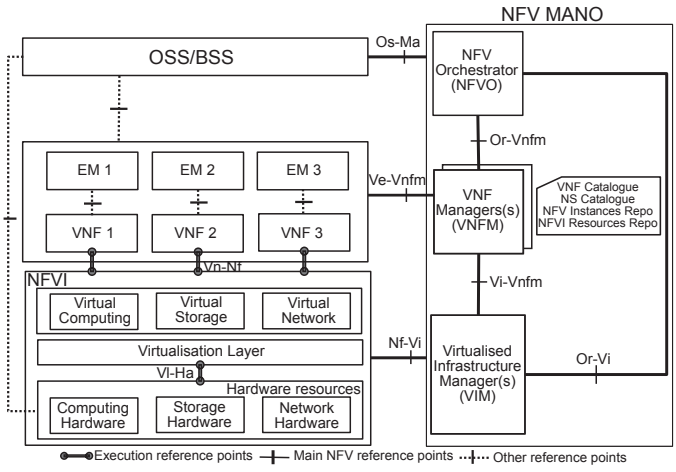


Fig. 1. The NFV Architecture proposed by ETSI [3].

and orchestration of NSs. The NFV-MANO consists of three functional blocks: the NFV Orchestrator (NFVO), the VNF Manager (VNFM), and the Virtualised Infrastructure Manager (VIM). In the following, we briefly discuss these functional blocks, which we identify as *actors* in the rest of the paper.

The main responsibilities of **NFV Orchestrator (NFVO)** include the NS orchestration and lifecycle management, *e.g.*, (1) on-boarding VNFs and NSs; (2) instantiation of NSs and VNFMs; and (3) NS policy management related to affinity/anti-affinity, scaling, fault, and performance.

VNF Manager (VNFM) is in charge of the lifecycle management of VNF instances. A VNFM is assigned to each VNF instance. However, a VNFM may also manage multiple VNF instances of the same type. The main responsibilities of VNFM include the instantiation, configuration, update and upgrade management, modification, scaling, and termination of its VNFs performed in collaboration with the NFVO [3].

Virtualised Infrastructure Manager (VIM) manages and orchestrates the NFVI resources. The VIM supports the management of VNF Forwarding Graphs (VNFFGs), which includes creating and maintaining virtual links (VLs). VIM also performs the NFVI physical and software resource repository management [3].

B. Other Functional Blocks in NFV Architecture

The NFV reference architecture includes other functional blocks, like the VNF, the EM, the NFVI, and the OSS/BSS.

Virtualised Network Functions (VNFs) are the building blocks of an NS in NFV. VNFs are software pieces with the same functionality as their corresponding physical network functions, *e.g.*, a virtual firewall (vFW) *vs.* a traditional firewall device. A VNF can be composed of multiple internal components (VNFC). The description of the deployment and operational behaviour of a VNFC is defined as a Virtual Deployment Unit (VDU) [4].

Element Management (EM) carries out the fault management, configuration, accounting, performance, and security (FCAPS) functionality for one or multiple VNFs. The EM works together with the VNFM by exchanging information on NFVI resources to manage the virtualised resources for a VNF [3].

Network Functions Virtualisation Infrastructure (NFVI) is the platform on which the network services are deployed. The NFVI has the physical layer of devices for computation, storage, and network; the virtualisation layer, *e.g.*, hypervisor; and the virtualised resources, *e.g.*, virtual machines. The NFVI is the totality of physical and virtual resources [3].

Operations Support Systems and Business Support Systems (OSS/BSS) refer to the operator's proprietary systems and management applications supporting their business. The OSS/BSS systems exchange a lot of information with NFV-MANO functional blocks to provide the desired network service [3]. Moreover, in the NFV architecture, the OSS/BSS provides management and orchestration of legacy systems.

C. NFV Artifacts

During the lifetime of an NS, various artifacts at various levels of abstractions are used and produced. The artifacts can be divided into three categories: *descriptors*, *records*, and *data repositories* (see Figure 2). In the following, we briefly describe each type of artifacts.

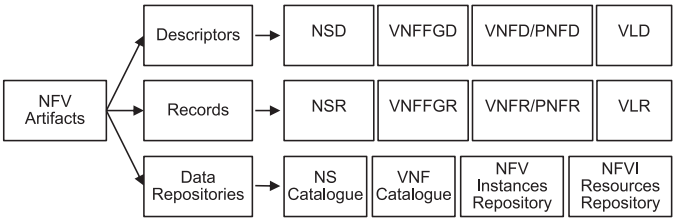


Fig. 2. NFV Artifacts.

Descriptors describe the deployment requirements, operational behaviour, and policies required by the NSs or VNFs. The virtual links (VLs) that connect VNFs to form a network topology also come with their descriptors—VLDs (Virtual Link Descriptors). Similarly, the VNF Forwarding Graph (VNFFG) has a descriptor, *i.e.*, VNFFGD, which defines an end-to-end sequence of VNFs that packets traverse. All these descriptors, *e.g.*, NSDs, VNFDs, VNFFGDs, and VLDs, are known as *deployment templates*.

Records describe individual instances of NSs, VNFFGs, VLs, and VNFs/PNFs. These records are stored within the NFV Instances Repository.

Data Repositories that are defined in the NFV architecture are (1) the NS Catalogue containing all the on-boarded NSDs, VNFFGDs, and VLDs; (2) the VNF Catalogue containing all the on-boarded VNFDs; (3) the NFV Instances Repository containing all the instance records; and (4) the NFVI Resources Repository holding information about the available, reserved, and allocated NFVI resources as updated by the VIM.

III. The NS Deployment and Management Process Implied in the Standard

In this section, we discuss the underlying NS deployment process as inferred from the standard functional block descriptions, their interfaces, operational flows, use cases, and case studies. Then, we identify its limitation and highlight the open issues.

A. The Implied Process

The ETSI standard includes a discussion on the initiation and deployment process of NSs and VNFs. As mentioned in the standard, a network service is requested by a tenant's OSS/BSS. The NFVO then on-boards the VNFDs into the VNF Catalogue and the NSD, the VNFFGDs, and the VLDs into the NS Catalogue. During on-boarding, the NFVO checks the integrity of NSD, VNFFGD, VLD, and VNFD/PNFD. Once the descriptors have been on-boarded the NS can be instantiated on request.

In the course of the instantiation process, the NFVO receives additional instantiation parameters from the entity (*e.g.*, OSS/BSS) initiating the instantiation process. These instantiation input parameters contain information on deployment flavours and are used to customise a specific NS instance and its VNF instances. The deployment flavours specify parameters for CPU utilisation and other factors related to the virtual machine on which the VNFs are deployed. The instantiation process results in various records, namely the Network Service Records (NSRs), the VNF Records (VNFRs), the Virtual Link Records (VLRs), and the VNFFG Records (VNFFGRs). This is followed by the deployment of the NS and the associated VNFs.

B. Limitations and Open Issues

We have identified some limitations and open issues that are not addressed in the current NFV standard:

- Although the NFV framework describes three functional blocks under MANO, they are typically not implemented separately due to lack of clear segregation of responsibilities during the instantiation and deployment phases [7];
- The standard identifies various orchestration functions and operations along with operational flows that are part of the NS and VNF lifecycle management. However, the standard does not propose any logical grouping of these functions and operations;
- The standard does not specify explicitly the *dependencies* among the various functional blocks in the NFV framework, *i.e.*, how the functions and operations are interdependent to achieve the NFV goals;

As a consequence, there is no concrete process (or workflow) that defines the execution order of the various functions and operations. A detailed and concrete process model is required to put the reference framework into practice. It will enable the automation of network service deployment and management.

A vendor providing a VNF implementation may not want to restrict it in the associated VNFD to a particular deployment option, but allows for the widest possible range of options appropriate for the deployment. Compared to this, a given NFVI may limit the deployment options, for example, due to some policies or available resource of the operator/NFV provider. Furthermore, an NS may impose further limitations depending on the tenant's policies and/or targeted key performance indicator (KPI) values. Considering that the same VNF package may be used to implement different NSs with diverse policies and KPIs the use of the same VNFD is problematic. Currently, the specifications do not distinguish these different stages and define only a single VNFD associated with each VNF to reflect these scenarios. A detailed process will distinguish these different stages and the different tailorings.

In the following section, we propose a process for NS design and deployment which is compliant to the NFV reference framework. In this process we identify high-level activities for grouping the different functions and operations, their dependencies and relationships. We clearly specify the inputs, outputs of each activity and the tailoring of the artifacts. In addition, we go beyond the standard and include an activity for the automated design of the NSD from the tenant's NS requirements (NSReq). We refer to this activity in the rest of the paper as the *NS Design* activity. This brings us to an additional limitation of the current specifications:

- The NFV specifications focus on the virtualisation and software management aspects of the VNFs, *e.g.*, the interfaces and KPIs. However, the current specifications do not define any method that would help in identifying the functionalities a VNF can provide for its users.

IV. Network Service Design and Deployment Process

We define a process for the design, configuration and deployment of network services. We elaborate on the functions of the OSS/BSS in the context of our process, and propose a workflow which is a chaining of the various NFV functions and operations grouped into activities required to carry out NS deployment. Our work is based on the **VNF as a Service** (VNFaaS) model as defined in the standard [3], [4], [6]. We go beyond what is defined in the standard: we bring the OSS/BSS, the EM, and other external subsystems (NFVI resources repository, NFV instance repository, NS and VNF catalogues) into the picture along with the NFV-MANO to propose an NS design and deployment process.

We identified the activities that are part of the workflow. We describe each activity (based on the orchestration functions and operations defined in the ETSI standard) along with the dependencies (in terms of information elements) between the activities which sets a well-defined scope for each of them. We also identify clearly and associate the *actor* responsible for each of these activities including the activity of NS design. The NS design is a complex activity that as mentioned earlier is outside the scope of the NFV standard. Before presenting the whole process we describe briefly the NS design activity.

Please note that in our process we have abstracted away from the policy management (e.g., scaling and threshold) and

monitoring aspects of NS that are required to carry out SLA monitoring, management and dynamic reconfiguration, and focused on the design and deployment of network services.

A. NS Design

Network service design entails the definition of deployment templates (namely NSD, VNFFGD, and VLD), which includes static information elements related to an NS. This NSD is used by the orchestrator during the deployment.

We propose a method for NSD generation by taking inspiration from [8]. A tenant may request a new NS by specifying the NS Requirements (NSReq), which consists of functional and non-functional requirements possibly with some initial decomposition targeting specific VNFs. In our work, we assume that the OSS/BSS of an NFV provider gets the NSReq and generates the NSD based on the provider's `VNF Catalogue`. The NSD generation method involves the decomposition of the NSReq and the selection of proper network functions, *e.g.*, VNFs (and/or PNFs[1]) from the `VNF Catalogue`. The NSReq decomposition is guided by a network function ontology (NFO), which captures standard network function (de)compositions as defined by different standardisation bodies such as 3GPP as well as knowledge and experience from previous decompositions. The NFO captures the decomposition of network functionalities to the level of granularity where each functionality can be mapped onto some VNF provided functionality. When the decomposition reaches that level, VNFs from the `VNF Catalogue` are matched and selected to compose the network service. The selection takes into account the non-functional requirements and the VNFD of the selected VNFs are tailored accordingly to NS-specific VNFDs, referred to as `NS-VNFDs`. During this activity, the VNF forwarding graph descriptor (VNFFGD) and the virtual link descriptors (VLD) are also generated.

B. NS Design and Deployment Process Model

In this section, we propose a concrete model-based workflow covering the phases from NS design to deployment. With the process, we have attempted to bridge the gaps between the various functional blocks and the various activities that are part of NS design and deployment. Having a model-based process also allows us to take advantage of the ETSI/TOSCA defined NFV profiles.

We present a Process Model (PM) that describes the process of designing and deploying a new network service along with the complete workflow. The PM clearly shows the control-flow and object-flow among the actors involved in the process. The PM language is a variant of the PM which is part of the FTG+PM language [9]. The PM is modelled using the UML2.0 Activity Diagram [10] formalism which is typically used to model software and business processes. The formalism allows the modelling of concurrent processes and their synchronisation with the use of fork and join nodes. Both control-flow and object-flow can be depicted in the model. An activity node can either be a simple *action* (representing a single step within an activity) or an *activity* (representing a decomposable activity which embeds actions or other activities). An activity specifies a behaviour that may be reused, *i.e.*, an activity can be included in other activity diagrams to invoke behaviour. For representation purpose and to add clarity to the entire process, we use activity diagram partitions (*a.k.a.*, swimlanes) where each partition represents one actor (participating functional block) in the process. Multicasting and multireceiving are used in conjunction with partitions to model object flows between activities that are the responsibility of objects determined by a publish and subscribe facility [10]. Object flows are tagged with `<<multicast>>` or `<<multireceive>>` in this case. Since UML2.0 Activity Diagrams are given semantics in terms of Petri Nets [10], the precise formal semantics allow the activity diagrams to be simulated and analysed.

In our PM, we consider the artifacts discussed in Section II as models (instances which conform to existing meta-models) which are either inputs or outputs of activities.

The PM covering the design, configuration generation, and deployment activities is presented in Figure 3. Along with the high-level activities, the PM clearly shows the input and output models associated with each activity via input and output pins (denoted by the tiny squares on the activity border). In the PM, thin edges denote object-flow, while thick edges denote control-flow. The join and the fork constructs are represented as horizontal bars.

The role of each NFV functional block is shown as *partitions* in the PM. The partitions tagged as `<<external>>` denotes blocks and actors outside the NFV-MANO. The partition labelled as `Other Blocks` includes blocks in the NFV architecture that are considered as black-boxes in our work and which communicate with the other actors via predefined interfaces. The activities associated with the partitions or actors are in accordance with the roles of the NFV functional blocks as specified in the current standard [6].

We now elaborate on the activities, the associations between the different actors, and the dependencies between the activities, as depicted in Figure 3. The activities (1) and (3) in the list below can be carried out concurrently (shown with the parallel flow coming out of the fork construct in Figure 3). Also, note that the activities in (1) and (2) are continuous in nature and not part of the process for the NS design and deployment, which starts with the NS design activity triggered by a tenant's request for a new service.

(1) Update VNF Catalogue (actor: OSS/BSS). The vendor, an external actor, sends a new VNF package (including the `VNFD` and the `VNF Implementation`). This is received by the OSS/BSS (also an external actor), and the `OSS-BSS VNFCatalogue` maintained by the OSS/BSS is updated. This is a continuous behaviour which accepts a stream of incoming VNF packages from vendors and updates the internal

[1]NS management typically also involves PNFs on-boarding, PNFs configuration generation, and PNFs deployment activities. The lifecycle of PNFs is similar to that of VNFs. However, the lifecycle of PNFs are not in the scope of this paper and we have chosen not to explicitly show PNFs.

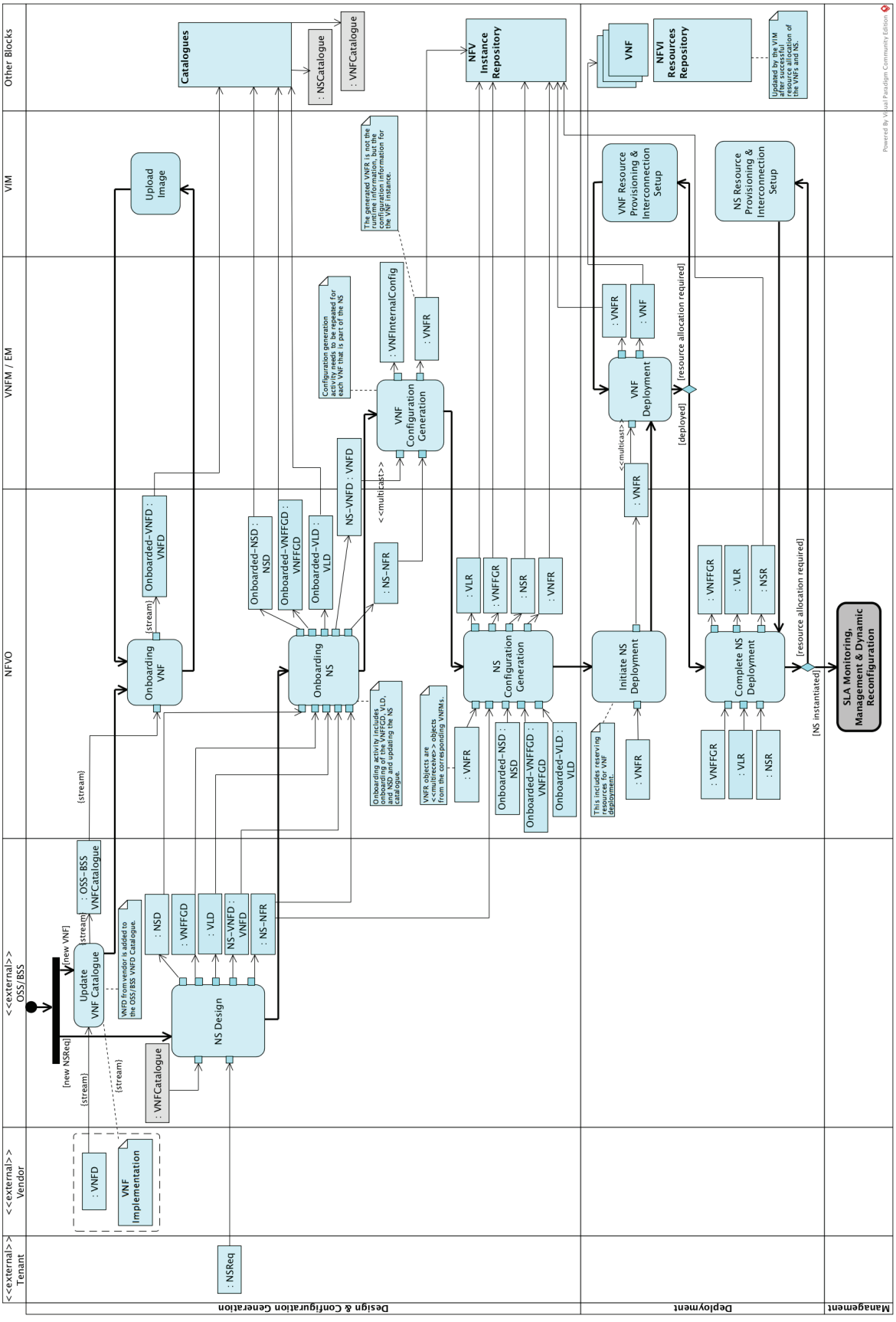


Fig. 3. NS Design and Deployment Process Model (PM).

catalogue. Streaming is shown with the text annotation *stream* placed near the (input/output) pin symbol in the PM.

(2) On-boarding VNF (actor: NFVO). The NFVO takes the VNF specification and configuration data that is part of the incoming VNF package from the `OSS-BSS VNFCatalogue` to on-board the VNFs. The NFVO validates the `VNFD` and updates the `VNFCatalogue` (maintained by NFV-MANO) with the `Onboarded-VNFD`. After that, the NFVO requests VIM to upload the VNF image (triggering the **Upload Image** action). The on-boarding of a VNF is required prior to its deployment. On-boarding is a continuous behaviour which periodically receives vendor-provided VNF packages from the OSS/BSS and streams out the `Onboarded-VNFD` model for each incoming VNF. At this point, the VNFDs are generic, as in they are not associated with any NS. The VNFs are made NS-specific in the **NS Design** activity.

(3) NS Design (actor: OSS/BSS). The tenant, an external actor, requests a new network service by providing a new network service requirements (`NSReq`) which triggers the whole process starting with the NS design activity within the OSS/BSS. The activity also takes as input the NFV-MANO `VNFCatalogue` consisting of on-boarded VNFs, to produce an NSD according to the available VNFs. The deployment templates including the `NSD`, `VNFFGD` and the `VLD` to be deployed are created during this activity as described in Section IV-A. The VNFDs (from (2)) are specialised for this NS resulting in `NS-VNFDs`. An additional model, `NS-NFR`, is also generated and contains the non-functional requirements (NFR) specified in the NSReq that need to be carried forward and addressed during the configuration generation activity.

(4) On-boarding NS (actor: NFVO). Following NS design, the NFVO continues with on-boarding the NS. On-boarding entails validating the NSD, VNFFGD, and VLD to check for missing elements and updating the `NS Catalogue`. This activity tailors the deployment templates resulting in the artifacts `Onboarded-NSD`, `Onboarded-VNFFGD` and `Onboarded-VLD`. The NS-specific VNFD(s) and the NFRs are received from the OSS/BSS and forwarded to the corresponding VNFM through the NFVO.

(5) VNF Configuration Generation (actor: VNFM). Following the NS on-boarding by the NFVO, the NS-specific VNFs need to be configured with deployment-specific details (based on their deployment templates) to proceed with deployment. The internal configuration of each VNF is generated as part of this activity. The internal connections between the VNF components (VNFCs) are also configured. A configuration generation request (along with the `NS-VNFD` and the `NS-NFR`) is received from the NFVO by each VNFM. Each configuration generation includes the generation of the `VNFInternalConfig` and the `VNFR`. The internal configuration may determine what the VNF configuration should be and how the `VNFR` should be created. The `NS-NFR` is used to determine the number of instances of VNFs. The records created are stored in the NFV Instance Repository. The VNFMs respond by returning the `VNFRs` created to the NFVO, which is done via *multireceive*, *i.e.*, the objects in the flow are gathered from respondents to multicasting.

(6) NS Configuration Generation (actor: NFVO). The deployment-specific details of an NS are now generated to enable NS deployment. Once the VNFRs are created for the VNFs, the NS configuration activity is triggered by the NFVO (once all `VNFR` input models for the associated VNFs are available for the activity). The external links (VLs) between the VNFs are configured during this stage. The instantiation takes into account the constraints defined in `NS-NFR`. The records created (*e.g.*, `VNFFGR`, `VLR`, and `NSR`) representing the NS instances along with the refined `VNFRs` are saved in the NFV Instances Repository.

(7) Initiate NS Deployment (actor: NFVO). The NFVO then initiates the NS deployment. This begins with the deployment of the VNFs that are part of the NS. The NFVO checks for resource availability for each VNF. The resource reservation for the VNFs can also optionally be carried out by the NFVO via the VIM. Once the resources are reserved, the NFVO sends the reservation acknowledgment to the corresponding VNFM (by multicasting the `VNFRs`). The VIM identifier is also passed to inform the VNFM where to deploy the VNF. This triggers the deployment of the VNFs by the VNFMs. *It is also possible that OSS/BSS controls the deployment and this activity is only triggered upon its request.*

(8) VNF Deployment (actor: VNFM/EM). Once the resource reservation is completed in activity (7), each VNFM communicates with the VIM for resource allocation. The **VNF Resource Provisioning and Interconnection Setup** action is triggered in the VIM. The VNFM then configures the `VNF` with deployment-specific parameters (VNF specific lifecycle parameters), which involves instantiation and configuration of the VNF components according to the VNFR. The EM is involved with the configuration of `VNF` with application-specific parameters to complete the actual VNF deployment. At this point the VNFR is updated as well as the NFV Instances Repository. Once all the VNFs are deployed, the NFVO continues with the NS deployment.

(9) Complete NS Deployment (actor: NFVO). Following the deployment of the VNFs, the NFVO continues to deploy the `NSR`, `VLR`, and `VNFFGR` by requesting the VIM for network connectivity creation. The **NS Resource Provisioning and Interconnection Setup** action is triggered at this point. Upon confirmation from the VIM of successful creation of network connectivity, the NFVO requests the VIM to connect the VNFs to the network (setting up the VLs that are part of the VNFFG(s)). The deployment parameters in the records are updated as well as the NFV Instances Repository.

According to the standard, it is possible to initiate resource allocations in different ways. For this reason, in the standard there are several variants of the operational flow for VNF and NS deployment (depending on whether the resource allocation is initiated by the NFVO or the VNFM). Therefore, it should

be noted that it is possible to have variations in the workflow depicted in the PM depending on the interaction model chosen.

Following deployment, the NS instance management continues until its decommissioning. The PM discussed in this paper does not include this part which is left for future investigations.

C. Artifacts Perspective of the Process Model

In the NFV domain, various *artifacts* are used as discussed in Section II. However, as we also pointed out with respect to the VNFD, there could be a substantial shift from what is initially provided by the vendor and the specific version that is deployed at the end for a tenant. The current NFV specification does not clearly state the different specialisations an artifact goes through during the NS design and deployment process. In fact, the NFV artifacts, *e.g.*, the VNF and NS, get transformed through a chain of activities as shown in Figure 4. In this section, we focus on the VNF and the NS specialisations and illustrate them from the artifact perspective to complement the PM in Figure 3. The information elements part of the NS and VNF artifacts are detailed in [6]. Other artifacts like VLs and VNFFGs are also tailored, but are not discussed here due to space constraints.

1) VNF Tailorings: The initial `Vendor-provided VNFD` is generic, *i.e.*, open to many deployment options of the VNF, and includes the VNF identification data (*e.g.*, ID, vendor, and version), VNF-specific data (*e.g.*, connectivity requirements, inter-dependencies of VNFCs, and deployment flavours), VNFC data (*e.g.*, specific VNFC configuration data and deployment constraints), and virtualised resource requirements. The successive tailorings of a VNF as shown in Figure 4 (top) are discussed here.

- Tailoring 1: The orchestrator checks for any missing mandatory information elements in the generic VNFD and provides with the default values, thus resulting in an `Onboarded VNFD` model instance.
- Tailoring 2: The `NS-specific VNFDs` are the result of VNF selection from the `VNFCatalogue` and the tailoring of the VNFDs for a specific NS based on the NSReq from the tenant where various VNF/VNFC deployment constraints and virtualised network resource requirements might be added/updated in the VNFD by the OSS/BSS.
- Tailoring 3: Once the VNFDs are tailored to be NS-specific, the VNFM/EM configures VNF-specific data for VNF instantiation. The `VNFR` is created and the VDU-level configuration is performed. Additional information elements are introduced in the `VNFR`, for instance, the parent NS, the logging capabilities for audit purpose, the network address, and the VNFM managing this instance.
- Tailoring 4: A further tailoring of the `VNFR` is possible based on the deployment flavour and geographical location constraint, which results in `NS-specific VNFR`.
- Tailoring 5: During the deployment of the VNFs, the policies, *e.g.*, affinity and anti-affinity rules, are applied between VMs and hosts, which results in the refinement of `VNFRs` to adhere to the policies.

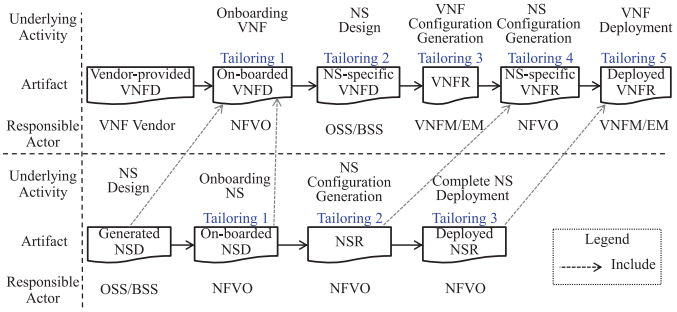


Fig. 4. Successive tailoring of a VNF (top) and an NS (bottom).

2) NS Tailorings: The `Generated NSD` includes the on-boarded NS-specific VNFDs, the VNFFGDs, and the VLDs. The successive tailorings of an NS as shown in Figure 4 (bottom) are as follows:

- Tailoring 1: For the `Onboarded NSD`, the NFVO checks for any missing mandatory information elements in the NSD and provides with the default values, if required, which is the first tailoring of an NSD.
- Tailoring 2: Later, the NFVO instantiates the NS by creating an `NSR` and adds additional information to the on-boarded NSD, for instance, information on NFVI resources reserved for this NS instance, the NS status, and the logging capabilities for audit purpose.
- Tailoring 3: During the deployment of an NS, various policies (*e.g.*, the NS scaling policy) are provided from OSS/BSS resulting in a further specialisation of an `NSR`.

V. Discussion

Our PM may have workflow variations, *i.e.*, the NSD can be designed from scratch or an existing NSD can be reused based on the new functional/non-functional requirements from tenants, which we discuss in the following. We also discuss how the proposed PM can be enacted using the benefits of model-driven engineering with the final goal to automate the workflow for designing and deploying an NS.

A. Variations in the Workflow

The proposed process covers the design, configuration and deployment of a new network service triggered by a request from a tenant in the form of a NSReq consisting of functional and non-functional requirements. The NS Design activity can include a function for checking the NS Catalogue for a service that meets the NSReq before starting the design to reuse what is already available. However, if the non-functional requirements are taken into account in the design of the NS, the probability of a matching NSD is low. One can think of an alternative workflow where the NSReq consists only of functional requirements to improve NS re-usability. In this case the NS is designed with multiple versions in a first step and the non-functional requirements are taken into account only in a second step where a specific version that can meet the non-functional requirements is selected and tailored further to meet the non-functional requirements.

Different tenants with different non-functional requirements may reuse the same NSD. The provider will reuse existing network services (NSDs) to configure and instantiate network services based on varying non-functional requirements.

Currently, the process is triggered by a new network service request from a tenant. From then on, the process can be executed automatically. It is also possible that the different phases of the process can be triggered by the OSS/BSS. Following NS on-boarding, the process can be suspended and resumes only when the OSS/BSS sends an instantiation and deployment request.

B. Towards Model-Driven Process Enactment

In model-driven engineering (MDE) and model-based engineering (MBE), software models can play a key role in the lifecycle management of a system [11]. Having an effective and efficient way of coordinating these models, maintaining consistency, and managing the propagation of changes across the models is crucial. As a step in that direction, it is important to model our domain and our processes using modelling languages that are well-established and have well-defined semantics. In our work, we use UML2.0 Activity Diagrams for this purpose. While at the moment, the PM is a means of modelling and documenting our process, the use of a such modelling language sets the pillars for model-driven process enactment or execution.

We intend to evolve the PM to a detailed FTG+PM model as defined in [9]. The FTG+PM language includes a *process model* (modelled using a subset of UML 2.0 activity diagrams) in which each activity is essentially a model transformation, and the *formalism transformation graph* (a form of megamodel) to support the PM. The framework is based on the notion of multi-paradigm modelling and uses metamodelling and model transformations as enablers. The use of such PMs or model transformation chains allows the enactment or automatic execution of the workflow defined in the process.

Having an underlying transformation chain also naturally leads to ensuring traceability requirements (not just at the stakeholder level but also at the software management level), since these chains explicitly model the relations between the steps of an MDE process [9].

VI. Related Work

We can summarise the contributions in the NFV literature as follows: (1) Studies of the NFV domain and the different functional modules in the NFV reference framework; (2) Investigations of research directions and risks associated with the current NFV framework; (3) Surveys on various standards, management/orchestration and monitoring tools; (4) Proprietary prototypes or PoC implementations, some of which rely on model-based approach; and (5) Solutions to the VNF placement problem.

For example, the authors in [12], [13], [14] give an overview of NFV and discuss its relationship with software defined networking (SDN) and cloud computing. The use of NFV technology and network slicing for software management at the network level is discussed in [15]. The main goal of these studies is to identify key future research directions in the NFV domain. Han *et al.* [13] discuss some NFV uses cases. Moreover, Mijumbi *et al.* [12] conclude that most of existing NFV industry solutions share vendor-specific resources hosted in the cloud without real support for flexibility, interoperability, orchestration and–or automation, which are the core requirements for NFV.

Chen *et al.* [16] propose to refine and implement the ETSI specification of NFV-MANO by realising a prototype of the underlying components and interfaces. As the first step towards realisation, the authors focus on the detailed analysis of the lifecycle management of the virtualised network functions (VNFs). The authors adopt a model-based monitoring solution, *e.g.*, Dell Foglight, for VNF lifecycle management, for example, auto-scaling [16]. Using their prototype implementation, the authors successfully deploy and scale a sample VNF. However, the authors limit their work only at the VNF-level and do not propose a process to complement the standards.

In another work, Xilouris *et al.* [17] outline an integrated NFV architecture designed and developed under T-NOVA EU project. The main goal of T-NOVA project is to realise the NFaaS (Network Functions as a Service) model by designing and implementing an integrated management architecture for automatic provision, management, monitoring, and optimisation of VNFs. However, T-NOVA project is concerned with VNFs only and a overall process for network service has not been proposed. Cisco provides guidelines for lifecycle management of Cisco network functions including registering, deploying, monitoring, scaling, and healing of VNFs as performed by the Cisco Elastic Services Controller (ESC), similar to a VNFM in the NFV framework [18]. On the other hand, Oracle develops a framework called *Oracle Communications Design Studio* [19] to design a network service (NS) and to support the NS orchestration. However, the NS design requires to create various framework-specific NS constituent resources and they do not follow the ETSI specifications.

Sahhaf *et al.* [20] consider different service compositions, *i.e.*, VNFs arranged in different ways with different VNFFGs and VLDs, and propose algorithms to select the optimal composition according to some criteria including resource demands, QoS, and available infrastructure resources. This work assumes the different compositions given as input and basically looks into the deployment as an *optimisation problem* as several other papers in the literature [21], [22]. Indeed, VNF placement is the most active research topic among academia interested in NFV.

Moreover, there are several white papers from industry that discuss proprietary proof of concept (PoC) implementation of their NFV architecture adopted from ETSI. For example, Intel [23] shows the benefits of having OSS/Orchestrator to deploy both the service configuration as well as a testing and assurance solution. Huawei [24] identifies major issues hindering the NFV. As mentioned in the paper, the major issue concerns interfaces and interoperability issues in management and orchestration in NFV. However, Huawei does

not discuss its NFV architecture. HP [25] briefly describes its implementation of the ETSI NFV reference architecture—HP NFV Director. The key features of HP NFV Director include its capability of using VNF Manager functionality and multi-vendor support. Finally, Cisco [26] defines and develops its proprietary platform to address the management and orchestration requirements for NFV framework [6].

VII. Conclusion and Future Work

In this paper, we identified some limitations and open issues in the NFV reference architecture. We proposed a model-based design and deployment process by establishing the core activities required to carry out NS design, configuration, and deployment along with their inter-dependencies and execution order. We defined the dependencies and the workflow explicitly in terms of control-flow and object-flow between these activities in a Process Model (PM) described using UML2.0 Activity Diagram. The PM also associates the activities to the relevant functional blocks in the NFV reference framework to clearly show the interactions between the various actors participating in the process and the tailoring and specialisations of the key NFV artifacts. The proposed process is compliant with the current standard. Moreover, the process includes an external activity for the automated generation of deployment templates which will speed up further network services provisioning.

We are currently working on detailing out the NS instance lifecycle management activity by including specific activities for *SLA Management* and *Dynamic Reconfiguration* along with the associated flows. Moreover, we intend to define the Process Model for each of the activities in our PM by detailing the actions along with the flow that are part of each activity. As future work, we plan on working on the automatic execution of the proposed workflow.

Acknowledgments: This work is partly funded by NSERC and Ericsson, and carried out within NSERC/Ericsson Industrial Research Chair in Model Based Software Management.

References

[1] T. Velte, A. Velte, and R. Elsenpeter, *Cloud Computing, A Practical Approach*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2010.

[2] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng *et al.*, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.

[3] *Network Functions Virtualisation - Architectural Framework: ETSI GS NFV 002 V1.2.1*, ETSI Std., December 2014.

[4] *Network Functions Virtualisation (NFV) - Virtual Network Functions Architecture: ETSI GS NFV-SWA 001 V1.1.1* , ETSI Std., December 2014.

[5] *TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0*, OASIS Committee Specification Draft 03, March 2016. [Online]. Available: http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html

[6] *Network Functions Virtualisation - Management and Orchestration: ETSI GS NFV-MAN 001 V1.1.1*, ETSI Std., December 2014.

[7] C. Chappell, "NFV MANO: What's Wrong and How to Fix It," February 2015. [Online]. Available: http://getcloudify.org/brochures/Heavy%20Reading%20NFV%20MANO%20Cloudify%20Snapshot.pdf

[8] M. Abbasipour, M. Sackmann, F. Khendek, and M. Toeroe, "A Model-Based Approach for User Requirements Decomposition and Component Selection," in *Formalisms for Reuse and Systems Integration*, T. Bouabana-Tebibel and H. S. Rubin, Eds. Springer International Publishing, 2015, pp. 173–202.

[9] L. Lúcio, S. Mustafiz, J. Denil, H. Vangheluwe, and M. Jukss, "FTG+PM: An Integrated Framework for Investigating Model Transformation Chains," in *SDL 2013: Model-Driven Dependability Engineering: 16th International SDL Forum, Montreal, Canada, June 26-28, 2013. Proceedings*, F. Khendek, M. Toeroe, A. Gherbi, and R. Reed, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 182–202.

[10] *OMG Unified Modeling Language (OMG UML), Version 2.4.1*, Object Management Group Std., Rev. 2.4.1, August 2011. [Online]. Available: http://www.omg.org/spec/UML/2.4.1

[11] R. France and B. Rumpe, "Model-based Lifecycle Management of Software-intensive Systems, Applications, and Services," *Software & Systems Modeling*, vol. 12, no. 3, pp. 439–440, 2013.

[12] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, January 2016.

[13] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network Function Virtualization: Challenges and Opportunities for Innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[14] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and Orchestration Challenges in Network Functions Virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, January 2016.

[15] "Network Functions Virtualization and Software Management," White Paper, Ericsson, December 2014. [Online]. Available: www.ericsson.com/res/docs/whitepapers/network-functions-virtualization-and-software-management.pdf

[16] Y. Chen, Y. Qin, M. Lambe, and W. Chu, "Realizing Network Function Virtualization Management and Orchestration with Model-based Open Architecture," in *11th International Conference on Network and Service Management (CNSM '15)*. IEEE, 2015, pp. 410–418.

[17] G. Xilouris, E. Trouva, F. Lobillo, J. M. Soares, J. Carapinha, M. McGrath, G. Gardikis, P. Paglierani, E. Pallis, L. Zuccaro *et al.*, "T-NOVA: A Marketplace for Virtualized Network Functions," in *European Conference on Networks and Communications (EuCNC '14)*. IEEE, 2014, pp. 1–5.

[18] "Cisco Elastic Services Controller 2.0 User Guide," White Paper, CISCO, October 2015. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/net_mgmt/elastic_services_controller/2-0/user/guide/Cisco-Elastic-Services-Controller-User-Guide-2-0.html

[19] "Oracle Communications Network Service Orchestration Solution Implementation Guide, Release 1.1," White Paper, Oracle, July 2016. [Online]. Available: https://docs.oracle.com/cd/E71075_01/doc.11/e65331/toc.htm

[20] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Network Service Chaining with Efficient Network Function Mapping Based on Service Decompositions," in *1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–5.

[21] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic Virtual Network Function Placement," in *IEEE 4th International Conference on Cloud Networking (CloudNet)*, October 2015, pp. 255–260.

[22] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On Orchestrating Virtual Network Functions," in *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*, ser. CNSM '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 50–56.

[23] "Using NFV Orchestration to Collapse Service Deployment and Service Assurance Silos," White Paper, Intel, December 2015. [Online]. Available: https://networkbuilders.intel.com/docs/ETSI_Netrounds_WP_121615.pdf

[24] "Huawei Observation to NFV," White Paper, Huawei Technologies Co., 2014. [Online]. Available: www.huawei.com/ilink/en/download/HW_399662

[25] HP, "HP NFV Director - HP solution for NFV Orchestration and VNF Management," White Paper, August 2013. [Online]. Available: www8.hp.com/h20195/v2/getpdf.aspx/4AA5-1082ENW.pdf?ver=1.0

[26] "NFV Management and Orchestration: Enabling Rapid Service Innovation in the Era of Virtualization," White Paper, CISCO, 2015. [Online]. Available: www.cisco.com/c/en/us/solutions/collateral/service-provider/network-functions-virtualization-nfv/white-paper-c11-732123.html

ViTeNA: An SDN-Based Virtual Network Embedding Algorithm for Multi-Tenant Data Centers

Daniel Caixinha
INESC-ID Lisboa
Instituto Superior Técnico
Universidade de Lisboa, Portugal
daniel.caixinha@tecnico.ulisboa.pt

Pradeeban Kathiravelu
INESC-ID Lisboa
Instituto Superior Técnico
Universidade de Lisboa, Portugal
pradeeban.kathiravelu@tecnico.ulisboa.pt

Luís Veiga
INESC-ID Lisboa
Instituto Superior Técnico
Universidade de Lisboa, Portugal
luis.veiga@inesc-id.pt

***Abstract*—Data centers offer computational resources with various levels of guaranteed performance to the tenants, through differentiated Service Level Agreements (SLA). Typically, data center and cloud providers do not extend these guarantees to the networking layer. Since communication is carried over a network shared by all the tenants, the performance that a tenant application can achieve is unpredictable and depends on factors often beyond the tenant's control.**

We propose *ViTeNA*, a Software-Defined Networking-based virtual network embedding algorithm and approach that aims to solve these problems by using the abstraction of virtual networks. Virtual Tenant Networks (VTN) are isolated from each other, offering virtual networks to each of the tenants, with bandwidth guarantees. Deployed along with a scalable OpenFlow controller, *ViTeNA* allocates virtual tenant networks in a work-conservative system. Preliminary evaluations on data centers with tree and fat-tree topologies indicate that *ViTeNA* achieves both high consolidation on the allocation of virtual networks and high data center resource utilization.

I. Introduction

The creation of data centers allowed global access to huge computational resources, previously only available to large companies or governments. By renting the desired computational power, small companies (or even an individual) avoid large capital expenses. In current data center environments, a client can ask for a computational instance of various sizes, and the service provider assures levels of guaranteed performance (through an SLA) for that computational instance. This guarantee is possible due to the huge evolution of virtualization technologies. Nowadays, a hypervisor can control the behavior of the virtual machines (VMs) it hosts, ensuring that a VM cannot use more CPU than what it was requested (except when the hypervisor allows). In this way, tenants are not harmed by the misbehavior of the other tenants.

This simplicity of computational resources on demand has generated a lot of interest around the world. However, there are still a lot to improve in this area - considerably, the lack of network accounting in the renting of resources. Cloud providers do not offer network performance guarantees to their tenants. In fact, a tenant's compute instances or VMs communicate over the network shared by all tenants. Thus, the network performance that a certain VM can get depends on several factors including those outside the tenant's control, such as the network load on a given moment or the placement of that VM in the network. This is further aggravated by the oversubscribed nature of a data center network.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

Lack of guarantees in a shared communication medium leads to unpredictable application performance often at tenants' cost. Machine virtualization has a considerable impact on network performance, where virtualized machines often present abnormally large packet delay variations, up to hundred times larger than the propagation delay between the considered two hosts. Moreover, TCP and UDP throughput can fluctuate rapidly (in the order of tens of milliseconds) between 1 Gb/s and zero, which shows that applications will have a very unpredictable performance [1]. Tenant applications in the cloud and data centers are often data intensive, such as video processing, scientific computing, or distributed data analysis. Hence a fluctuation in tenant virtual bandwidth allocation may severely degrade the performance achieved by an application. With intermittent network performance, MapReduce[2] applications will experience harsh issues when the data to be shuffled amongst mappers and reducers is quite large.

Software-Defined Networking (SDN) [3] is an abstraction that decouples the control plane from the data plane consisting of forwarding hardware such as switches and routers. Hence, the control mechanism can be extracted from the network elements and logically centralized in the SDN controller. The controller creates an abstraction of the underlying network, and thereby provides an interface to the higher-layer applications. SDN controllers can be leveraged to create a Virtual Tenant Network (VTN) on top of the data plane. Each of the tenants is given isolation guarantees at network level, with an illusion of a dedicated virtual network. VTN can be leveraged in data center and cloud networks to ensure that SLAs are met with Quality of Service (QoS) guarantees from the bottom-most level.

Virtual network embedding [4] aims to completely vir-

tualize the network, providing performance isolation among tenants at the network level. So, virtual network embedding consists in the mapping of virtual networks (consisting of virtual nodes and links) onto the substrate network (consisting of physical nodes and links). Virtual network embedding is considered the main challenge in the implementation of network virtualization [5]. In the data center context, network virtualization is advantageous as it allows clients to define the desired network topology, which will be allocated within the infrastructure exactly as defined by the client. This enables seamless migrations of current or legacy networks to a data center environment, since the client defines the topology and the required guarantees (such as CPU or bandwidth between machines), which will be enforced by the embedding algorithm by placing the virtual network only where the resources are sufficient.

In this paper, we describe the design, implementation, and evaluation of *ViTeNA* virtual network embedding approach. *ViTeNA* leverages the global view of the network offered by SDN controllers for virtual network allocation for tenants in a data center network. This allows tenants to express their requirements in terms of bandwidth, which is then enforced through virtual networks. *ViTeNA* is designed as a scalable solution for data center environments. It further achieves high consolidation within the placement of virtual networks, and high utilization of the data center's physical resources - servers and network.

II. Background and Related Work

OpenFlow [3] is a southbound protocol and core enabler of SDN. Many SDN controllers such as OpenDaylight [6], ONOS [7], and Floodlight [8] have developed OpenFlow implementation in high-level languages. Supported by the Linux Foundation and many big players in the networking industry, OpenDaylight and ONOS have grown to be large and complex SDN projects, with various sub-projects and use cases. Floodlight remains fairly simple as a compact Java-based open source SDN controller.

OpenDaylight VTN offers virtual network provisioning, flow and QoS control over virtual network, and virtual network monitoring. However, OpenDaylight VTN focuses on network virtualization function; not on virtual network allocation guarantees. $MicroTE$ [9] uses OpenFlow as a framework to have centralized control over the data center and make routing decisions based on predictions of the traffic matrix. Current traffic engineering techniques do not apply well to a data center, as they are too slow to react to micro-congestions, and the reaction time must be under 2 seconds to be effective [9]. Hence, $MicroTE$ creates a hierarchical structure to make traffic measurements scalable with the data center size, having a centralized controller.

Unpredictability of network performance in data center environments is damaging to both tenants and service providers, since the tenant applications suffer from the unpredictability and the service provider can incur in avoidable revenue losses [10]. Oktopus [10] facilitates predictable networks, offering network guarantees to the tenants. SecondNet [11] utilizes a central unit that receives virtual network requests, and runs the embedding algorithm to process and allocate the virtual tenant network requests. But, unlike Oktopus, the routes calculated by the central node do not translate into routing rules to the switches, because in SecondNet the physical machines keep information about the routes of each VM it owns. Silo enables co-existence of tenants in a competitive environment for resources, though with a trade-off of reduced network utilization [12]. EyeQ offers a distributed transport layer for network performance isolation in multi-tenant environments [13].

Seawall [14] tackles the problem of fair bandwidth sharing and network performance isolation in data centers. It overcomes the lack of performance isolation at the network level by assigning weights to each entity (such as a VM, or a process inside a VM). Thus, Seawall devices a solution where the share of bandwidth obtained by the entity in each network link is proportional to its weight. Gatekeeper [15] shares some design ideas and goals with Seawall. However, it provides minimum bandwidth guarantees, by using Open vSwitch [16] in each server to control all the VMs within a server. Each VM has a virtual network interface card (vNIC), that connects to the Open vSwitch. A minimum receive bandwidth guarantee as well as a minimum send bandwidth guarantee is assigned to each VM. Minimum bandwidth guarantees are achieved using an admission control mechanism that limits the sum of guarantees to the available physical link bandwidth.

Heuristics based virtual network embedding algorithms have various specific objectives including low execution time. Survivable networks [17] aim to make the virtual network embedding with fast failure recovery times. Despite the differences in goals, all these algorithms focus on delivering bandwidth guarantees in a data center network. Server locality of the same virtual network embedding request is exploited and leveraged by these algorithms, to reduce the search space for a solution, thus reducing the algorithm execution time.

As virtual network embedding is an NP-hard problem, node mapping and link mapping phases are separated in typical heuristic-based algorithms. If the request is smaller than the VM capacity of the largest available server, upon a virtual network request, the system first attempts to do the node mapping by trying to accommodate everything inside a single server. If not, they try the servers on the same rack, followed by the adjacent racks, and so on, to minimize the distance. The choice of the first server rack to analyze varies from work to work, but it is either random or in a round-robin fashion. The link mapping phase only starts if the virtual network request completes the node mapping phase. If it does not, it can be put into a queue to be processed later or the request discarded to alert the client there are no resources available for their request.

This is a $greedy$ approach, since it picks the locally optimal choice at each branch in the road (i.e. it chooses the best solution that complies with the virtual network constraints). With this type of algorithms, the virtual networks also benefit

from the reduced number of hops in the substrate network, which will yield low latency (as low as possible, but with no guarantees) between the VMs communicating in the virtual network. Some works [11], [18], [10], [19], [20] follow this approach. Path splitting (i.e. bifurcated traffic) and migration of VMs can also be considered for small data centers [19], as this does not significantly reduce the search space in a reasonable time for data centers beyond a few hundreds of nodes. Chowdhury et al [4] formulate the virtual network embedding as a mixed integer problem and solve it as a linear program by relaxing the integer constraints. Extending the later advances in networking and SDN, *ViTeNA* aims to improve virtual network embedding, offering a virtual tenant network for multi-tenant data centers.

III. *ViTeNA* Approach for Network Allocation

We will describe *ViTeNA* virtual tenant network allocation in this section. Although *ViTeNA* can be deployed on any network research topologies, we will limit our focus on the traditional tree-like topologies (tree and fat-tree) since they are still the most used data center topologies [21].

Figure 1 depicts a sample deployment of data center network in *ViTeNA* approach. The OpenFlow controller is the core of *ViTeNA*, as it is responsible for running the virtual network embedding algorithm to map the requests on the substrate network, and programs the switches to deploy the requested virtual networks. Each switch acts merely as a packet forwarder, per the rules dictated by the controller. The controller has a connection to every switch in the network, represented by the thinner dotted lines that are from the control plane in Figure 1. Though the data flows and control flows are differentiated using different lines in the diagram, the control plane does not necessarily need dedicated connections; it can use the same physical links of the data plane.

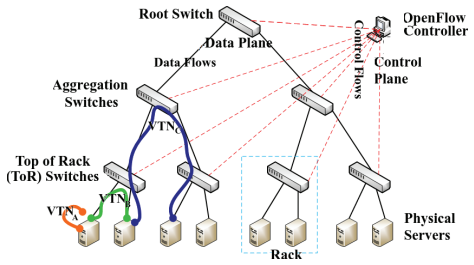


Fig. 1. Deployment landscape of *ViTeNA* with a tree topology.

ViTeNA network embedding algorithm tries to allocate the requests on the smallest available subset of the substrate network, like the other existing virtual network embedding approaches. It thus aims to maximize the proximity of VMs belonging to the same tenant, which results in minimizing the number of hops between those VMs. This is advantageous for two reasons: i) with less hops, the delay is generally reduced; and ii) keeping the VMs close (e.g. in the same rack) relieves the bandwidth usage in the upper links of the tree, where the bandwidth is scarcer in a data center [21]. With this approach, we will be able to accept more virtual network requests, since the core links will not be so likely to become the bottleneck of the data center.

Figure 1 shows the placement of three virtual networks in tree topology with depth equal to 3 and fanout equal to 2. The virtual network of tenant A (VTN_A) represents the best possible case where all the VMs of the virtual network can be mapped on the same physical server. In this case, there is no usage of the network (which saves bandwidth for future requests), and the bottleneck of the virtual network is only the speed within the server. In the virtual network of tenant B (VTN_B) the request could not be mapped to a single server, and hence it uses another server belonging to the same rack to accommodate the entire request. The virtual network of tenant C (VTN_C) shows a case where the virtual network could not be mapped in the same rack, and must use a server on the adjacent rack. As we can see, the bandwidth of the links on the top of the tree is only used in the worst cases (i.e. when the request is large or the data center is operating near saturation).

Besides the network embedding algorithm that ensures that the network can provide the bandwidth guarantees requested by each tenant, *ViTeNA* exploits the centralized information in the controller to provide fair bandwidth sharing (i.e. work-conservation) among tenants (non-existent in network embedding systems) and incremental consolidation of virtual network requests. Fair bandwidth sharing is achieved by instructing every switch used by a virtual network (which is determined by the embedding algorithm) to create a new queue for that virtual network. The queues in OpenFlow are used to provide QoS guarantees (in this case, bandwidth). Incremental consolidation of virtual network requests is enforced in the network embedding algorithm itself, choosing the location of a virtual network according to a best-fit heuristic on the VM placement. To do this, the algorithm leverages the current state of the network and the physical servers available to the OpenFlow controller.

A. Software Architecture

Figure 2 depicts each component of *ViTeNA* as well as the most important interactions between them. Mininet [22] open source network emulator has been used to emulate the data center networks, as it considered the de facto standard of OpenFlow emulators [23]. *ViTeNA* could be ported to a physical data center with a few or no changes in code, from the current Mininet-based emulations. The only exception is in the Linux Process, which in a real scenario would be running a hypervisor to manage the VMs inside that host. In this paper, this is simplified to an operating system managing processes, where each process will simulate a VM. We assume all the physical servers have equal CPU, so that in a virtual network request a tenant asks for a percentage of a CPU (instead of a CPU with a certain frequency).

Information Flow: The tenant expresses its demands in a virtual network request in an XML configurations file. This includes defining the number of VMs required (and the percentage of CPU of each one), as well as the bandwidth required between the VMs that will be connected (expressed in

MBit/s). This request is fed into the virtual network embedding algorithm that is running in the OpenFlow controller. Upon receiving the request, the embedding algorithm contacts the network information manager to get the current state of the network. Based on this state, the algorithm determines (if the request is accepted) where this virtual network will be allocated. As all the requests are processed by the controller, this updated view of the network involves zero control messages over the network (both to switches and hosts), since the controller just should update this information when it processes a new virtual network request.

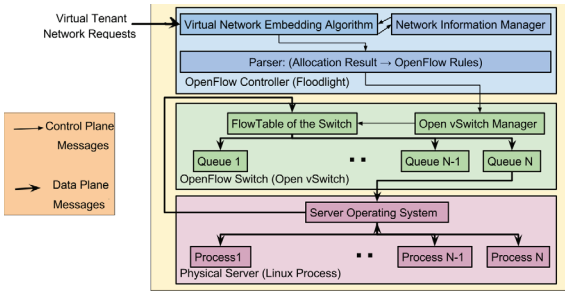


Fig. 2. Software architecture of *ViTeNA*.

The controller then translates the output of the algorithm (i.e. the affected switches and hosts) to OpenFlow rule(s) to reprogram the switch(es). Upon receiving this message, the Open vSwitch manager creates a new queue for this virtual network, with the assured bandwidth present in the received message. It further installs a new rule in the switch's flow table to forward packets from a certain VM to the newly created queue. Hence, there will be a queue for each pair of linked VMs in a virtual network. Thus, a VM can use more bandwidth than its minimum when the link is not throttled. Moreover, the sharing of bandwidth between queues is made fairly according to the minimum bandwidth a queue has (a queue with a higher minimum bandwidth will use more spare bandwidth). Thus, the resource usage is maximized, since the tenants share unused bandwidth fairly, but at the same time get their minimum bandwidth guarantee when the network is saturated.

The VMs are represented by and implemented as Linux processes. To simulate tenant workloads, each process runs a traffic generator. Upon the necessary configurations, each process (i.e. VM) can communicate with other processes on the same virtual network, using either the operating system (in case the processes are on the same server), or contacting its adjacent switch, which will use the flow table to check to which queue it should forward this solicitation (in case the processes are on different servers). In this paper, we will only focus on communication inter-server as the intra-server communication is a responsibility of the hypervisor.

B. Virtual Network Allocation

As the core procedure of *ViTeNA*, algorithm 1 aims to guarantee that every VM pair in the virtual network request gets at least the requested bandwidth; and, to consolidate the virtual networks on the least amount of physical resources possible. Instead of making a consolidation algorithm that runs periodically, *ViTeNA* performs the consolidation incrementally at each request, merging the network embedding and the consolidation algorithms. Hence it avoids heavy migrations of VMs between servers, which would stop temporarily the work of those VMs and possibly generate a lot of control and data traffic. Periodic execution of the algorithm would diminish the controller performance. *ViTeNA* thus avoids performance degradation while processing the virtual network requests.

Algorithm 1 *ViTeNA* Virtual Network Embedding

```
1: procedure isNetworkRequestAccepted(VNR)
2:     totalVMLoad ← getVMLoadFromVNR(VNR)
3:     highestCPUAvailable ← getMostFreeCPU()
4:     if (totalVMLoad < highestCPUAvailable) then
5:         appropriateList ← findAppropriateList(totalVMLoad)
6:         for all (server in appropriateList) do
7:             serverCPUAvailable ← getCPUAvailable(server)
8:             if (totalVMLoad < serverCPUAvailable) then
9:                 allocVirtualNetwork(VNR, server)
10:                Return True                          ▷ Request is accepted
11:    else
12:        server ← getMostFreeServer()
13:        firstServer ← server
14:        sortedVNR ← sortVNRByBWDemands(VNR)
15:        [preAllocatedVMs, remainingVMs] ← splitRequest(sortedVNR, highestCPUAvailable)
16:        VMsAlreadyAllocated ← (preAllocatedVMs, server)
17:        preAlloc(preAllocatedVMs, server)
18:        while (True) do
19:            server ← getNextServer(server)
20:            if server.isFirstServer() then
21:                cancelAllPreAllocs()
22:                Return False                         ▷ Request is not accepted
23:            CPUAvailable ← getCPUAvailable(server)
24:            [preAllocatedVMs, remainingVMs] ← splitRequest(sortedVNR, CPUAvailable)
25:            BWDemands ← calcSumOfDemands(sortedVNR, VMsAlreadyAllocated, preAllocatedVMs, server)
26:            linksResidualBW ← calcResidualBW(VMsAlreadyAllocated, server)
27:            if (BWDemands > linksResidualBW) then
28:                clearLastSplitRequest()
29:                continue
30:            else
31:                preAlloc(preAllocatedVMs, server, BWDemands)
32:                VMsAlreadyAllocated ← VMsAlreadyAllocated + (preAllocatedVMs, server)
33:                if (remainingVMs.isEmpty()) then
34:                    allocAllPreAllocs()
35:                    Return True                      ▷ Request is accepted
```

The algorithm is divided into two base cases depending on the virtual network request: i) when it fits in one physical server, and ii) when it needs to be spread across multiple servers. This is the first check made, comparing the total CPU load requested with the highest CPU available at the moment (line 4). If the request fits in one server, we want to find the server with the least free CPU that fits the request (i.e. best-fit). Once the server is found, we allocate this request on it (which includes updating the network state with this new request). To find the best-fit server, we first find a sub set as the search space (line 5). We will maintain 10 sets: the first keeps the servers with 0 to 10 % of CPU free, the second the servers with 10 to 20 % of CPU free, and so on. This reduces the search space for the appropriate physical server, which in a large data center environment can reduce the run time of the algorithm considerably.

If the request does not fit in one server, we sort the request by decreasing bandwidth (line 14), and allocate as much VMs as possible in the freest server on the entire data center. In this way, the most consuming demands are on the same physical server, which significantly relieves the load on the network (and thus we can accept more virtual network requests). After pre-allocating (since the request can be rejected) what is possible on the freest server, we try to allocate the rest on adjacent servers. In line 19, the getNextServer(*server*) function returns the servers on the same rack of *server*, then the servers on an adjacent rack, and so on. Next we check if we already tried on every server of the data center (line 20), and reject the request if so, cancelling all the pre-allocations.

In the next lines (23-26), we see if the server we are checking has connection(s) with sufficient bandwidth (as defined in the request) to the other server(s) already pre-allocated in previous iteration(s). If it does not have enough bandwidth, we try on the next server (lines 27-29). If it does, we pre-allocate the VMs mapped onto this server. Finally, we check if the set remaining VMs to be allocated is empty (lines 33-35): if it is, we allocate all the pre-allocations (i.e. commit) and return *True*; if it is not, we move on to the next server to allocate the remaining ones.

We expect to have a high consolidation (and consequently low fragmentation) of VMs within the servers, since the algorithm either allocates a whole server (if the request does not fit one server), or finds the best-fit server (if the request fits in one server).

IV. Implementation

Floodlight 1.1 has been leveraged as the core SDN platform for *ViTeNA* implementation. Mininet 2.2.1 and Open vSwitch 2.3.1 were used in emulating data centers with various topologies. The Floodlight controller is based on an event driven architecture. Hence, for a module to receive an OpenFlow PacketIn message, the module must subscribe for this type of messages. When the controller receives an OpenFlow message from a switch it will dispatch that message to all modules that have subscribed for that specific message type.

If the controller receives multiple messages from one or more switches, these messages are enqueued for dispatching because the controller only supports dispatching one message at a time. This means that the performance of the controller is dependent on the time it takes to process a message in each individual module, as the processing time of a single message is equal to the sum of all the processing time done in each individual module.

To add a new module to the Floodlight controller, a new Java package must be created directly in the code base of the controller. *Floodlight default properties file* defines which modules are launched when Floodlight is started. For a custom-made module to start up, its path should be added to the properties file. When the controller starts, it will start the modules, and set references between them per the inter-module dependencies. These dependencies are defined by implementing the appropriate methods (according to which dependencies one wants to set) of the `IFloodlightProviderService` interface. We modified the *link discovery* module of Floodlight. *ViTeNA* also includes 3 additional modules into Floodlight - i) Multipath Routing, ii) Queue Pusher, and iii) Virtual Network Allocator modules. We will now dive into the details of each of these four modules of *ViTeNA*.

A. Link Discovery Module

The controller needs to place the VMs of a request in physical servers that have links with enough bandwidth to accommodate what is requested in the XML file. Thus, each link should know how much free bandwidth it possesses. However, in Floodlight's original implementation, the Link object does not have such information. Hence, it was necessary to extend the link discovery module, so that each link would be aware of its spare bandwidth. This consisted in adding a new field to the Link class, and defining the appropriate *getters* and *setters* to configure this field.

The link's bandwidth isn't set automatically as one would expect (i.e. through the LLDP packets this Module sends). Mininet emulates all the virtual links with a bandwidth of 10 Gb/s, even if we configure it to have a lower bandwidth. We had to work around this, and we did it by initializing all link bandwidth when the controller starts, according to what is defined in a start-up XML file. To ease the process of running our controller, this XML file is automatically generated when the Mininet topology is created, according to the parameters in the Mininet script.

B. Multipath Routing Module

Floodlight's original routing module provides a Java API to use. Its $getRoute()$ method calculates the route between two endpoints by applying the Dijkstra's algorithm [24] to the graph that contains the network topology. This means that it always gives the shortest path between two nodes in the graph. Since we want to maximize the number of allocated virtual networks by the controller, we want to test every possible route between two endpoints. The shortest path between two endpoints may not have enough bandwidth to accommodate a certain request, and a longer path may have that required bandwidth. By choosing to use more than just the shortest path can turn many otherwise rejected requests into accepted ones.

As of our implementation, the multipath routing module registers itself as a receiver for events of the type `topologyChanged`. By receiving these events, the module builds a graph, adding and removing links or hosts as the events dictate. This graph represents the network topology. Having this graph, one must only apply a search algorithm on top of it to find the paths between two nodes. We use a Depth-First Search algorithm [25] to compute all possible paths between a pair of nodes.

C. Queue Pusher Module

ViTeNA queue pusher module is responsible for providing an API to create queues in Open vSwitches. Queues in Open vSwitch are created using the OVSDB protocol [26]. The

queue pusher module creates queues, with only the Assured Rate configured (called `min-rate` in the OVSDB command) and no Ceil Rate (or `max-rate`) configured.

The queue pusher module uses the `ovs-vsctl` utility that comes pre-installed with the Open vSwitch, to create a new QoS entry and a new Queue below that QoS entry for each Queue the controller wants to create. When creating Queues, each one gets assigned an identifier, which should be unique per switch. The traffic will be directed to the Queue by matching this identifier, since the *enqueue* action receives it as argument.

D. Virtual Network Allocator Module

The allocation algorithm runs in the virtual network allocator module. When the request fits in one server, the sum of CPUs requested can be accommodated in the same physical server. In this case, this module should merely update the *global* data structures, with the information from the *local* data structures, that was gathered from the XML file.

When the request does not fit in one server, the request should be divided among two or more physical servers, and this module will start by sorting the links of the virtual network request in descending order (since the tenant can provide the XML file in any order). Then, it will allocate as much VMs as possible in the server that has the most CPU available. After that, it will try to allocate the remaining VMs in the neighbors of this server. It will start from the server adjacent to this one, and it will continue this logic until there are no remaining VMs.

In each iteration of going to the neighbor of the server with the freest CPU, this module uses the multipath routing module and the link discovery module. Every time it advances to an adjacent server, it uses the multipath routing module to get all paths between this server and the ones that already have allocated VMs. Upon getting these paths, the virtual network allocator module will use the link discovery module to check each link of each path, to make sure that those links have enough bandwidth to provide the guarantees required by this tenant's request.

Once all the VMs have a physical server assigned (assuming a request where this happens), the module knows this request is going to be accepted. So, it is necessary to translate this request's results into real network rules. This module uses the queue pusher to create the required queues on each OpenFlow switch and the necessary OpenFlow rules, returning `True` following that.

V. Evaluation

ViTeNA was evaluated on a computer with Intel® Quad-Core i7 870 @ 2.93 GHz processor, 12 GB DDR3 @ 1333 MHz RAM, and 450 GB Serial ATA @ 7200 rpm hard disk, on Ubuntu 14.04.3 LTS (Linux Kernel 3.13.0). The controller processes virtual network requests. We stop an experiment when the controller returns `False` to an allocation, as that means it cannot allocate any more virtual networks. Each experiment is run a thousand times to get the mean and variation of the results. To generate our dataset, we produced virtual network requests (XML files) where: a VM asks for a CPU that is generated randomly (using a uniform distribution) between 0.1 and 5%; the connections between VMs are also randomly generated (with a uniform distribution as well) between 0 and 10 Mbit/s. For each size of the virtual network requests (i.e. number of VMs in it), which we defined as going from 2 to 40, we generated 10000 virtual network requests.

A tree topology (depth = 3; fanout = 5) with 125 servers, which entails 31 switches and 155 links, and a fat-tree topology (factor k = 32, i.e. switches consist of 32 ports) with 128 servers, which entails 160 switches and 384 links, were emulated with Mininet for the evaluations.

A. Scalability to Data Center Environments

First, we evaluated the scalability of *ViTeNA* in data center scale. To this end, we measured the time it takes to process each virtual network request using the method `currentTimeMillis` from the controller Java API. Figure 3 depicts the results obtained with tree topology.

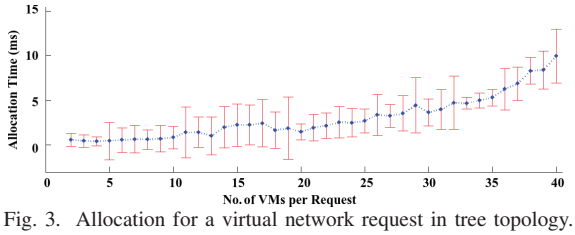


Fig. 3. Allocation for a virtual network request in tree topology.

Processing time less than 5 ms was observed for up to about 25 VMs in a request. SecondNet [11] achieves 10 ms in requests with 10 VMs, which is twice the processing time in requests with less than half of the VMs. *ViTeNA* consumes about 10 ms to process requests with 40 VMs. It should be noted that a request with 40 VMs is almost one third of the number of physical servers in the network. Even in these conditions, processing time did not grow abruptly, indicating the high scalability of *ViTeNA*.

Figure 4 depicts the allocation time for fat-tree. It can be noticed that the processing time using fat-tree topology is higher than the one observed for tree topology. Fat-tree peaks at around 15 ms, which is 5 ms more than that is observed with tree topology.

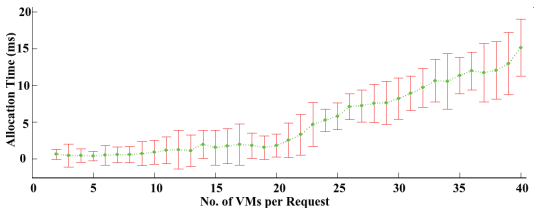


Fig. 4. Allocation for a virtual network request in fat-tree topology.

As fat-tree has a lot more links and switches than the tree topology, there are more paths between any two endpoints. Thus, the higher processing time can be explained by the extra work controller had to perform by checking more routes. Hence the extra processing time was not wasted, as with the fat-tree we received a total of 15688 accepted requests, *versus* 15055 with the tree topology.

B. High Consolidation

As *ViTeNA* takes server locality into account, allocating the VMs of a virtual network as close as possible, next we measured how consolidated the virtual networks are. A low number of hops leads to a low latency in the communication between the VMs of a virtual network. This metric is calculated by counting the numbers of physical servers in a virtual network allocation. The results of using a tree topology are depicted in Figure 5.

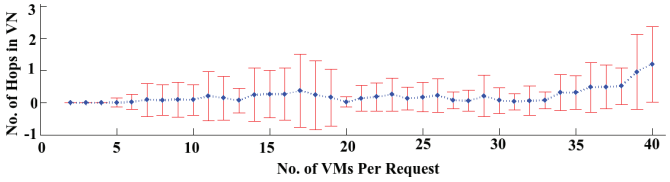


Fig. 5. Avg. number of hops in a virtual network in tree topology.

Figure 5 indicates that *ViTeNA* offers high consolidation of the virtual networks, since the average is almost always near zero. It starts out equal to zero up to 5 VMs per request, then the average is almost the same but the variance increases, meaning most of the virtual networks do not have any hop, but some do. It keeps this behavior along the line, with the average increasing less than linearly. On 40 VMs per request we get an average number of hops close to 1. 40 VMs in a request is significant, because the evaluated data center network has 125 servers, and still the virtual networks only needed one hop on average.

The average number of hops using a fat-tree topology is shown in Figure 6. Fat-tree exhibits a similar pattern to the one observed in tree topology case, and high consolidation is achieved in fat-tree topology as well. As we process more requests with this topology, we notice a higher average number of hops as well as the standard deviation. The extra requests we get with fat-tree topology are processed when the data center is near saturation (since we are stopping on the first rejected request in the tree topology, and with fat-tree topology we go further). Since the data center is near saturation, each request will more likely need a high number of hops, which explains the increase compared to the tree topology.

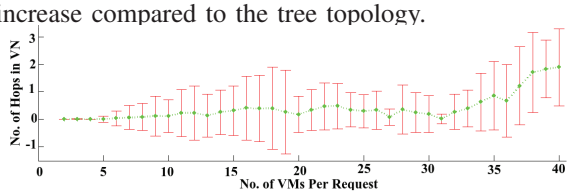


Fig. 6. Avg. number of hops in a virtual network in fat-tree topology.

C. High Resource Utilization

Resource utilization within the data center with *ViTeNA* was measured, by calculating the server and link utilization. Resource utilization is computed by calculating the utilization of the resources when the experiment stops, and dividing it by the full capacity. The resource utilization results using a tree topology are portrayed in Figure 7. As *ViTeNA* does a best-fit placement of the VMs within the servers, it was observed that most of the time *ViTeNA* achieves high server utilization. As *ViTeNA* already does an incremental consolidation, it does not need or have a consolidation algorithm running periodically in the controller.

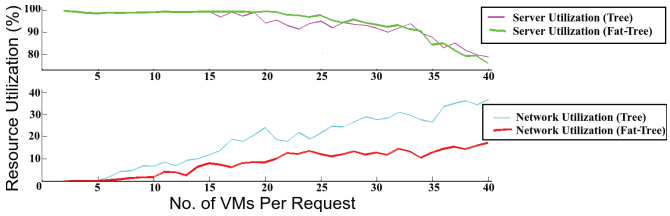


Fig. 7. Resource utilization in tree and fat-tree topologies.

The server utilization starts to drop when the number of VMs in a virtual network is around 20. This happens because with a request of this size (and larger), some of the VMs must be placed on different servers, which causes fragmentation of the CPU utilization by a server. This results in a lower server utilization. Obviously, the network utilization starts to grow when this happens, since we have more and more utilized links across the network. Moreover, the low network utilization is a result of getting all the servers full before we get some virtual networks that require link usage, as this is just a matter of which resource is exhausted first.

The results appear similar for both tree and fat-tree topologies. However, fat-tree allows the server utilization to remain high for longer (up to 25 VMs per request), where as in the tree topology it starts to drop at 15 VMs per request. This can be explained by higher number of requests served by fat-tree topology, since more requests allow to decrease the fragmentation in CPU usage across servers, which in turn causes the server utilization to increase. Nevertheless, fat-tree network utilization is significantly lower compared to the tree topology. This is due to the much higher number of links that fat-tree topology has (more than double of that of tree topology), which all add up to the denominator of this metric and causes it to decrease significantly.

D. Bandwidth Guarantees in a Work-conservative System

To evaluate the bandwidth guarantees in a work-conservative system, we created a topology with 8 hosts, where 4 hosts generate traffic towards the other 4. Each host generates 50 Mbit/s, and we simulate a 100 Mbit/s link using a queue with the `max-rate` parameter set to this value. We used the `iperf` tool to generate traffic with a constant bit-rate, each one generating traffic with a rate of 50 Mbit/s. We used a constant bit-rate to make sure that changes we see in the rate on the receiver side are due to the network changes and not from changes in the sending side. Figure 8 shows the graph generated accordingly.

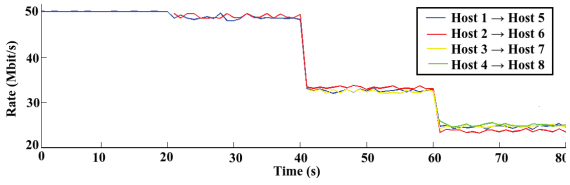


Fig. 8. Throughput achieved by multiple hosts sharing a single link.

In the beginning ($t = 0s$), only *Host 1* is generating traffic with the bit-rate stated above, and with *Host 5* as destination. As *Host 1* is the only one doing so, it gets the full bandwidth that it requested - 100 Mbit/s. This goes on until $t = 20s$, when *Host 2* starts to generate traffic towards *Host 6*, and there are two hosts generating traffic at 50 Mbit/s, which is the *link capacity* (i.e. the `max-rate` allowed by *Switch 2*). Each host on the right gets practically the bandwidth that its correspondent is generating; but we can already see the action of *Switch 2*, portrayed by the irregularities in both lines.

When we reach $t = 40s$, *Host 3* starts to generate traffic to *Host 7*. Now, the sum of the traffic generated by the hosts exceeds the *link capacity*', which will cause dropped packets. However, each host gets more than its assured bandwidth (25 Mbit/s), as the three of them divide the link, each one getting about 33 Mbit/s. Finally, at $t = 60s$, *Host 4* begins to generate packets towards *Host 8*. Now, the 100 Mbit/s link is divided by the four hosts, and each one gets about its assured bandwidth. Thus *ViTeNA* offers bandwidth guarantees to the tenants, while they utilize more resources when the resources are abundant.

VI. Conclusions and Future Work

Current data centers lack network performance guarantees, since all tenants interchangeably share the network. This makes the performance of tenant applications unpredictable, since it depends on factors outside of its control. This unpredictability severely prevents a wider cloud adoption, as many cloud use cases require network performance and isolation guarantees. These problems are solved using the abstraction of virtual networks. Virtual Tenant Networks (VTN) are isolated from each other, providing performance guarantees. Virtual network embedding algorithms attempt to solve this NP-hard problem of an efficient tenant network resource allocation.

ViTeNA is a virtual network embedding approach that extends an OpenFlow SDN controller to allocate virtual networks with bandwidth guarantees in a work-conservative system, providing a QoS-aware multi-tenanted data center. Evaluation on tree and fat tree topologies confirm that *ViTeNA* offers, 1) low execution time, 2) high consolidation on the allocation of virtual networks, and 3) high resource utilization of the data center resources. As a future work, *ViTeNA* should be extended for reliability and isolation guarantees, in addition to efficient network allocation.

Acknowledgements: This work was supported by national funds through Fundação para a Ciência e a Tecnologia with reference UID/CEC/50021/2013 and a PhD grant offered by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC).

References

[1] G. Wang and T. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 783–791.

[5] A. Fischer, J. F. Botero, M. Till Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 4, pp. 1888–1906, 2013.

[6] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 1–6.

[7] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[8] R. Wallner and R. Cannistra, "An sdn approach: quality of service using big switch's floodlight open-source controller," *Proceedings of the Asia-Pacific Advanced Network*, vol. 35, pp. 14–19, 2013.

[9] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: fine grained traffic engineering for data centers," in *Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies*. ACM, 2011, p. 8.

[10] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 242–253.

[11] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th international conference on Emerging networking experiments and technologies*. ACM, 2010, p. 15.

[12] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: predictable message latency in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 435–448, 2015.

[13] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, A. Greenberg, and C. Kim, "Eyeq: practical network performance isolation at the edge," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 297–311.

[14] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: performance isolation for cloud datacenter networks," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. USENIX Association, 2010, pp. 1–1.

[15] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks." in *WIOV*, 2011.

[16] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer." in *Hotnets*, 2009.

[17] M. R. Rahman, I. Aib, and R. Boutaba, "Survivable virtual network embedding," in *NETWORKING 2010*. Springer, 2010, pp. 40–52.

[18] T. Benson, A. Akella, A. Shaikh, and S. Sahu, "Cloudnaas: a cloud networking platform for enterprise applications," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 8.

[19] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[20] M. F. Zhani, Q. Zhang, G. Simona, and R. Boutaba, "Vdc planner: Dynamic migration-aware virtual data center embedding for clouds," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 18–25.

[21] K. Bilal, S. U. Khan, J. Kolodziej, L. Zhang, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, and D. Chen, "A comparative study of data center network architectures." in *ECMS*, 2012, pp. 526–532.

[22] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[23] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[24] S. Skiena, "Dijkstra's algorithm," *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pp. 225–227, 1990.

[25] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[26] D. Palma, J. Goncalves, B. Sousa, L. Cordeiro, P. Simoes, S. Sharma, and D. Staessens, "The queuepusher: Enabling queue management in openflow," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*. IEEE, 2014, pp. 125–126.

A Scalable Peer-to-Peer Control Plane Architecture for Software Defined Networks

Kuldip Singh Atwal
Department of Computer Science
University of Central Florida
Orlando, FL USA
Email: kuldip@cs.ucf.edu

Ajay Guleria
Computer Centre
Panjab University
Chandigarh, India
Email: ag@pu.ac.in

Mostafa Bassiouni
Department of Computer Science
University of Central Florida
Orlando, FL USA
Email: bassi@cs.ucf.edu

***Abstract*—Control plane scalability is one of the major concerns in Software Defined Networking (SDN) deployment. Although the centralization of the control plane by decoupling it from the data plane facilitates ease of network management, however, it introduces new challenges. One of these challenges is to maintain performance, consistency, and scalability while minimizing the corresponding overheads. In this paper, we propose an architecture that allows the control plane to evolve at a hyper-scale level as well as address important performance and reliability issues. A hierarchical control plane architecture with peer-to-peer communication among logically distributed controllers is designed with the goal of achieving optimum performance and consistency gains while mitigating overheads. A root controller is deployed at the top layer of the hierarchy to maintain global network view. The proposed model is helpful in improving network robustness against failures and supporting a desired level of reliability. To evaluate our model, we developed a realistic emulation platform using ONOS, FlowVisor, Mininet, and Open vSwitch. The proposed architecture is compared with earlier solutions and experimental results are presented to demonstrate the effectiveness of the proposed model.**

I. Introduction

Software Defined Networking (SDN) has gained very much attention from academia as well as the industry in recent times. It is envisioned to overcome the existing shortcomings of data center networks, access networks, and enterprise networks. Learning from prior experiences [1], separation of the control plane and the data plane is a crucial aspect for achieving SDN goals. Furthermore, centralized control, network programmability, and flow-based decision making are some of the other important features of SDN. Due to its salient features, SDN is being widely deployed in multi-datacenter and multi-domain networks by using OpenFlow protocol.

However, there are many challenges still need to be addressed. The control plane scalability is one of the prominent challenges among others. The issue of scalability escalates even further if we consider performance and robustness of the network, which are generally the prerequisites of any realistic network. Therefore, it becomes imperative to address the control plane scalability in a systematic manner by considering the other important aspects of a pragmatic network. In this paper, we propose a model to deal with scalability, robustness, and performance of SDN control plane architecture. Although a single controller can handle sufficiently large number of requests with an acceptable average response time [2], still there are other factors that require multiple controllers deployment [3], such as:

- Geographically wide distribution of the network.
- High availability and low response time requirements for QoS or real-time services.
- Handling bottleneck of the single point of failure.
- Partitioning large-scale networks (e.g. data center, enterprise networks) into many sub-networks, that can be controlled separately.
- Management of segregated inter-networks by different proprietary domains.

There are multiple ways to deal with the scalability of the control plane. First, the performance of the physically centralized control logic (i.e single controller) can be increased by deploying more hardware resources or performing some optimization techniques for performance enhancements up to a certain level [4]. Second, the overall workload of the control plane can be alleviated by minimizing the set of operations performed by it and devolving some functionality to other components [5]. Third, multiple controllers can be deployed that form a physically distributed or logically centralized control plane [6]. Given the earlier reasons for requirements of multiple controllers, we chose the last option to address the control plane scalability issue; therefore, other approaches are out of scope. Further, multiple controllers could be deployed in a fashion such that the control plane is fully decentralized and physically distributed [7], or logically centralized [8]. Our approach maintains logically centralized but physically distributed control plane.

Following are the major contributions of this paper:

- We propose highly scalable control plane architecture for SDN, that can be adopted to meet optimum performance demands of enterprise and data center networks.
- Consistency of the network state is handled very efficiently while mitigating the corresponding overheads.
- The proposed model allows the network to be configured dynamically as per desired level of robustness requirements.
- An integrated emulation platform is developed to eval-

978-1-5090-3216-7/16/$31.00

uate the proposed model, and corroborative results are presented.

Rest of the paper is organized as follows. Related work is discussed in Section II. Section III presents the system model of the proposed architecture. Then, evaluation platform for the proposed scheme is described in Section IV, followed by an elaboration of results in Section V. Finally, Section VI includes the conclusion and future scope of the work.

II. Related work

Our model is motivated from Orion [9], which is a hierarchical control plane architecture for large-scale networks. Our proposal differs from Orion in many respects. First, the top layer of controllers is not distributed. Rather, a root controller is placed in the hierarchy to maintain the coherent network view while incurring minimum overheads. Secondly, communication channels among zone controllers are provided to serve information requirement of any controller at relatively local level. Finally, if any zone controller fails, load will be distributed either to existing neighboring zone controllers or to newly added controller, depending on the adaptive controller provisioning mechanism.

Zoning mechanism for hierarchical network optimization is proposed in [10], that does not consider coherent network abstraction at the application layer. Our model utilizes resources more efficiently by off-loading some functionality to the root controller at the top layer of the hierarchy. Dynamic controllers adaptability and controller-switch assignment/reassignment are proposed in [11], that uses distributed data store to maintain coherent network view and perform load adaptation among multiple controllers. However, load balancing can be handled more efficiently if it is controlled by a physically centralized entity; therefore, we delegate this functionality to the root controller. A distributed hierarchical control plane to improve scalability and service flexibility is proposed in [12]. However, it does not deal with failures, and our solution is more robust against failures at multiple layers. HyperFlow [13] is an event-based, logically centralized but physically distributed control plane architecture for OpenFlow. Although HyperFlow is resilient to network partitioning and component failures, however, it does not consider dynamic controllers adaptation and load balancing, which are crucial functionalities of data centers and other similar networks. Kandoo [14] proposes the two-layer hierarchy of controllers to achieve scalability. However, it also does not consider failure and adaptability of controllers. A formal model on SDN control plane is presented in [15], which shows that the hierarchical organization is required to achieve scalability and elasticity of any practically feasible network.

Most of the existing logically distributed control plane architectures either fall short of robustness against failures or incur the communication or state management overheads that directly hampers the performance. Furthermore, the intense demands of data center type networks are not taken into consideration, such as dynamic resources provisioning based on user demands, optimum utilization of the existing infrastructure to save cost. Therefore, contrary to the state of the art solutions, our proposal uniquely addresses the control plane scalability while making sure to provide robustness and handle overheads in a systematic way.

III. System model

Following are the main challenges for a logically centralized control plane architecture of SDN:

- The global network view has to be maintained coherently across multiple controllers in case of any component failure and dynamic network topology changes.
- The inconsistency and competing resources issues may emerge among multiple controllers that need to be resolved with the priority to achieve overall optimum state.
- In a multi-controller scenario with a global network view, it is important to have the state synchronization and reachability between all controllers.
- The communication overheads are inherently increased in the deployment of multiple components.
- Dynamic controllers adaptation is required for efficient resources utilization. For instance, on-demand resources provisioning is needed in data center networks to fulfill user demands and meet the Service-Level Agreements (SLAs).

Other than that, an effective failover mechanism is needed for resilience and robustness of the network. Considering these aspects, we designed a logically centralized but physically distributed control plane architecture for SDN. By virtue of its design, the proposed model is not strictly physically distributed, since some functionality of the control logic is handled by the physically centralized controller. Figure 1 depicts the hierarchical model of our proposed system. The hierarchy is formed in such a way that the relatively local events are handled by the zone controllers and global events are handled by the root controller. The idea is to distribute the load among controllers, along with maintaining the coherent network state, while minimizing the communication and other overheads that contributes towards performance gains.

The root controller is mainly responsible for managing the global network view and some other network-wide functionalities, while the zone controllers directly control the underlying physical devices (i.e forwarding plane) via OpenFlow protocol. Furthermore, the communication channels are established between zone controllers for peer-to-peer communication. These channels are used to share any kind of information among zone controllers that is not available to a particular controller. Fetching the required information from neighboring controllers contributes to minimizing the latency that can be higher if the information is requested from the root controller at the top layer. Also, due to localized information sharing, the peer-to-peer interaction reduces the communication overheads in the network.

As shown in the diagram, the forwarding plane is formed by partitioning physical resources in zones. The zones can either be created as per segregation requirements of geographically distributed sub-networks, or as per multiple proprietary

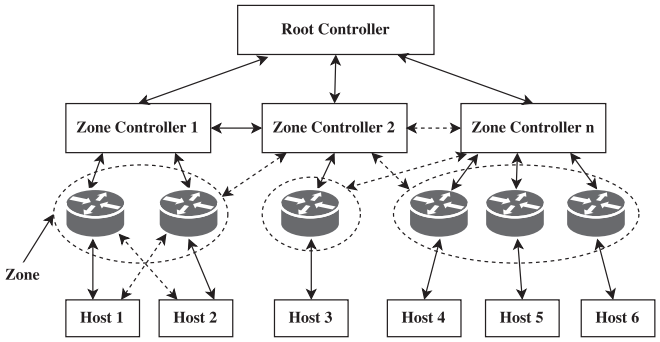


Fig. 1. Hierarchical SDN architecture.

domain management services. The zones can also be configured for the purpose of evenly distributed load management and optimum resources utilization of a large enterprise. The reliable TCP connections are employed for peer-to-peer communication among zone controllers as well as interaction with the root controller at the top layer. The solid arrows in the diagram represent the established connections among various components, while the dotted arrows highlight the provision to add multiple connections for backup purpose.

The next major task is to categorize the roles and responsibilities of the root controller and the zone controllers. In this aspect, our main focus is to delegate functionalities so that the consistent network view is maintained without many overheads, and service requests are served as per proximity of the components. With these goals, the following subsections elaborate functionalities of the control layers and describe the modules defined to implement such functionalities.

A. Root Controller

Below are the main responsibilities of the root controller:

- Maintain the globally consistent network view and share it to the zone controllers, application layer, and other services.
- Configuration of network components, such as zone controllers, switches.
- Dynamic provisioning of the zone controllers for average resources utilization, and failover mechanism enforcement in case of failures.
- Perform controller-switch assignment/mapping and initiate the switch migration when needed.
- Network statistics collection via zone controllers.
- Rules generation for network-wide policies enforcement.

Following are the modules defined to implement these tasks of the root controller:

Storage module: The network state information is stored in the storage module. The physical centralization of the network topology information alleviates the overheads of the distributed data stores or shared file systems.

Monitor module: The desired level of consistency in the network state is dependent on the real time topology changes and other events in the network. To achieve this objective, the monitoring module is carefully defined to continuously observe the network state and find a "sweet spot" or an optimal point having sufficient level of accuracy while minimizing the corresponding overheads.

Load adaption module: Optimum resources utilization is an important priority of the data center networks and enterprise networks. By getting the real time usage updates from the monitoring module, the load adaptation module provides dynamic provisioning of the network resources. It implements the load balancing methods that make sure to efficiently utilize the capacity of the zone controllers.

Partitioning module: The logically centralized but physically distributed control plane implies segregation of workload among multiple controllers. It closely resembles the multi-tenant and multi-domain model of the modern data centers. By implementing the clustering techniques, the partitioning module creates slices of the network as per specific requirements and assign the slices to the zone controllers in an optimized way.

B. Zone controllers

Below are the major responsibilities of the zone controllers:

- Flow management of the switches that belong to their respective zones.
- Computation, selection, and installation of the routes for the relevant flows.
- Network topology discovery and maintenance with coordination of the root controller.
- Handling host and switch management issues, such as path failover, traffic engineering, quality of service, host specific updates etc.

Following are the modules that define these tasks of the zone controllers:

Path computation module: In SDN design philosophy, the complexity of the forwarding devices is significantly reduced by shifting the decision-making capability to the controller. The main objective behind this structural change is to let the control logic and forwarding logic evolve and innovate separately. Therefore, path computation and selection are performed by the controller, and the routes are installed in the switches either proactively or reactively by employing OpenFlow or any other similar protocol.

Events processing module: Events triggering is a very frequent activity in networks. The events processing module is implemented to handle the events generated by either users or other network components. It also coordinates with the root controller to notify the events in respective zones.

Application module: Users and network administrators implement network logic in the form of applications and submit to the controller for eventually deploying the sought functionality in the forwarding devices. The application module exposes sufficient set of APIs to let the intended functionality gets quickly deployed without any hassle.

Communication module: One of our design objectives is to provide the required information to various components of the network without incurring much overheads and delay. To this end, the communication module is defined to manage

the peer-to-peer interaction among zone controllers, as well as communications with the root controller at the top layer and switches at the bottom layer (via southbound channels).

Failover module: In case of any zone controller failure, the affected switches are either assigned to the other available controllers or a new controller is deployed in case of the insufficient capability of the running controllers. The recovery mechanism tries to stabilize the network without much loss, and the steady degradation strategy is applied in worst case.

Robustness: As explained earlier, the provision fo alternative connections is provided at the multiple layers of the proposed design. Zone controllers failure is handled by the logically distributed control plane. Similarly, the root controller can be replicated to a backup controller much easily due to its physical centralization. Furthermore, the timeout or heartbeat strategy can be used to decide any component failure, and the preemption can be enforced to migrate the affected devices back to their original source. Therefore, the proposed model provides sufficient opportunities to get the desired level of robustness against failures.

IV. Evaluation Platform

To validate the potential of our proposed model, we use ONOS, FlowVisor, Open vSwitch, and Mininet to build an integrated emulation platform. By default, the global network topology state is cached in memory of each instance of ONOS. And, the joining and leaving nodes are managed by the cluster membership management, that is implemented using Hazelcast's distributed structure. However, these tasks do not conform with our proposal. Therefore, we accordingly customized ONOS and FlowVisor for global network view and nodes management. Figure 2 shows the evaluation scenario of the proposed reference model. As in the diagram, ONOS instances are deployed as the zone controllers that communicate with each other to share the required information; however, the global topology management is handled by FlowVisor.

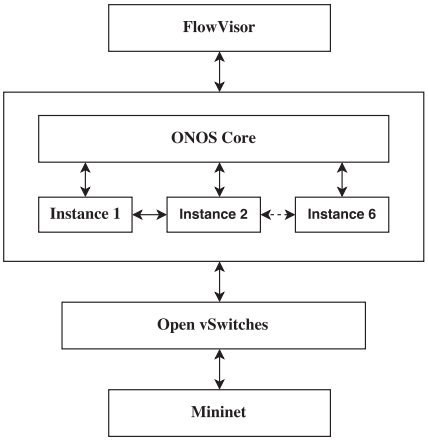


Fig. 2. The evaluation platform for the proposed model

V. Results

We used Intel (R) Xeon (R) CPU E5-2603, running at 1.60 GHz, with six cores, 64 GB RAM, 2 TB hard disk, and Ubuntu 14.04.1 LTS 64-bit operating system. Cbench and iperf tools are used for benchmarking the results. Each experiment is averaged by 20 runs. The number of hosts and switches ranges from 20 to 120 per zone for all experiments.

For the comparative analysis, we benchmarked our model with Orion. The topology configuration parameters are equally chosen for accordance of both models. In the first experiment, flow setup rates of both models are compared. Figure 3 shows the relative results of the time required by both models. As shown in the diagram, the flow setup time is lower in our model mainly due to centralized control at the root controller. With only one zone, the controller of our model can handle 17279 new flows per second, while the Orion area controller handles 8126 flows per second. Furthermore, the rate of flow setup growth is stable while increasing the number of zone controllers.

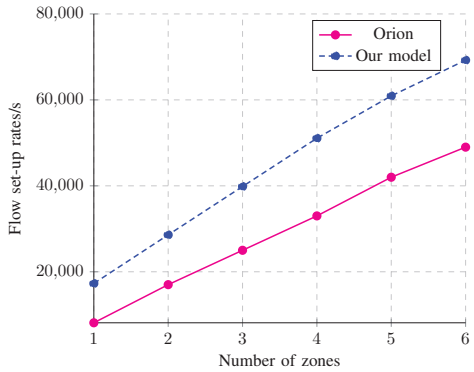


Fig. 3. Comparison of the flow setup rate

The second experiment compares the delay time incurred in Orion and our model. Again, the testing parameters are same for comparing the two models. From Figure 4 we can see that our model experiences less delay as compared to Orion. The delay time of Orion is 14 ms in 5 number of areas and 20 switches, whereas our model has 12.3 ms delay with an equal number of zones and switches. The major factor for the lower average delay time of our model is the communication among zone controllers to share the topology state and other relevant information. Improvement of few milliseconds is crucial for the time-sensitive services and applications.

To measure scalability, we performed experiments to evaluate the effect of increasing the number of zone controllers deployment. Figure 5 shows the time required to setup and tear down a topology with respect to zone controllers. We observe that the time grows relatively steadily given that the multifold addition of switches and hosts for each zone controller (120 switches, hosts per controller). It shows the large-scale capability of the proposed model.

We measured CPU utilization to assess the overheads incurred by our proposed model. As in previous experiments, the number of switches and hosts were varied to get the relative performance metric of the architecture. Figure 6 shows the graph of CPU consumption. It indicates that the rate of growth is fairly low corresponding to an increment in the number

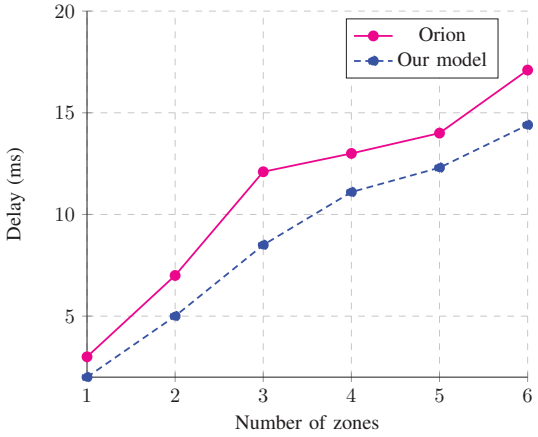


Fig. 4. Comparison of the average delay time

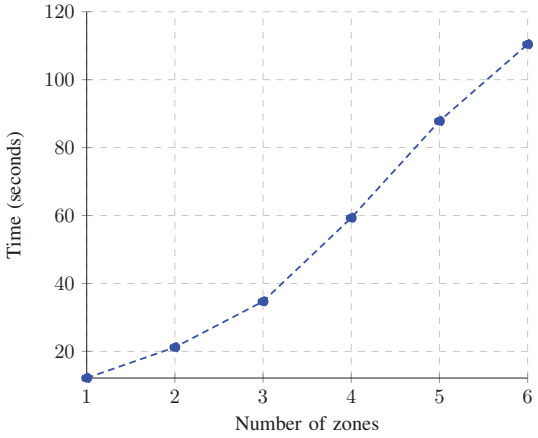


Fig. 5. Topology setup and tear down time

of zone controllers. Therefore, we can infer that the control plane carries out intended operations while minimizing the corresponding overheads.

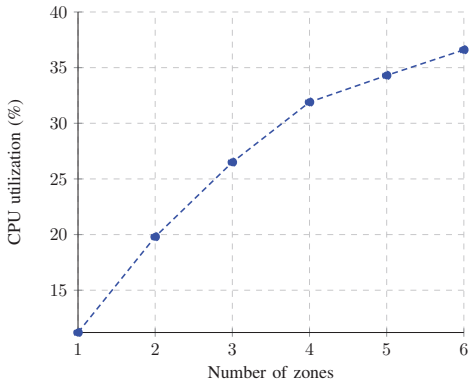


Fig. 6. CPU utilization

VI. Conclusion and Future Work

In this paper, we propose a logically centralized but physically distributed control plane architecture for SDN, that aims to be highly scalable as well performance and robustness intensive while minimizing the overheads. Our model eliminates the need of the distributed data store, distributed protocols or any similar mechanism to maintain coherent network view. And, the modular approach with the layered architecture makes it suitable to deploy in the data center and enterprise networks. Comparative results are also presented to demonstrate the potential usefulness of the model. Components migration algorithms and events registration techniques are the future scope of the work.

References

[1] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[2] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.

[3] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Computer Communications*, vol. 67, pp. 1–10, 2015.

[4] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 13–18.

[5] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.

[6] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 1–6.

[7] A. S.-W. Tam, K. Xi, and H. J. Chao, "Use of devolved controllers in data center networks," in *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*. IEEE, 2011, pp. 596–601.

[8] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[9] Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A hybrid hierarchical control plane for flow-based large-scale software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 117–131, 2015.

[10] X. Li, P. Djukic, and H. Zhang, "Zoning for hierarchical network optimization in software defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–8.

[11] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: an elastic distributed sdn controller," in *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2014, pp. 17–28.

[12] A. Koshibe, A. Baid, and I. Seskar, "Towards distributed hierarchical sdn control plane," in *Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 International*. IEEE, 2014, pp. 1–5.

[13] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, pp. 3–3.

[14] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.

[15] Y. Liu, A. Hecker, R. Guerzoni, Z. Despotovic, and S. Beker, "On optimal hierarchical sdn," in *2015 IEEE International Conference on Communications (ICC)*. IEEE, 2015, pp. 5374–5379.

Enabling Software-Defined Networking for Wireless Mesh Networks in Smart Environments

Prithviraj Patil*, Akram Hakiri†‡, Yogesh Barve * and Aniruddha Gokhale*
*Dept of EECS, Vanderbilt University, Nashville, TN, USA.
† CNRS, LAAS, 7 Avenue du colonel Roche, F-31400 Toulouse, France
‡Univ de Carthage, SYSCOM ENIT, ISSAT Mateur, Tunisia.
Email: {prithviraj.p.patil,yogesh.d.barve, a.gokhale}@vanderbilt.edu, hakiri@laas.fr

***Abstract*—Wireless Mesh Networks (WMNs) serve as a key enabling technology for various smart initiatives, such as Smart Power Grids, by virtue of providing a self-organized wireless communication superhighway that is capable of monitoring the health and performance of system assets as well as enabling efficient trouble shooting notifications. Despite this promise, the current routing protocols in WMNs are fairly limited, particularly in the context of smart initiatives. Additionally, managing and upgrading these protocols is a difficult and error-prone task since the configuration must be enforced individually at each router. Software-Defined Networking (SDN) shows promise in this regard since it enables creating a customizable and programmable network data plane. However, SDN research to date has focused predominantly on wired networks, e.g., in cloud computing, but seldom on wireless communications and specifically WMNs. This paper addresses the limitations in SDN for WMNs by allowing the refactoring of the wireless protocol stack so as to provide modular and flexible routing decisions as well as fine-grained flow control. To that end, we describe an intelligent network architecture comprising a three-stage routing approach suitable for WMNs in uses cases, such as Smart Grids, that provides an efficient and affordable coverage as well as scalable high bandwidth capacity. Experimental results evaluating our approach for various QoS metrics like latency and bandwidth utilization show that our solution is suitable for the requirements of mission-critical WMNs.**

***Index Terms*—SDN, Wireless Mesh Networks, Smart Grids, Home Area Network.**

I. Introduction

Wireless networks, and in particular, Wireless Mesh Networks (WMNs) have been actively considered as one of the most promising wireless technologies to build highly scalable wireless backhaul networks [1] for Smart Grid power systems [2], renewable energy sources [3] and solar energy harvesting base stations [4]. These WMNs assume a hierarchical structure composed of Home Area Networks (HANs) and Neighbor Area Networks (NANs), which are plugged to a Network Gateway (NG) to access the wide area networks (WANs). A HAN connects a group of sensing devices in a home to smart meters that record the energy consumption for a given home and transmit the collected data to Meter Controlling Systems (MCS). NANs connect multiple MCSs of the HANs that are geographically in close proximity and interact with cloud services in the WAN for various kinds of data collection and analytics.

978-1-5090-3216-7/16/$31.00

Despite the promise, current wireless routing protocols for WMNs are fairly limited and their extensions to power grid systems are very difficult. For example, in traditional WMNs, the nodes (i.e., mesh switches and routers) communicate with each other using routing protocols like AODV (Ad hoc On Demand Distance Vector) and OLSR (Optimized Link State Routing Protocol), which are inherently distributed because each node broadcasts information about its directly connected end devices to all other nodes. Using this information, each node will derive its own routing path to all the other nodes independently and in a distributed fashion. As a result, they reflect a partial visibility of the network without paying attention to the real network conditions. This local visibility limits the ability of WMNs to perform network engineering in large-scale wireless networks. Additionally, WMNs are difficult to manage and upgrade because their configuration must be enforced manually at each mesh router, which is both difficult and error-prone.

Software-Defined Networking (SDN) [5], [6] shows significant promise in meeting the networking needs of Smart Grid systems [7] due to the separation of the control and forwarding plane. Recent trends in Smart Grids reveal that SDN has been used to implement a multi-rate, multicast network for Phasor Measurement Unit (PMU) data [8]. Similarly, Rinaldi et al. [9] investigated a wired SDN controller to manage the network infrastructure of smart grid and provide a resilient Smart Grid systems [10]. Despite these advances, all these efforts have used SDN only in wired networks, which make their solutions unusable in smart environments due to their distributed and often wireless nature.

Addressing these needs is challenging for the following reasons. Since SDN was initially conceived for wired networks, the controller in the traditional case statically establishes paths to every switch so that it can then run centralized routing algorithms. In wireless networks, however, the controller has to discover all the switches before it can run the centralized routing algorithms. This can be achieved by installing wireless routing algorithms like AODV and OLSR in switches, but this means that the switches lose the simplicity that is envisioned by the SDN paradigm, which calls for no intelligence to reside in the switches. In other words, the SDN requirements of a centralized routing control and simple switch design contradict with the distributed routing algorithms and sophisticated

switch design of the wireless network architecture.

Consequently, we need an approach that enhances SDN for WMNs such that the new approach can centralize the control decisions – a trait of SDN – over distributed wireless mesh networks. To realize such a capability, we present a novel approach for creating an intelligent Software-Defined Wireless Mesh Network suitable for use cases, such as Smart Grids. Our approach proposes a novel way of performing routing in three stages in SDN-based WMNs by using a modified OpenFlow protocol, which allows us to remain faithful to the SDN philosophy of keeping the switch design simple and of a logically centralized control plane while also allowing flexibility and mobility that is inherent in distributed wireless mesh networks. The rest of the paper delves into the details and evaluation of our proposed approach.

II. Related Work

This section compares related work along two dimensions with our work.

A. SDN and Routing in Wireless Mesh Networks

Wm-SDN [11] attempts to address the routing challenge in SDN-based WMNs described in Section I by using a hybrid protocol. It uses the traditional AODV distributed protocol for switch-controller connection. Subsequently, when the controller-switch connection is established, it then uses SDN-based centralized protocols for switch-switch routing decisions. In this architecture each switch must support both SDN OpenFlow and also legacy routing protocols. Consequently, this approach is technically not a pure SDN-based approach since the switch is involved during the first part of routing decisions (i.e. during AODV). Moreover, it requires the switch to support complex hardware and software than what SDN envisions.

Some other efforts [12] [13] have also proposed hybrid approaches for routing though of a different kind. They propose to combine SDN-enabled switches and legacy switches (or routers) in a wireless mesh router , where SDN-enabled switches form the SDN network while traditional switches form the legacy network. In this architecture, the legacy switches run traditional routing algorithms (OSPF in this case) while every SDN enabled switch has to be in direct contact (wireless or wired) with one such legacy switch. This allows SDN-enabled switches to not support any complex hardware and software. However it requires each SDN-enabled switch to communicate with at least one legacy switch.

B. SDN-enabled Smart Grids

There are some research initiatives to leverage the potential of SDN in Smart Grid communication. Jianchao et al. [14] discussed opportunities that bring SDN to support the potential use cases in Smart Grid. Similarly, Dong et al. [10] studied the benefits of SDN to improve network resilience in Smart Grid. Rinaldi et al. [9] proposed using a wired SDN controller to simplify and automate the network management in power grids. Cahn et al. [15] presented a Software-Defined Energy Communication Network (SDECN) framework for self-configuring IEDs for substation automation. Likewise, Dorsch et al. [7] presented fast recovery and load management algorithms for the distribution and transmission in power grids. Additionally, multicast Phasor Measurement Unit (PMU) has been investigated [8]. Thomas et al. [16] proposed using a SDN-enabled multicast scheme to support flexible and fault-tolerant group communications in power systems.

Although the previous works enabled SDN in smart grids, they consider only wired communications. In contrast to wired networks which are known to be stable and robust, our contribution addresses wireless mesh smart grid networks so as to extend power systems to rural and disaster regions. Our solution provides an efficient and affordable coverage as well as low-latency, and flexible and network-aware communications.

III. Three-stage Routing Architecture and Design Considerations

This section delves into the details of our three-stage routing approach for SDN-enabled wireless mesh networks.

A. Three-Stage Routing in SDN-enabled WMNs

In this section, we describe our 3-stage routing approach for SDN-enabled WMNs. Our work is realized as an extension to the OpenFlow protocol for the wireless mesh networks, where the new three-level routing strategy enables existing OpenFlow protocol to adapt to the dynamic nature of the WMNs. Below we describe the three stages of this routing strategy.

- **Stage 1: Initial Controller-Switch Connection:** As shown in Figure 1, in the SDN-based wireless mesh networks, only a few switches are directly connected to the controller. The first task is to connect all the switches to (at least one) controller by setting up initial/basic routing. We propose an initial (i.e., non-permanent) routing stage where a SDN controller will find all the switches by flooding the network without considering whether the path it finds is best or not. To achieve this, we use an OpenFlow-based routing algorithm for initial controller-switch connection by adapting OLSR in two ways. First, instead of the switches broadcasting their link state (i.e. information about directly connected end devices), the controller will broadcast information about its directly connected switch. Second, instead of running a pure wireless mesh routing protocol like AODV or OLSR in the switches, we modify the OpenFlow client in the switch such that it finds the initial path to the controller without requiring any additional software.
- **Stage 2: Controller-Switch Path Optimization:** Once the initial connection is established, the routing paths set up in this stage will be used to install new alternatives (i.e., shortest, optimum or load balanced) paths. The controller can decide to choose a path among these alternative paths between itself and a switch since at this point, the controller has a global view of the network. Thereafter, it installs the corresponding rule to

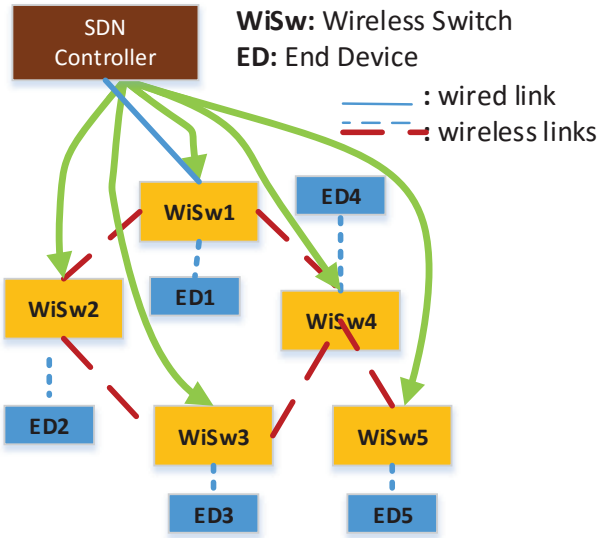


Fig. 1: Example SDN Mesh Scenario

route packets from the switches to itself in the switches by propagating the information using the original non-optimized paths.

- **Stage 3: Routing among Switches:** After the second step, the controller will derive the shortest path routing among switches themselves and it will then install these routing paths and corresponding forwarding rules using the shortest paths set-up in the previous stage. As described above, to achieve the above routing strategy, we had to modify the OpenFlow client.

B. Implementation Details

We now present the details of our implementation and the message types we created for the three stages of our approach.

1) *OF_Initial_Path_Request:* Initially, the controller will send the OF_Initial_Path_Request messages to all its directly connected switches. As described in Figure 1, the switches could be either connected via wired interface or wireless interface to the controller. Additionally, the controller could be at the same location as that of the switch or could be situated in the cloud data center.
Once a switch receives those messages, it updates the controller path destination with the source id found in the received OF_Initial_Path_Request message. Then, the switch creates a new OF_Initial_Path_Request message with its own source id and broadcasts it to the other switches. This step is performed periodically by every switch. Thereafter, every switch that receives the OF_Initial_Path_Request message establishes an initial path to the controller. This path may not provide shortest paths but only be used as a first step for obtaining the shortest path in the next stage. Also, since every switch broadcasts this message periodically, this helps to handle the mobility in the network.
2) *OF_Initial_Path_Response:* The switch sends this message to the controller on finding the initial path in stage I. This message is directed towards the controller and is only sent to that neighbor from which the switch received OF_Initial_Path_Request message first, i.e. this message is sent via the initial path between switch and controller. However, this response message contains SSIDs of all the neighboring switches. i.e. all the neighboring switches from whom this switch received OF_Initial_Path_Request message.
3) *OF_Controller_Shortest_Path:* This OpenFlow message is used to optimize the initial connection path between the controller and the switch. The Controller sends this message to the switches to update the path towards the controller with the shortest path. This message is sent only when the initial path differs from the shortest path between controller and switch. Moreover, this message is always sent using the initial path. When the switch receives this message, it installs the new path to the controller, which is shorter than the previous path. Moreover, the switch gives this new path higher preference by installing the rule for this path before the rule of the initial path. So from that point onward, whenever the switch sends any message to the controller, it takes the shortest path. Only when the shortest path fails to deliver a message, the initial path is used as a backup.

C. Demonstrating the Approach in a Use Case

We now demonstrate how our approach works for a topology shown in Figure 1. Each switch is connected to a single edge device while only the first switch is connected to the controller directly.

Figure 2 shows the interactions in all the three stages. In Stage-I, the controller sends the OF_Initial_Path_Request to Switch-1 using flooding. Switch-1 then duplicates this message and sends it to Switch-2 and Switch-4. This allows each switch to find an initial path to the controller. In Stage-II, each switch sends OF_Initial_Path_Response message to the controller via the initial path found in Stage-I. This message contains information about neighboring switches.

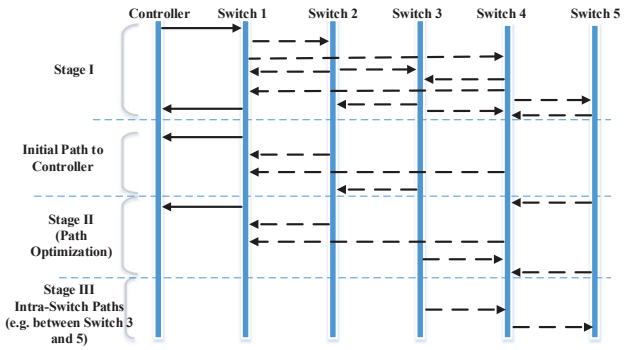


Fig. 2: Three-Stage Routing Interactions

In our example, Switch-3 has received OF_Initial_Path_Request from two switches (Switch-2 and Switch-4). However, it has received the message from

Switch-2 first. Hence, for Switch-3, the initial path to the controller is via Switch-2. However, when the Switch-3 replies to the Controller using OF_Initial_Path_Response message via initial path (i.e. via Switch-2), it includes the SSID of Switch-2 and Switch-4 in it. This allows the Controller to deduce the neighbors of each switch and hence the topology of the entire network. Using the knowledge about the topology of the network, the controller now installs the shortest path routes in switches as shown in Figure 2 where Switch-3 now has a shorter path to the Controller via Switch-4 instead of via Switch-2.

Finally in Stage-III (Figure 2), the controller installs routing paths among switches using the shortest path from Stage-II. As shown in Figure 2, the Controller installs OpenFlow rules such that Switch-3 can reach Switch-5 via Switch-4.

IV. Experimental Evaluation

We have implemented our three-stage routing strategy using the SDN emulation framework called Mininet. The basic Mininet does not provide wireless link support. An extension called Mininet-WiFi provides basic support for simulating wireless links but lacks support for essential wireless network based algorithms like shortest path or AODV or OLSR, which are normally supported by other network simulators like NS2 or NS3. Hence, we have used NS3 which is a network simulator with support for wired and wireless links. NS3 also provides support for OpenFlow client, which is required for SDN based switched. Thus, in order to bridge NS3 with Mininet, we have used OpenNet. The OpenNet simulator bridges NS3 and Mininet to provide wireless simulation in the SDN based network settings.

Once the wireless mesh network (of mobile switches and mobile hosts) is created using Mininet and NS3, the SDN controller is connected to one or more of the wireless switches. For this purpose we have used the POX controller. The three stage routing strategy is implemented on top of the POX controller as a network application. This strategy also makes use of shortest-path algorithms supported by the NS3 simulator under the hood.

We have evaluated our approach by comparing its performance with the hybrid approach in [11]. For comparison we used three metrics (1) controller-switch connection latency, (2) controller-switch reconnection latency, and (3) switch-switch connection latency.

In the beginning, the controller will try to connect to all the switches using messages OF_Initial_Path_Request and OF_Initial_Path_Response for finding the initial path in Stage-I, which in turn will be used to install the shortest path in Stage-II. We measure the latency to perform Stage-I and Stage-II, and compare it with the hybrid approach. Figure 3 plots this controller-switch connection latency against the number of hops between the controller and switch. We measure this latency for the hybrid approach, Stage-I and Stage-II. As can be seen from the figure, our approach basically breaks down the latency required by the hybrid approach into two stages (Stage I and Stage-II). The latency incurred by the hybrid approach is approximately the sum of the latency incurred by Stage-I and Stage-II.

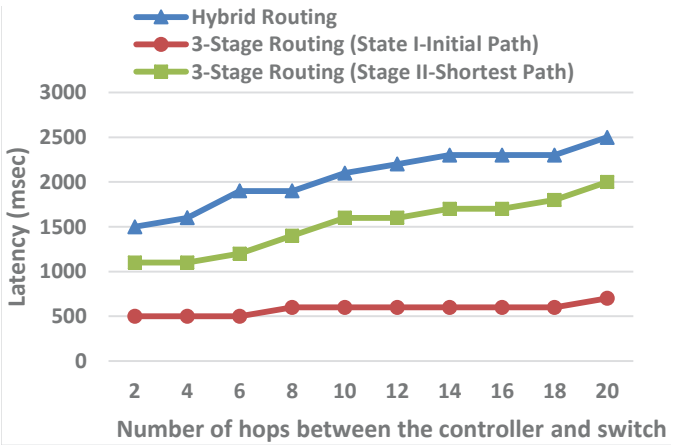


Fig. 3: Controller Switch Connection Latency

We observed the same behavior when we measured the latency for re-connecting the controller-switch link during failure. We measured this latency against the number of broken links between controller-switch as seen in Figure 4. Here also we can see the Stage-I and Stage-II reconnection latency adds up to the reconnection latency in the hybrid approach.

It is evident that our approach breaks up routing into two stages where the first stage finds a potentially inefficient route to the controller but takes lesser time while the second stage tries to optimize the route found in the first stage but takes more time. This helps overall performance of actual routing between switch-switch connection in Stage-III as seen in Figure 5. This figure shows the connection latency among switches against number of hops between them. It is seen that the stage-III of our approach outperforms the hybrid approach as the number of hops increase between switches. The reason for this result is that the switch has better connection to the controller in our approach than in the hybrid approach as shown in Figures 3 and 4.

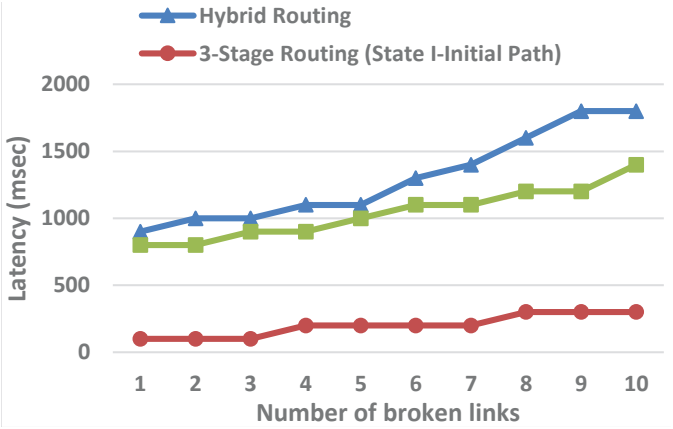


Fig. 4: Controller Switch Re-Connection Latency

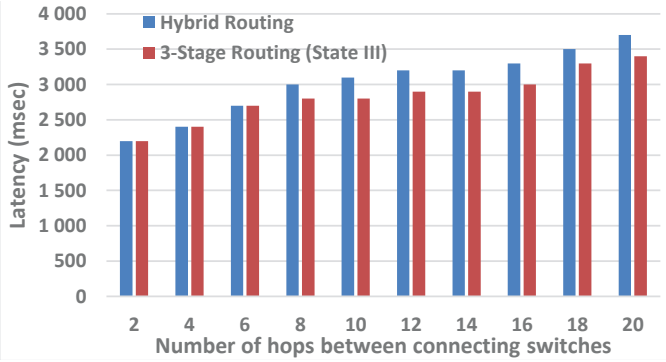


Fig. 5: Switch-Switch Connection Latency

In SDN, whenever a switch wants to connect to another switch, it requests the controller to install the routing rules. Hence, the connection to the controller plays a big role in the switch-switch connection latency. In wired networks, there is almost no mobility of nodes and hence once a controller installs rules in switches, these rules may not need to be changed frequently. Hence, routing among switches is not impacted by the controller-switch latency in wired networks. However in wireless mesh networks, as nodes can move more frequently, the switch needs to establish a reliable connection to the controller in order to improve the switch-switch routing.

This is the advantage of our approach over the hybrid approach. The hybrid approach always tries to find the best route to the controller, which in turn incurs higher latency and hence the controller-switch connection becomes unavailable for longer durations. Our three-stage approach, however, tries to find the first available route to the controller incurring smaller latency in Stage-I which helps to keep controller-switch connection alive for longer durations and thereby improving system availability as well as helping in reducing the latency in Stage-III.

V. Conclusions

In this paper, we proposed a three-stage routing strategy to efficiently use SDN in wireless mesh networks. In this regard, we proposed extensions to the existing OpenFlow protocol with three new type of messages which facilitates the three stage routing. We then evaluated the three-stage routing approach using latency metrics: one for the connections between the controller and switch, and another for connection between switches. The code for the prototype simulation can be found at [1]. In this work, we used a centralized controller in this work. However, in wireless scenarios, distributed controllers are more realistic. We plan to extend our work to centralized controllers and compare the performance.

Acknowledgments

This work was funded by the Fulbright Visiting Scholars Program, NSF CNS US Ignite 1531079 and by AFOSR DDDAS FA9550-13-1-0227. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Fulbright, NSF and AFOSR.

[1]https://github.com/prithviraj6116/sdn-wireless-mesh

References

[1] IEEE Std 1646-2004, "Ieee standard communication delivery time performance requirements for electric power substation automation," *IEEE Std 1646-2004*, pp. 1–24, 2005.

[2] Y. Xu and W. Wang, "Wireless mesh network in smart grid: Modeling and analysis for time critical communications," *Wireless Communications, IEEE Transactions on*, vol. 12, no. 7, pp. 3360–3371, July 2013.

[3] L. Cai, Y. Liu, T. Luan, X. Shen, J. Mark, and H. Poor, "Sustainability analysis and resource management for wireless mesh networks with renewable energy supplies," *Selected Areas in Communications, IEEE Journal on*, vol. 32, no. 2, pp. 345–355, February 2014.

[4] Z. Fadlullah, T. Nakajo, H. Nishiyama, Y. Owada, K. Hamaguchi, and N. Kato, "Field measurement of an implemented solar powered bs-based wireless mesh network," *Wireless Communications, IEEE*, vol. 22, no. 3, pp. 137–143, June 2015.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, "Software-defined networking: challenges and research opportunities for future internet," *Computer Networks*, vol. 75, pp. 453–471, 2014.

[7] N. Dorsch, F. Kurtz, H. Georg, C. Hagerling, and C. Wietfeld, "Software-defined networking for smart grid communications: Applications, challenges and advantages," in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, Nov 2014, pp. 422–427.

[8] A. Goodney, S. Kumar, A. Ravi, and Y. Cho, "Efficient pmu networking with software defined networks," in *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, Oct 2013, pp. 378–383.

[9] S. Rinaldi, P. Ferrari, D. Brandao, and S. Sulis, "Software defined networking applied to the heterogeneous infrastructure of smart grid," in *Factory Communication Systems (WFCS), 2015 IEEE World Conference on*, May 2015, pp. 1–4.

[10] X. Dong, H. Lin, R. Tan, R. K. Iyer, and Z. Kalbarczyk, "Software-defined networking for smart grid resilience: Opportunities and challenges," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, ser. CPSS '15, 2015.

[11] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, "Wireless mesh software defined networks (wmsdn)," in *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 2013, pp. 89–95.

[12] Y. Peng, L. Guo, Q. Deng, Z. Ning, and L. Zhang, "A novel hybrid routing forwarding algorithm in sdn enabled wireless mesh networks," in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*. IEEE, 2015, pp. 1806–1811.

[13] A. Abujoda, D. Dietrich, P. Papadimitriou, and A. Sathiaseelan, "Software-defined wireless mesh networks for internet access sharing," *Computer Networks*, vol. 93, pp. 359–372, 2015.

[14] J. Zhang, B.-C. Seet, T.-T. Lie, and C. H. Foh, "Opportunities for software-defined networking in smart grid," in *Information, Communications and Signal Processing (ICICS) 2013 9th International Conference on*, Dec 2013, pp. 1–5.

[15] A. Cahn, J. Hoyos, M. Hulse, and E. Keller, "Software-defined energy communication networks: From substation automation to future smart grids," in *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, Oct 2013, pp. 558–563.

[16] T. Pfeiffenberger, J. L. Du, P. Bittencourt Arruda, and A. Anzaloni, "Reliable and flexible communications for power systems: Fault-tolerant multicast with sdn/openflow," in *New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on*, July 2015, pp. 1–6.

SDAR: Software Defined Intra-Domain Routing in Named Data Networks

Yaoqing Liu
Clarkson Univesity

Hitesh Wadekar
Clarkson Univesity

***Abstract*—Named Data Networking (NDN) is a newly proposed content-centric network architecture that naturally supports efficient content distribution by routing data names instead of conventional IP prefixes. Taking advantage of the unique feature-adaptive forwarding in NDN and the centralized management and control in Software Defined Networking (SDN), we design and develop SDAR, a Software Defined Intra-Domain Routing Control Platform in NDN, that can manage network-wide routers and various dynamics effectively. To the best of our knowledge, it is the first time that we combine SDN and NDN ideas to implement adaptive forwarding via intra-domain multi-path routing algorithms. More specifically, we made the following contributions: (1) Designed an efficient communication model between routers and the controller within a single administrative domain; (2) Prototyped a centralized platform for handling different types of network dynamics, such as link failures and cost changes; (3) Ported multiple existing single-path and multi-path routing algorithms to the platform for robust and adaptive intra-domain routing; (4) Evaluated the effectiveness of SDAR through NS3/ndnSIM simulation with realistic settings.**

I. Introduction

In NDN, routers forward Interest packets and maintain state information for the corresponding Data packets to take the reverse path. The feature that NDN routers keep stateful forwarding information through symmetric Interest-Data exchange enables that the forwarding plane can detect and recover from network failures or changes through exploring multiple alternative paths without assistance from the control plane. As a result, this adaptive forwarding capability relaxes the requirement of routing protocols that were originally designed to react to short-term dynamics of peer participation, and thus significantly improves network scalability and stability. Meanwhile, the unique feature opens doors for other routing protocols that were previously considered infeasible or ineffective for IP networks [1], such as centralized intra-domain routing protocols. Software Defined Network (SDN) architecture [2] decouples the network control and forwarding functions enabling flexible, manageable, cost-effective and adaptable network control over underlying infrastructure and network services. It features logically centralized management maintaining a global view of the network topology and optimizing dynamic network resources. SDN and related standards were created to solve the limitations of current networks: Complexity that leads to stasis, inconsistent policies, inability to scale and vendor dependence. Note that the SDN here is a broder definition rather than the traditional OpenFlow oriented SDN. Actually, one of our primary contributions is to use an non-OpenFlow approach to implement centralized routing managment.

978-1-5090-3216-7/16/31.00

We encountered three major challenges when implementing SDAR as follows: (1) Communication Model Design: SDAR framework was motivated by SDN in IP networks, but SDAR communication model, which will be within NDN context, cannot be directly transformed from traditional IP push-based communication model. (2) Routing Platform Integration: The second challenge is to design and develop a generic and centralized routing platform from the ground up. The platform will be able to handle both incoming and outgoing routing traffic, to store network-wide routing information in various efficient data structures, to provide uniform interfaces for different routing algorithms, and to quickly compute single and multiple paths upon topology and link changes. Although existing work has tried to integrate NDN to IP-based OpenFlow framework, these centralized functionalities have not been implemented from pure data-centric perspective in NDN research communities, e.g., ndnSIM package ([3]). (3) Link Detection and Network Dynamics Handling: There is neither a centralized NDN-oriented link layer discovery protocol, like Link Layer Discovery Protocol (LLDP) in SDN & IP world [4], to detect whether a link to its neighbor is alive or not, nor a protococl that can deal with topology changes and link cost changes dynamically and intelligently in a centralized manner. Therefore, the third challenge is how to obtain the status of links between neighbors and how to handle various network dynamics from scratch. As such, we implemented a link detection protocol that can identify link or node failures regularly.

The rest of this paper structured as follows: Section II presents a high-level overview of SDAR routing platform; Section III articulates detailed design and implementation; Section IV shows evalution results; Section V states related work; and Section VI concludes the paper with future work.

II. Overview

SDAR system consists of two types of components: controller and node. A node could be either an NDN router or an end system. An NDN router is responsible for forwarding packets to destinations and registering its directly connected links and name prefixes to the controller. The controller's role is three-fold: (1) Collecting link information, such as costs, adjacencies, status changes and name prefixes, as well as generating network-wide topology; (2) Calculating single-path or multi-path routes for individual nodes based on the generated network topology; (3) Dispatching the computed prefix-level routes to individual routers. These components run on top of NDN protocol stack and use NDN packets to communicate with each other. In our SDAR platform, we manually configured static routes for nodes to reach each other during the bootstrap stage. Note that the configured

Naming Format	Usage
/controller/<node>/link	Node requests controller to pull link information
/<node>/controller/link	Controller pulls link information from node
/<node>/controller/route	Controller informs node to retrieve calculated routes
/controller/<node>/route	Node pulls calculated routes from controller
/controller/<node>/update	Node requests controller to pull link update information
/<node>/controller/update	Controller pulls link updates from node
/<node>/controller/upt_route	Controller informs node to retrieve updated routes
/controller/<node>/upt_route	Node pulls updated routes from controller

Fig. 1: SDAR Naming Format and Usage

routes are merely used to achieve reachability, rather than for best performance or shortest paths, which will be done by the controller. The reachability routes can also be installed dynamically through broadcasting discovery Interests, where the first ingress face that receives the broadcasting Interest on a node can be the next-hop face toward the destination node.

III. Design

A. Naming Scheme

In NDN, content is identified by hierarchical structured names with varying numbers of components. Based on current network structures and operations, a hierarchical naming scheme will well depict relationship among various components in the system. Figure 1 describes eight types of Interests that will be used to fulfill the needs for intra-domain routing. SDAR is designed mainly for intra-domain routing, each node can be uniquely identifed by its name and thus for simplicity the node name is used for its NDN name. For example, the controller can be named as $/controller$, and a node with name $consumer$ has a NDN name of $/consumer$. If the name is used in inter-domain routing eventually, a network and site name can be appended in front of the node name.

B. Communication Model

SDAR is motivated by SDN and designed to decouple routing control from forwarding and move routing decision processes to a centralized controller. Under SDAR architecture, routing traffic will be flowing from nodes to the SDAR controller and vice versa. The SDAR controller's tasks include creating routing topology and calculating routes. One question is that which end is in a better position to initiate the request to pull link adjancency or change information. There are a few tradeoffs: if the controller initiates the request, then it only needs to establish a two-way communication (one Interest to a node and one Data back) to obtain this information. But the problem is that the controller does not know when such information will be ready to be retrieved, thus it needs to periodically send Interests to each individual node. Tracking all of the nodes in the network by the controller leads to high computational and traffic overhead as the controller has to send tracking Interests to individual nodes continuously irrespective of topology changes. As such, we adopt another demand-driven approach, where each node initiates the communication to

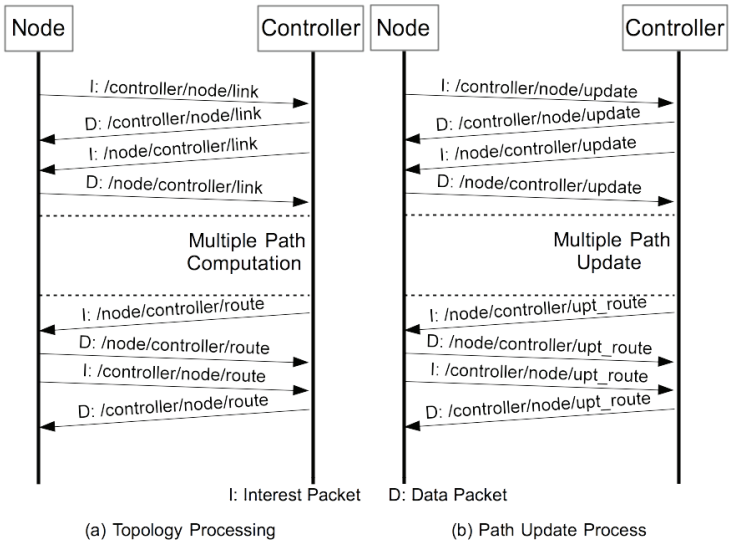


Fig. 2: Communication Model

notify the controller to pull data by sending a request Interest. Once the controller receives the notification, it acknowledges the request with an empty Data packet to satisfy the Interest and meanwhile issues its own Interest to pull the data from the node. This four-way communication model, although may incur additional traffic overhead, will be more efficient if the network does not suffer from highly frequent link changes and topology updates. Figure 2 illustrates the communication model for both topology and update processing.

C. Routing Platform Integration

The most challenging part for SDAR framework turns out to be the implementation and integration of different system components from the ground up. Current ndnSIM package only provides a lower-layer support of NDN protocol stack, including FIB, PIT, CS and forwarding strategies. However, it does not offer any dynamic routing protocol for our reference, nor other necessary functional support for uppper-level routing-related applications. Therefore, we strive to design and develop these functionalities from scratch. These components include Routing Traffic Management, Topology Management, Path Computation, Event Management, and Configuration Management. Path computation is strongly connected by other components as shown in Figure 3. Routing Traffic Management module is mainly responsible for parsing incoming Interest and Data packets for further processing as well as pushing application packets out to the network. The main responsibility of Topology Management module is to maintain a global topology of all routers and their links in the network for different routing algorithms to use. In Path Management Module, we have successfully ported single-path Dijkstra's algorithm, multi-path Dijkstra's algorithm, and Yen's K-shortest path algorithm to SDAR routing platform. The configuration management is in charge of configuring different features for the SDAR controller, such as configuring NS3/ndnSIM libraries, selecting different algorithms for routing, logging, and data structures initialization. Network Dynamics Handling module, as shown in Figure 4, is mainly responsible for handling various network dynamics, including link cost changes and link failures.

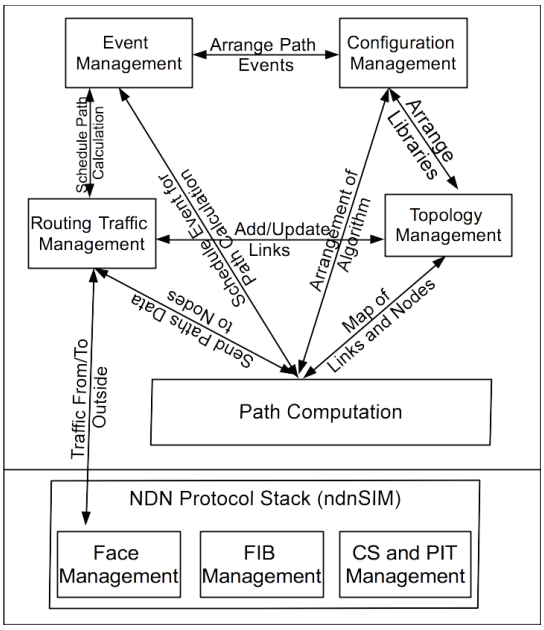


Fig. 3: SDAR Routing Platform

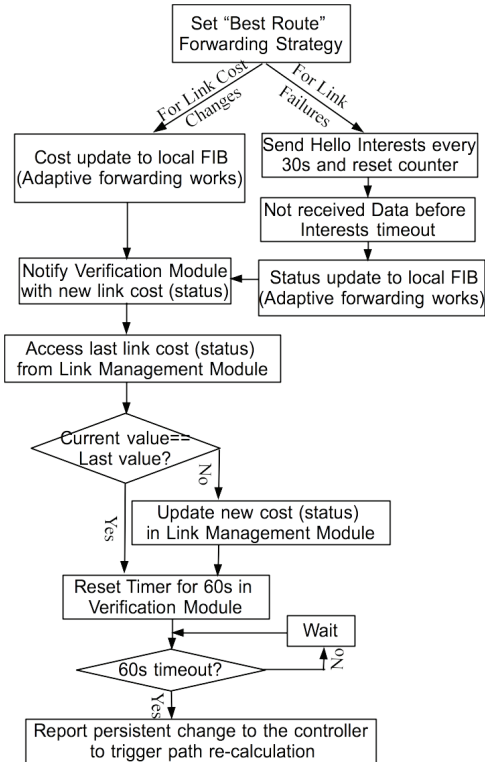


Fig. 4: Network Dynamics Handling

IV. Evaluation

This section we introduce the experiment environment and evaluate the effectiveness of multi-path routing on SDAR platform. We also make a comparison between single-path and multi-path routing algorithms on a centralized controller in terms of efficiency and traffic overhead.

A. Methodology

We conducted our simulation and evaluation on an Intel(R) Core(TM)2 Duo CPU E7500 @2.93GHz, 4GB memory server.

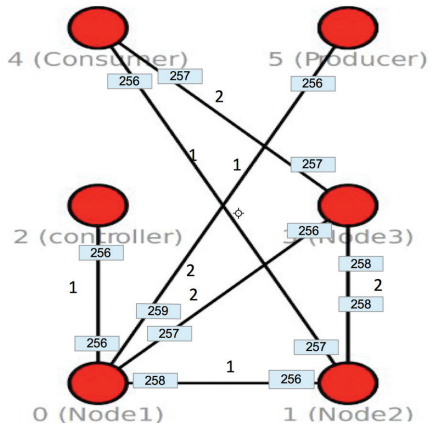


Fig. 5: Network Topology

We used ndnSIM simulator [5] to design, implement our SDAR routing platform. We use the 'topologyReader' module in ndnSIM to generate a six-node topology, as shown in Figure 5, for testing and evalution. Each link in the topology has a bandwidth 1Mbps, delay 10ms and the number of maximum packets in the transmission queue is 10. Face id (rectangle box) and link cost between two nodes are also shown in the topology. We configured ndnSIM as follows: we implemented and installed our SDAR controller application at ndnSIM application layer on $controller$. We installed customized version of consumer and producer applications on $Consumer$ and $Producer$ nodes. Based on our design, each node in the topology initially uses our developed applications to communicate with the controller to announce its adjacency information, and afterwards obtains multiple routes to reach other nodes in the network. We set three paths for each destination while calculating paths using multi path algorithm. We configured forwarding strategy as 'BestRoute' on each node. Namely, NDN forwarding engine is able to dynamically select the best route, e.g., lowest-cost route, among all available routes to the same destination and switch the traffic to the new route. The 'Consumer' node generates Interest towards 'Producer' node with frequency of 10 Interests per second. For all experiments we used the same topology and traffic rate.

B. Effectiveness of SDAR Routing Multi-Path

In order to demonstrate the effectiveness of multi-path routing in SDAR architecture, we design a particular scenario: At the beginning of the simulation, each node sends link information to the controller, which calculates three shortest paths between any pair of nodes and installs the corresponding routes to the forward information base (FIB) of every node. Subsequently, we schedule to increase the cost to a very large value on the link between $Consumer$ and $Node2$. Thereafter, we observe what path the traffic flows will take toward its destination. At the beginning of the experiment, We observe that three next hops are available in the FIB on the consumer node with a preferred face number based on the $BestRoute$ stragety. Then we changed the link cost between the consumer and an internal node and we found the change triggerred the controller to calculate a new path that was used afterwards. Therefore, we demonstrate the effectiveness of SDAR multi-path routing by leveraging the feature of NDN adaptive forwarding. Due to space constraints, we skip the figures to show the changes of paths and FIB entries.

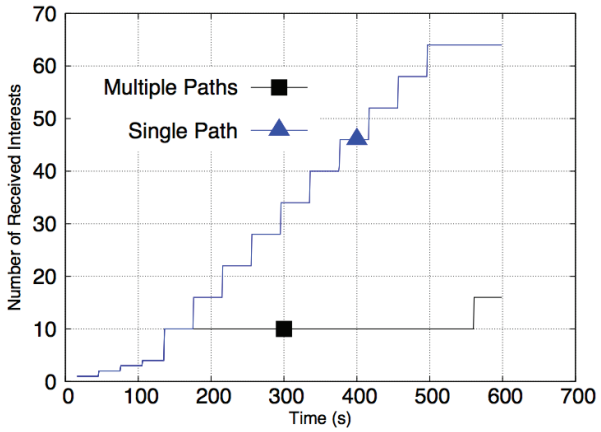


Fig. 6: Traffic Overhead Comparison

C. Single-Path and Multi-Path Comparison

If there is only one single path from a source to a destination, controller will be interrupted for every occurance of link changes on the path. However, if multiple paths are available, then an alternative path will be selected for use temporarily by NDN forwarding engine and only persistent changes need to interrupt the controller for path re-computation. We designed two experiments to show the strengths of mutli-path routing compared with single-path routing under SDAR in terms of traffic overhead. Both experiments schedule exactly the same events of link changes, but the controller uses a single-path (Dijkstra's) algorithm in the first experiment and a multi-path (Yen's K-shortest) algorithm in the other one. Figure 6 demonstrates the accumulative number of Interest packets that interrupted the controller obtained from the two simulations. From the figure, we can notice that bootstrap process finished at $140s$ when all routes are collected, calculated and installed to individual nodes by the controller. From then on, a link is scheduled up and down every $40s$ and the change stays since $500s$. The line with a Triangle represents the results of single-path scenario and the other line denotes the results of multi-path scenario. We can observe that every $40s$, the controller receives a cluster of Interests that were issued by the affected node of the link change, because of no alternative paths available; however, the controller was not interrupted until $560s$, when the change has been lasting more than the assigned threshold ($60s$) since last link change at $500s$. During the entire simulation, 64 Interests interrupted the controller under the single-path settings and only 16 Interests in the multi-path settings. As such, we verified that multi-path routing in SDAR can greatly reduce traffic overhead to both the network and the controller. Most importantly, it will enhance response speed for traffic redirecting upon link changes and failures.

V. Related Work

Torres *et al.* proposed a Controller-based Routing Strategy (CRoS) [6] for NDN and has centralized architecture similar to our SDAR design but it differs from SDAR in a couple of ways. First, they did mention about routing algorithms, but we provided detailed implementation of two multi-path algorithms and their effectiveness on ndnSIM. Second, they did not take advantage of NDN features for handling network failures intelligently, like we did with the help of adaptive and multi-path forwarding. Hoque *et al.* proposed a Named-data Link State Routing protocol (NLSR) [7], for NDN. Although they offered efficient update dissemination, built-in update authentication, and native support of multi path forwarding, but every routers in the networks are involved for path calculation. Soliman *et al.* discussed about link failure in source routed forwarding with Software Defined Control, but those are IP based OpenFlow protocol solutions [8]. We took their work as reference to implement it in SDAR. Other work such as [9], proposed adaptive forwarding in NDN and opens a door for centralized routing. As compared to their work, SDAR has been proved to be a good application of adaptive forwarding. Tortelli *et al.* discussed about a bloom-filter based intra-domain routing algorithm for NDN (COBRA) [10]. They handle link failures and detection using a bloom-filter, but in our case, we use adaptive forwarding, multi-path routing and a link detection protocol.

VI. Conclusion

Through SDAR, we demonstrate how SDN concepts could feasibly and efficiently be applied to the NDN context to manage and control the networks. To the best of our knowledge, it is the first time that we combine SDN and NDN ideas to implement adaptive forwarding via intra-domain multi-path routing algorithms. Designing and implementing a centralized routing system on a centralized controller in NDN can greatly reduce the complexity and overhead in network management and routing configurations rather than on all participating routers. We hope the SDAR routing platform will help researchers better understand the role of routing protocols in NDN and also gain insights to improve current IP routing architecture.

References

[1] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "On the role of routing in named data networking," 2013. [Online]. Available: http://named-data.net/techreports.html

[2] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: decoupling architecture from infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 43–48.

[3] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2.0: A new version of the NDN simulator for NS-3," NDN, Technical Report NDN-0028, January 2015.

[4] D. in Software-Defined Networks, "http://vlkan.com/blog/post/2013/08/06/sdn-discovery/."

[5] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," NDN, Technical Report NDN-0005, October 2012. [Online]. Available: http://named-data.net/techreports.html

[6] J. V. Torres and O. C. M. Duarte, "Cros-ndn: Controller-based routing strategy for named data networking."

[7] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 15–20.

[8] P. A.-S. Mourad Soliman, Biswajit Nandy, "Source routed forwarding with software defined control, considerations and implications," in *Proceedings of CoNEXT 2012 and Co-located Workshops*. ACM, 2012, pp. 15–20.

[9] A. Yi, B. Lan Wang, and L. Zhang, "Adaptive forwarding in named data networking," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 133–145.

[10] G. B. M. Tortelli, L. A. Grieco and K. Pentikousis, "Cobra: Lean intra-domain routing in ndn," in *Proceedings of IEEE CCNC*. IEEE, 2014, pp. 839–844.

Cost-effective Processing for Delay-sensitive Applications in Cloud of Things Systems

Yucen Nan[1], Wei Li[1], Wei Bao[1], Flavia C. Delicato[2], Paulo F. Pires[2], Albert Y. Zomaya[1]
[1]The Centre for Distributed and High Performance Computing, School of Information Technologies, The University of Sydney, Australia
[2]Department of Computer Science, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
ynan2995@uni.sydney.edu.au, liwei@it.usyd.edu.au, {wei.bao, albert.zomaya}@sydney.edu.au,
fdelicato@gmail.com, paulo.f.pires@gmail.com

***Abstract*—The steep rise of Internet of Things (IoT) applications along with the limitations of Cloud Computing to address all IoT requirements promotes a new distributed computing paradigm called Fog Computing, which aims to process data at the edge of the network. With the help of Fog Computing, the transmission latency and monetary spending caused by Cloud Computing can be effectively reduced. However, executing all applications in fog nodes will increase the average response time since the processing capabilities of fog is not as powerful as cloud. A tradeoff issue needs to be addressed within such systems in terms of average response time and average cost. In this paper, we develop an online algorithm, unit-slot optimization, based on the technique of Lyapunov optimization. It is a quantified near optimal solution and can online adjust the tradeoff between average response time and average cost. We evaluate the performance of our proposed algorithm by a number of experiments. The experimental results not only match up the theoretical analyses properly, but also demonstrate that our proposed algorithm can provide cost-effective processing while guaranteeing average response time.**

I. Introduction

Nowadays, an increasing number of objects and physical devices are connecting to Internet, significantly impacting the global volume of generated traffic and ushering in a world of interconnected smart devices, thus giving rise to the Internet of Things (IoT) paradigm [1]. The ever-increasing number of IoT devices will inevitably produce a huge amount of data, which has to be processed, stored and properly accessed by end users and/or client applications. The IoT devices often have limited computing and energy resources, and are not able to perform sophisticated processing and storing large amounts of data. In this context, Cloud Computing seems to be the best candidate to meet the requirements of IoT scenarios. The theoretically unlimited resources of clouds allow efficient processing and storage of massive amounts of data generated by the IoT devices. The combination of IoT and Cloud Computing brought about a new paradigm of pervasive and ubiquitous computing, called Cloud of Things (CoT) [2], [3]. CoT denotes the synergetic integration of Cloud Computing and IoT, in order to provide sophisticated services in a ubiquitous way, for a multitude of application domains. A CoT, as a two-tier IoT system, is composed of virtual things built on top of the networked physical objects and provides on-demand provision of sensing, computational and actuation services, allowing users to pull the data from the physical things by using a pay per use, flexible and service-oriented approach.

978-1-5090-3216-7/16 $31.00 ©2016 IEEE

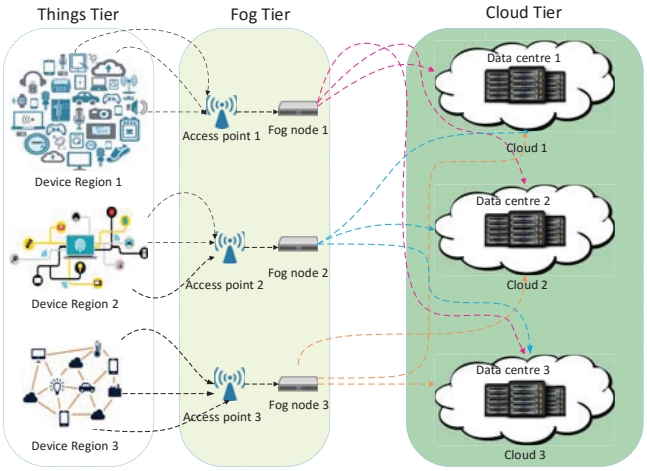


Fig. 1: The system architecture of the three-tier CoT

However, despite all the benefits provided by the integration of cloud and IoT, recent works [4], [5] raised several concerns of the two-tier CoT model. In particular, the unpredictable communication delay to the cloud data centres can become a high risk factor for the time sensitive IoT applications. Besides, the network bandwidth may be over-utilized to transfer raw data to the cloud, while most of such data could be locally processed and discarded right away. For addressing such issues, a novel and promising computing paradigm called Edge Computing [6] has recently been advocated. Fog Computing [7] as a representative paradigm of Edge Computing still retains the core advantages of using clouds as a support infrastructure, but it keeps the data and computation close to end users thus reducing the communication delay over the Internet and minimizing the bandwidth burden by wisely deciding about which data needs to be sent to the cloud. By including the fog devices in the CoT scenario, it becomes a three-tier system. As shown in Fig. 1, the Things tier (leftmost tier) includes the various physical devices such as wireless sensor and actuators networks (WSAN), RFID tags, mobile devices and several other real-world objects instrumented by sensors and/or actuators. The Cloud tier (rightmost tier) comprises the data centres. The Fog tier (intermediate tier) contains fog nodes, which are connected to the Internet and interact with the cloud and the IoT devices. In many cases, the fog nodes can perform the processing of data collected by the sensors, without resorting to the cloud. On the other hand, many applications require complex processing and/or rely on historical data stored for

a long period of time, which typically will not be available in fog nodes. Therefore, there is an important decision-making process to be managed, which refers to when to send data from the fog tier to the cloud. Several factors must be taken into account in this decision-making process, such as the availability of fog nodes, the average response time of the applications and the monetary cost, since processing the data at the fog nodes generally has zero cost while outsourcing the processing to the cloud requires payment for the services.

In this paper, we present a comprehensive analytic framework for the three-tier CoT system to prevent the fog devices to be overloaded so that the users within that area would not experience a poor quality of service. In general, such problem can be converted into a constrained stochastic optimization problem. Using the technique of Lyapunov optimization [8] over renewals, we designed an adaptive decision-making algorithm called unit-slot optimization for distributing the incoming data to the corresponding tiers without a priori knowledge of the status of users and system. The main goal of this algorithm is to ensure the data is processed within a certain amount of time, meanwhile the availability of the fog servers is still guaranteed, and the cost spending on renting the cloud services is minimized. In addition, our proposed algorithm achieves the average response time arbitrarily close to the theoretical optimum. The performed discrete-event simulation demonstrates that, under the three-tier CoT scenario, our proposed mechanism outperforms the selected benchmarks and provides the best overall performance for the users.

The rest of the paper is organized as follows: Section II introduces the system analysis. Section III presents the theoretical solution and the proposed algorithm. Section IV evaluates the performance of the proposed algorithm. Related work is provided in Section V followed by a conclusion in Section VI.

II. System Analysis

A. The Overview of the Three-tier Cloud of Things

As shown in the Fig. 1, our theoretical system model involves three tiers, namely Things tier, Fog tier and Cloud tier. In the Things tier, IoT devices send out the applications to the system backend for further processing and these applications could be processed in either the Fog or Cloud tier. If the Fog tier is selected to process applications, the applications will be directly sent to the fog node. Since fog nodes have limited resources, the application arrival rate may be higher than the node capacity to serve the application requests (we call this the service rate). Therefore, there will be a growth of queueing length in the fog node. Otherwise, the applications will be processed at the Cloud tier. Unlike sending the applications straight to the Fog tier, the applications will then be sent to the Cloud tier via the corresponding communication channel. The service rates of communication channels and cloud nodes are both higher than those of the fog nodes, but their usage incur in a certain monetary cost for renting the required resources from external stakeholders. To provide a cost-effective service with a shorter response time in a better way, we implement a decision maker located at the front of the Fog tier to make online decisions regarding if the applications will be sent to the Fog tier or to the Cloud tier. Fig. 2 shows the application processing flow in our system. Table I provides definitions of the important parameters used in our system modeling. For the sake of the simplicity of analysis, we use only one fog node to demonstrate how our proposed algorithm can optimize the system performance, but the approach applies also in the case of multiple fog nodes. Please note that, the cooperation between fog nodes is out of the scope of this paper.

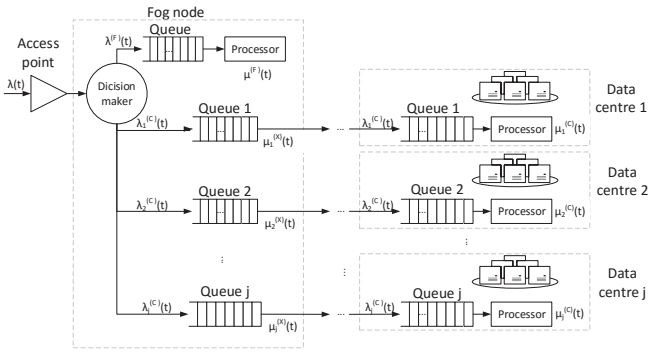


Fig. 2: Application processing flow

B. Fog Node

During the tth time slot, the arrival rate of applications in the fog node is $\lambda(t)$ and the service rate is $\mu^{(F)}(t)$. Since the fog node has limited computation capacity, $\lambda(t)$ may be greater than $\mu^{(F)}(t)$, we allow a portion of the applications to be offloaded to the Cloud tier. Let $\eta^{(F)}(t)$ denote the portion of the applications processed locally, and $\eta_j^{(C)}(t)$ denote the portion offloaded to the jth cloud node. Let $\lambda^{(F)}(t) = \eta^{(F)}(t)\lambda(t)$ denote the equivalent local arrival rate of the applications at the processor of the fog node, and $\lambda_j^{(C)}(t) = \eta_j^{(C)}(t)\lambda(t)$ denote the equivalent arrival rate of the communication channel to the jth cloud. We have

$$\lambda(t) = \lambda^{(F)}(t) + \sum_{j=1}^{J} \lambda_j^{(C)}(t) x_j(t). \tag{1}$$

In this work, we assume that the fog node can only offload its applications to one of the clouds in each time slot, so that at most one of $\lambda_j^{(C)}(t), \forall j \in \mathcal{J}$ can be positive during the tth time slot. Let $x_j(t)$ indicates if cloud j is in use. There is one and only one $x_{j^*}(t) = \mathbf{1}\left(\lambda_{j^*}^{(C)}(t) > 0\right)$ and all other $x_j(t)$ equal zero.

We model the processor at the fog node as an $M/M/1$ queue [9], and the average local response time is

$$\tau^{(F)}(t) = \frac{1}{\mu^{(F)}(t) - \lambda^{(F)}(t)}, \tag{2}$$

note that we must satisfy

$$\mu^{(F)}(t) > \lambda^{(F)}(t). \tag{3}$$

C. Communication Channel from Fog to Cloud

During the tth time slot, the arrival rate to the jth communication channel equals the arrival rate to the jth cloud node, which is denoted as $\lambda_j^{(C)}(t)$. The service rate in communication channel from fog node to cloud node is $\mu_j^{(X)}(t)$. We model the

TABLE I: Key Parameters

Notation	Definition
t	The tth time slot for a long-term period (for short, we use t in the rest of the paper).
$\lambda(t)$	Arrival rate of applications at the fog node in consideration at t.
$\lambda^{(F)}(t)$	Arrival rate of applications processing in the processor of the fog node at t.
$\lambda_j^{(C)}(t)$	Arrival rate of applications in jth cloud node at t.
$\mu^{(F)}(t)$	Service rate of the fog node at t.
$\mu_j^{(X)}(t)$	Service rate of the communication channel to the jth cloud node at t.
$\mu_j^{(C)}(t)$	Service rate of the jth cloud node at t.
$\tau^{(F)}(t)$	Average response time of processing applications in the fog node at t.
$\tau_j^{(X)}(t)$	Average response time of transmitting applications to jth cloud via communication channel at t.
$\tau_j^{(C)}(t)$	Average response time of processing applications in jth cloud node at t.
$\tau_j(t)$	A propagation delay for sending applications to the jth cloud node via communication channel at t.
$\overline{\tau}_j(t)$	Overall average response time of processing applications at t.
$x_j(t)$	Indicator of using jth channel and cloud node at t.
$P_j^{(X)}(t)$	The cost of using the communication channel to the jth cloud node at t.
$P_j^{(C)}(t)$	The cost of using the jth cloud node at t.
D	Required mean response time

communication channel as an $M/M/1$ queue if the jth channel is used, i.e., $x_j(t) = 1$, and the average transmission time in the channel is

$$\tau_j^{(X)}(t) = \frac{1}{\mu_j^{(X)}(t) - \lambda_j^{(C)}(t)}. \tag{4}$$

Furthermore, if the jth communication channel is used, there will be a propagation delay $\tau_j(t)$. In summary, if the jth cloud is used, the average response time for sending applications to the jth cloud node is

$$\tau_j^{(X)}(t) = \frac{1}{\mu_j^{(X)}(t) - \lambda_j^{(C)}(t)} + \tau_j(t), \tag{5}$$

we must have

$$\mu_j^{(X)}(t) > \lambda_j^{(C)}(t). \tag{6}$$

Using the communication channel from the fog node to the jth cloud node introduces a cost $P_j^{(X)}(t)$.

D. Processing at Cloud

The arrival rate at the cloud node is the same as that in the communication channel, so that the arrival rate to the cloud node is $\lambda_j^{(C)}(t)$. The service rate is $\mu^{(C)}(t)$. Still we model the jth cloud as $M/M/1$ queue.

Therefore, if the jth cloud node is used, the average processing time at the cloud node is

$$\tau_j^{(C)}(t) = \frac{1}{\mu_j^{(C)}(t) - \lambda_j^{(C)}(t)}, \tag{7}$$

note that we must have

$$\mu_j^{(C)}(t) > \lambda_j^{(C)}(t). \tag{8}$$

We also assume that using the cloud node introduces a cost $P_j^{(C)}(t)$.

E. Data Offloading for Cost-Effective Processing

The decision maker which involved in the system needs to decide which application to be offloaded to the cloud, and which cloud node should be used, in order to balance the response time and cost. As we introduced previously, we presume that there is no cost if the applications are processed in the fog node since the fog node are generally owned by the service providers. However, as discussed in Section II-C and II-D, if the applications are processed in the Cloud tier, it will introduce communication cost and processing cost. The overall cost at t is

$$P(t) = \sum_{j=1}^{J} x_j(t) \left(P_j^{(X)}(t) + P_j^{(C)}(t) \right). \tag{9}$$

On the other hand, as discussed in Section II-B, II-C and II-D, the average response time at t is

$$\overline{\tau}(t) = \frac{\lambda^{(F)}(t)}{\lambda(t)} \tau^{(F)}(t) + \sum_{j=1}^{J} \frac{x_j(t)\lambda_j^{(C)}(t)}{\lambda(t)} \left[\tau_j^{(X)}(t) + \tau_j^{(C)}(t) \right]. \tag{10}$$

Our aim is to minimize the long-term average cost

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left[P(t)\right] \tag{11}$$

and satisfy the following requirement about response time

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left[\overline{\tau}(t)\right] \leq D. \tag{12}$$

III. Problem Solution based on Lyapunov optimization

In order to satisfy the response time constraint (12), we need to introduce a virtual queue $Q(t)$, which is used to accumulate the part of the response time that exceeds the expected finish time, and we define $Q(0) = 0$. $Q(t)$ evolves as follows

$$Q(t+1) = \max[Q(t) - D, 0] + \overline{\tau}(t) \tag{13}$$

Lemma 1. If $Q(t)$ is mean rate stable, then (12) is satisfied

Proof. Due to the definition of $Q(t)$, we have

$$Q(t+1) \geq Q(t) - D + \overline{\tau}(t). \tag{14}$$

Then taking expectation of the above inequality, we have

$$\mathbb{E}\left[Q(t+1)\right] - \mathbb{E}\left[Q(t)\right] \geq -D + \mathbb{E}\left[\overline{\tau}(t)\right].$$

Summing up both sides of the above inequality over $t \in [0, T-1]$ for some positive integer T yields and using the law of iterated, we have

$$\mathbb{E}\left[Q(T)\right] - \mathbb{E}\left[Q(0)\right] \geq -TD + \sum_{t=0}^{T-1} \mathbb{E}\left[\overline{\tau}(t)\right].$$

Then through dividing by T, we have

$$\frac{\mathbb{E}\left[Q(T)\right] - \mathbb{E}\left[Q(0)\right]}{T} \geq -D + \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\overline{\tau}(t)\right].$$

Applying $Q(0) = 0$ to the above inequality, we have

$$\frac{\mathbb{E}\left[Q(T)\right]}{T} \geq -D + \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\overline{\tau}(t)\right].$$

Finally, letting T$\to\infty$, we have

$$\limsup_{T\to\infty} \frac{\mathbb{E}\left[Q(T)\right]}{T} \geq -D + \limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\overline{\tau}(t)\right].$$

If $Q(t)$ is mean rate stable, then $\limsup_{T\to\infty} \frac{\mathbb{E}[Q(T)]}{T} = 0$

$$0 \geq -D + \limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\overline{\tau}(t)\right].$$

Rearranging terms in the above, we have

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[\overline{\tau}(t)\right] \leq D.$$

By introducing the virtual queue $Q(t)$, we are able to convert the constraint (12) into a queue stable problem. If we can guarantee that $Q(t)$ is mean rate stable, we are able to satisfy (12). □

A. Lyapunov Optimization Formulation

Next, we define a Lyapunov function as a scalar measure of response time in the system as follows

$$L\left(Q(t)\right) \triangleq \frac{1}{2}Q^2(t). \tag{15}$$

Then we define the conditional unit-slot Lyapunov drift as follows

$$\Delta\left(Q(t)\right) \triangleq \mathbb{E}\left[L\left(Q(t+1)\right) - L\left(Q(t)\right)|Q(t)\right]. \tag{16}$$

B. Bounding Unit-slot Lyapunov Drift

Our primary aim is to optimize the upper bound of the overall response time which is the upper bound of $\Delta\left(Q(t)\right)$.

Lemma 2. For any $t \in \{0,\ T-1\}$, given any possible control decision, the Lyapunov drift $\Delta\left(Q(t)\right)$ can be deterministically bounded as follows

$$\Delta(Q(t)) \leq H + Q(t)\mathbb{E}\left[-D + \overline{\tau}(t)|Q(t)\right], \tag{17}$$

where $H = \frac{1}{2}\left[\max \mathbb{E}\left(\overline{\tau}(t)^2\right) + D^2\right]$.

Proof.

$$\begin{aligned}\Delta\left(Q(t)\right) &= \mathbb{E}\left[L\left(Q(t+1)\right) - L\left(Q(t)\right)|Q(t)\right] \\ &= \frac{1}{2}\mathbb{E}\left[Q^2(t+1) - Q^2(t)|Q(t)\right].\end{aligned}$$

Applying equation (13), we have

$$\Delta(Q(t)) = \frac{1}{2}\mathbb{E}\left[\left[\max\left[Q(t) - D, 0\right] + \overline{\tau}(t)\right]^2 - Q(t)^2|Q(t)\right].$$

For any $Q(t) \geq 0$, $D \geq 0$, $\overline{\tau}(t) \geq 0$, we have

$$\begin{aligned}\left[\max[Q(t) - D, 0] + \overline{\tau}(t)\right]^2 \leq Q(t)^2 + \overline{\tau}(t)^2 + D^2 \\ + 2Q(t)(\overline{\tau}(t) - D),\end{aligned}$$

then we have

$$\begin{aligned}\Delta(Q(t)) &\leq \frac{1}{2}\mathbb{E}\left[\overline{\tau}(t)^2 + D^2 + 2Q(t)(\overline{\tau}(t) - D)|Q(t)\right] \\ &\leq \mathbb{E}\left[\frac{\overline{\tau}(t)^2 + D^2}{2}|Q(t)\right] \\ &\quad + \mathbb{E}\left[Q(t)\overline{\tau}(t) - Q(t)D|Q(t)\right].\end{aligned}$$

Defining H as a finite constant that bounds the first term on the right-hand-side of the above drift inequality, so that for all t, all possible $Q(t)$, we have

$$H = \frac{1}{2}\left[\max \mathbb{E}(\overline{\tau}(t)^2) + D^2\right].$$

□

C. Minimizing the Drift-Plus-Cost Performance

Defining the same Lyapunov function $L\left(Q(t)\right)$ as in (15), and letting $\Delta\left(Q(t)\right)$ represents the conditional Lyapunov drift at t. While taking actions to minimize a bound on $\Delta\left(Q(t)\right)$ every time slot would stabilize the system, the resulting cost might be unnecessarily large. In order to avoid this, we minimize a bound on the following drift-plus-penalty expression instead of minimizing a bound on $\Delta\left(Q(t)\right)$

$$\Delta\left(Q(t)\right) + V\mathbb{E}\left[P(t)|Q(t)\right], \tag{18}$$

where $V \geq 0$ is a parameter that represents an "important weight" on how much we emphasize cost minimization. We add a penalty term to both sides of (17) yields a bound on the drift-plus-penalty

$$\begin{aligned}\Delta\left(Q(t)\right) + V\mathbb{E}\left[P(t)|Q(t)\right] \leq H + \mathbb{E}\left[Q(t)\overline{\tau}(t)|Q(t)\right] \\ - DQ(t) + V\mathbb{E}\left[P(t)|Q(t)\right].\end{aligned} \tag{19}$$

At each time slot, we are motivated to minimize the following term.

$$\min_{\lambda^{(F)}(t),\lambda_j^{(C)}(t),x_j(t),\forall j\in\mathcal{J}} \mathbb{E}\left[Q(t)\overline{\tau}(t)|Q(t)\right] + V\mathbb{E}\left[P(t)|Q(t)\right], \quad (20)$$

substituting (20), we have the following one-time slot optimization problem $P1$. (20) is minimized if we opportunistically minimize (21) as follows at each step [8].

$$\min_{\lambda^{(F)}(t),\lambda_j^{(C)}(t),x_j(t),\forall j\in\mathcal{J}} Q(t)\left(\frac{\lambda^{(F)}(t)}{\lambda(t)}\tau^{(F)}(t) + \sum_{j=1}^{J}\frac{x_j(t)\left(\lambda(t)-\lambda^{(F)}(t)\right)}{\lambda(t)}(\tau_j^{(X)}(t)+\tau_j^{(C)}(t))\right) + V\sum_{j=1}^{J}x_j(t)\left(P_j^{(X)}(t)+P_j^{(C)}(t)\right), \quad (21)$$

$$\text{subject to } x_j(t)\in\{0,1\}, \quad (22)$$

$$\sum_{j=1}^{J}x_j(t)=1, \quad (23)$$

$$\lambda^{(F)}(t)\le\mu^{(F)}(t), \quad (24)$$

$$\lambda_j^{(C)}(t)\le\min(\mu_j^{(C)}(t),\mu_j^{(X)}(t)). \quad (25)$$

In the next subsection, we show that if (20) is minimized at each time slot, we can achieve quantified near optimal solution.

D. Optimality Analysis

Let † denote any S-only offloading policy[1], and $\overline{\tau}^{\dagger}(t)$ and $P^{\dagger}(t)$ denote the average response time and average cost at t based on policy †.

$$\Delta(Q(t))+V\mathbb{E}\left[P(t)|Q(t)\right]$$
$$\le H+Q(t)\mathbb{E}\left[\overline{\tau}(t)-D|Q(t)\right]+V\mathbb{E}\left[P(t)|Q(t)\right] \quad (26)$$
$$\le H+Q(t)\mathbb{E}\left[\overline{\tau}^{\dagger}(t)-D|Q(t)\right]+V\mathbb{E}\left[P^{\dagger}(t)|Q(t)\right] \quad (27)$$
$$= H+Q(t)\mathbb{E}\left[\overline{\tau}^{\dagger}(t)-D\right]+V\mathbb{E}\left[P^{\dagger}(t)\right]. \quad (28)$$

Now we assume that there exists $\delta>0$ such that $\mathbb{E}\left[\overline{\tau}^{\dagger}(t)\right]\le D-\delta$ can be achieved by an S-only policy [10], and among all feasible S-only policies, $\overline{P}^{*}(\delta)$ is the optimal average cost. We have

$$(27)\le H-Q(t)\delta+V\overline{P}^{*}(\delta). \quad (29)$$

Taking expectations of (29), we have

$$\sum_{t=0}^{T-1}\mathbb{E}\left[\Delta(Q(t))\right]+V\sum_{t=0}^{T-1}\mathbb{E}\left[\mathbb{E}(P(t)|Q(t))\right]\le TH-\sum_{t=0}^{T-1}\mathbb{E}\left[Q(t)\right]\delta+VT\overline{P}^{*}(\delta).$$

[1]S-only offloading policy means that the decision on $\lambda^{(F)}(t),\lambda_j^{(C)}(t),x_j(t),\forall j\in\mathcal{J}$ depends only on the system state at t (i.e., $\lambda(t),\mu^{(F)}(t),\mu_j^{(X)}(t),\mu_j^{(C)}(t),\tau_j(t),P_j^{(X)}(t),P_j^{(C)}(t),\forall j\in\mathcal{J}$), but does not depend on $Q(t)$.

Using the law of iterated expectations as before yield and summing the above over $t\in[0,T-1]$ for some positive integer T yield, we have

$$\mathbb{E}\left[L(Q(T))\right]-\mathbb{E}\left[L(Q(0))\right]+V\sum_{t=0}^{T-1}\mathbb{E}\left[P(t)\right]\le TH-\sum_{t=0}^{T-1}\mathbb{E}\left[Q(t)\right]\delta+VT\overline{P}^{*}(\delta). \quad (30)$$

Rearranging the terms in the above and neglecting non-negative quantities where appropriate yields the following two inequalities

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[Q(t)\right]\le\frac{H}{\delta}+\frac{V\overline{P}^{*}(\delta)-\frac{V}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[P(t)\right]}{\delta}+\frac{\mathbb{E}\left[L(Q(0))\right]}{T\delta} \quad (31)$$

and

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[P(t)\right]\le\frac{H}{V}+\overline{P}^{*}(\delta)+\frac{\mathbb{E}\left[L(Q(0))\right]}{VT}, \quad (32)$$

where the first inequality follows by dividing (30) by VT and the second follows by dividing (30) by $T\delta$. Taking limits as $T\to\infty$ shows that

$$\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[Q(t)\right]\le\frac{H+V\left[\overline{P}^{*}(\delta)-\overline{P}^{*}\right]}{\delta}, \quad (33)$$

where $\overline{P}^{*}$ is the optimal long-term average cost achieved by any policy. We also have

$$\limsup_{T\to\infty}\frac{1}{T}\sum_{T_k=0}^{T-1}\mathbb{E}\left[P(t)\right]\le\frac{H}{V}+\overline{P}^{*}(\delta). \quad (34)$$

The bounds (33) and (34) demonstrate an $[O(V),O(1/V)]$ tradeoff between average response time and average cost. We can use an arbitrarily large V to make $\frac{H}{V}$ arbitrarily small, so that the inequality (34) illustrates that with the increasing of parameter V, the money cost is closer to the P^{*}. However, when V is too large, the data queue is not stable, which means the response time will exceeded the predefined system expected finish time and it's not satisfy the constraint (12) anymore. It is obvious that tuning the parameter V can minimize the response time and money cost at the same time. This comes with a tradeoff: the average response time bound in inequality (33) is $O(V)$.

E. Unit-slot Optimization

In this subsection, we need to further solve the problem $P1$ in (21)–(25) and the pseudo code of our proposed algorithm Unit-slot Optimization is given in Algorithm 1. First, we have to traverse j, to find $x_j(t)=1$. Our aim is to optimize $\lambda^{(F)}(t)$ $\left(\lambda_j^{(C)}(t)=\lambda(t)-\lambda^{(F)}(t)\right)$ when $x_j(t)=1$. This optimization is a simple optimization problem. To solve this problem, we just need to examine all feasible solutions whether satisfy that the value of the first derivate is zero. In other words, the first derivate of (21) with respect to $\lambda^{(f)}(t)$ is zero guarantees the value of (21) is the minimal one.

Algorithm 1 Unit-slot Optimization Algorithm

Set $F_{\text{opt}} = \infty$
for each $j \in \mathcal{J}$ **do**
  Set $x_j(t) = 1$. Set $x_i(t) = 0$, $\forall j \in \mathcal{J}$ and $j \neq i$
  Given the above $x_j(t)$, $\forall j \in \mathcal{J}$, optimize (21) under the constraints (23), (24) and (25). Derive optimal the $\lambda^{(F)}(t)$, and the optimal value of (21) is denoted by F
  if $F \leq F_{\text{opt}}$ **then**
    Set $F_{\text{opt}} = F$,
    set $x_j^* = x_j(t)$, $\forall j \in \mathcal{J}$
    Set $\lambda^{(F)*} = \lambda^{(F)}(t)$
  end if
end for
The optimal solution is $x_j(t) = x_j^*$, $\forall j \in \mathcal{J}$, and $\lambda^{(F)}(t) = \lambda^{(F)*}$

IV. Performance Evaluation

In this section, we first provide the details of our simulation studies, then we investigate the performance of our proposed algorithm by evaluating several performance metrics, e.g. average response time, cost and backlogs under different settings. Besides the proposed algorithm, We implemented two extra algorithms, one called Fog-First, and the other is called Cloud-First as the performance benchmarks in our experiments.

A. Simulation Environment Setting

In order to evaluate the performance of our proposed algorithm, we carried out a discrete event simulation (DES) making use of the SimPy [11]. We first established the three-tier CoT system as shown in Fig. 2 in our simulations. There were totally 200 things (this number is indicated as N) generated in the things tier by default and each of them connected to the fog node which with a relatively low but fixed service rate in the Fog tier. We also created 10 different level of communication channels connected to 10 different clouds, each level of communication channel has different transmitting rate while the corresponding cloud have various service rates. Due to the different service rate of the communication channel and cloud, the rental price of them are different (the higher service rate, the more expensive the rental price). In order to balance the distribution of processing application, we preset 10 as the maximum number of queueing length in fog node or in communication channel. What's more, regarding to our proposed algorithm, there is a penalty term V to show how much we emphasize on the average cost.

To evaluate the performance of our proposed unit-slot optimization algorithm, we also implement the "Fog-First" algorithm and the "Cloud-First" algorithm as our performance benchmarks. In the "Fog-First" algorithm, each newly arrival application is sent to fog node for processing by default until the preset queueing length on the fog node is reached. After that, The remaining applications will be sent to the cloud for processing until the queueing length on the fog node reduces. In "Cloud-First" algorithm, the applications distribution strategy is opposite to the "Fog-First" algorithm. The newly arrival applications are tend to send to cloud for their processing in the first place. Due to the factors of monetary cost and the preset backlog limitation, Some of the applications will be sent to fog node for processing.

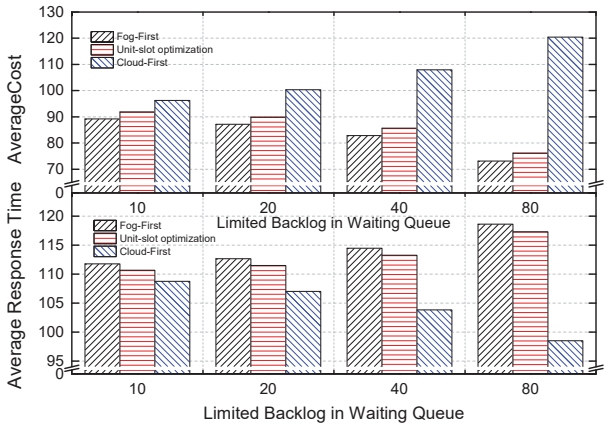


Fig. 3: The performance of processing applications in unit-slot optimization algorithms

B. Performance Evaluation of Different Algorithms

In Fig. 3, the variation of average cost and average response time of three different algorithms are shown in upper and lower parts separately with different limitation of the backlog in the waiting queue. For the "Fog-First" algorithm, the upper part of the figure demonstrates that the average cost decreases when the limitation of the backlog in the waiting queue (the maximum allowed number of applications waiting in fog nodes for processing) increases, and the lower part of the figure shows that the average response time increases when the limitation of the backlog in the waiting queue (the maximum allowed number of applications waiting in cloud center for processing) increases. This is because more and more applications are allocated to the fog node for their processing. With the free but relatively slow services on the fog nodes, the average cost is thus decreased while the average response time is increased. Our proposed unit-slot optimization algorithm represents the same cost spending trend and average response time trend as the "Fog-First" algorithm, this is due to our algorithm is capable of finding an optimized tradeoff online in terms of average cost and average response time. However, the cost spending of "Cloud-First" algorithm is increasing while the response time is decreasing due to more and more applications are attempted to be processed at the remote cloud node with much higher service rate, which is spontaneously caused the increment of cost spending and decrement of response time. Regarding to the balancing of average cost and average response time, we can clearly draw from Fig. 3 that our proposed unit-slot optimization algorithm is the best choice among three mentioned algorithms.

C. The Impact of Different Penalty Terms on the Performance of the proposed Algorithm

To investigate the effects of the penalty term V in our proposed unit-slot optimization algorithm, we varied its value and observed how the average cost and average response time change accordingly. In this experiment, the value of V

represents the importance of the average cost spending will impact the system performance. As depicted in Fig. 4, by setting the value of V to a small one, we observed that the average response time remains small while the average total cost remains large. This is due to the penalty of cost spending is low in the system, so that the applications are sent to cloud for their processing in order to get a better response time. However, by setting V to a large value, the significant decrement of the average total cost and increment of the average response time can be observed. Furthermore, when the average response time drops, the average cost grows remarkably.

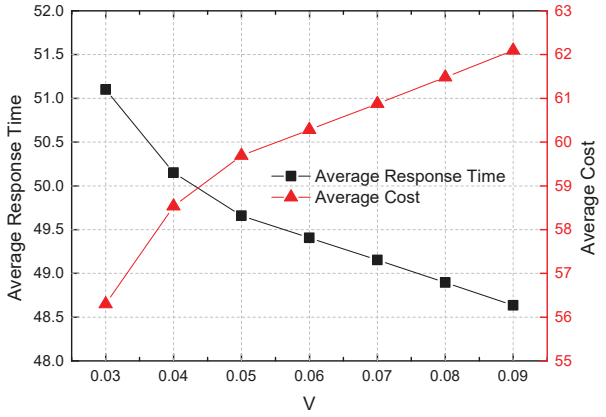


Fig. 4: The performance tradeoff on processing applications in different algorithms

Apart from studying the impact of the change of V on average cost and average response time, we also study how the backlog of the queue in the system is affected. In Fig. 5, the upper part depicts the number of backlog in fog node and communication channel on a smaller V, and the lower part depicts the number of backlog in fog node and communication channel on a larger V with time increasing by using the unit-slot optimization algorithm. As shown in the Figure, the backlog in Fog tier is increasing from 0 to the preset limit of the backlog (set as 10 in our experiment) very quickly. After that, the applications will be sent to cloud for processing until space becomes available in the fog node. In addition, the larger V is given, the shorter time of reaching the limitation of the queue backlog in fog node is observed.

D. The Impact of Different Length of Time Slot on the Performance of the Proposed Algorithm

Fig. 6 depicts the changes of both average response time and average total cost of the unit-slot optimization algorithm according to the increase of the length of time slot. In this experiment, we use how many applications are expected to be processed to set the length of time slot. As shown in the Figure, the average response time of our proposed algorithm increases almost linearly since more applications need to be processed but the processing capacities of the system remain the same. Meanwhile, the average cost grows more smoothly since once the backend service providers are determined, the cost rate is also fixed.

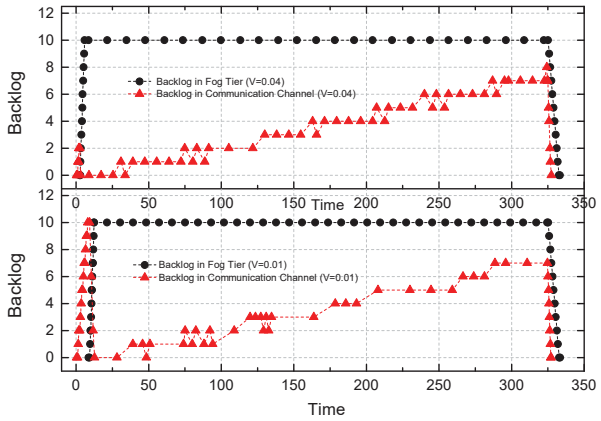


Fig. 5: The backlog number of processing applications by unit-slot optimization algorithm

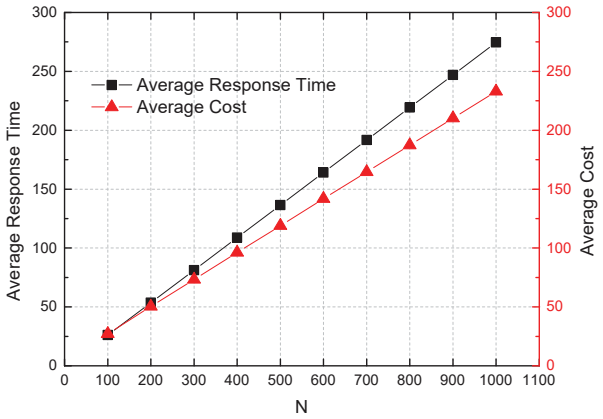


Fig. 6: The performance of processing applications by using unit-slot optimization algorithm

V. RELATED WORK

A. IoT with Fog Computing

The integration of Cloud Computing and IoT has drawn quite some attention from different aspects, such as cloud support for the IoT [12], devices virtualization and service provisioning [13], and computation offloading in mobile clouds [14]. Along with the introduction of Fog Computing, three-tier CoT systems start becoming a hot spot recently. There was a theoretical model of three-tier CoT systems proposed in [15]. With the aim of minimizing the system resource usage while meeting the defined QoS requirements, the authors studied the joint optimization problem of service provisioning, data offloading and task scheduling. In [16], the authors focused on studying the interactions between fog layer and cloud layer in such systems. They mathematically characterized a Fog Computing network in terms of power consumption, service latency, CO_2 emission, and cost. The authors in [17] modeled the Fog Computing system as a Markov decision process, and provided a method to minimize operational costs while providing required QoS. In [18], the authors aimed to minimize the task completion time while providing a better user experiences by utilizing the knowledge of task scheduling and resource management in Fog Computing system. The above works did not consider the propagation delay incurred by the transmission

in communication channels and make processing decision in a distributed way. To the best of our knowledge, this work is the first one to address the Delay-sensitive applications in the three-tier CoT systems in an online distributed control manner.

B. Lyapunov Optimization Approach in Fog and Cloud Computing

In dynamic systems, Lyapunov optimization is a promising approach to solve resource allocation, traffic routing, and scheduling problems. It has been applied to the works of traffic routing and scheduling in wireless ad hoc networks [19], [20], but they did not consider processing applications in the processing network environment.

In processing networks, [21] proposed a method on how to offload one application from a mobile device to a cloud datacentre for its processing. [10], [22], [23] studied processing networks with limited numbers of network nodes, while we allow arbitrarily large numbers of things, nodes and datacentres in the system. In [24], the authors focused on studying task processing and offloading in a wired network, but they overlooked to address the potential conflict of performance requirements between them.

VI. Conclusion

Three-tier Cloud of Things is a promising model that if well exploited can provide efficient data processing for IoT applications. An increasing number of smart devices are emerging and joining the Internet, thus giving rise to a huge amount of data. Those data has to be processed in either Fog or Cloud tier as fast as possible with a reasonable cost. In this paper, we studied the problem of providing cost-effective data processing service, and proposed an efficient and effective online algorithm for balancing the tradeoff between data processing time and cost. Simulation results demonstrated that the proposed algorithm successfully achieves its goals. In the future, we plan to investigate the uncertainties of the communication among the components within the three-tier Cloud of Things systems. Particularly, the communication between the things and the fog can be constructed in a much more reliable, safe, real-time, secure, way than between the fog and the cloud.

VII. Acknowledge

Dr. Wei Li's work is supported by Faculty of Engineering and IT Early Career Researcher scheme, The University of Sydney, and the Faculty of Engineering & Information Technologies, The University of Sydney, under the Faculty Research Cluster Program. Professor Zomaya's work is supported by an Australian Research Council Discovery Grant (DP130104591). Flavia C. Delicato and Paulo F. Pires as CNPq fellows.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[2] M. Aazam, I. Khan, A. A. Alsaffar, and E. N. Huh, "Cloud of things: Integrating internet of things and cloud computing and the issues involved," in *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014*, Jan 2014, pp. 414–419.

[3] S. Distefano, G. Merlino, and A. Puliafito, "Enabling the cloud of things," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, July 2012, pp. 858–863.

[4] M. Aazam and E. N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, Aug 2014, pp. 464–470.

[5] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. Santa Clara, CA: USENIX Association, 2015.

[6] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.

[8] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[9] P. Bocharov, *Queueing Theory*, ser. Modern probability and statistics. VSP, 2004.

[10] Y. Niu, B. Luo, F. Liu, J. Liu, and B. Li, "When hybrid cloud meets flash crowd: Towards cost-effective service provisioning," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1044–1052.

[11] N. Matloff, "Introduction to discrete-event simulation and the simpy language," *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, vol. 2, p. 2009, 2008.

[12] C. M. D. Farias, W. Li, F. C. Delicato, L. Pirmez, A. Y. Zomaya, P. F. Pires, and J. N. D. Souza, "A systematic review of shared sensor networks," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 51:1–51:50, Feb. 2016.

[13] A. Botta, W. de Donato, V. Persico, and A. Pescape, "Integration of cloud computing and internet of things: A survey," *Future Generation Computer Systems*, vol. 56, pp. 684 – 700, 2016.

[14] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3317–3329, Dec 2015.

[15] W. Li, I. Santos, F. C. Delicato, P. F. Pires, L. Pirmez, W. Wei, H. Song, A. Zomaya, and S. Khan, "System modelling and performance evaluation of a three-tier cloud of things," *Future Generation Computer Systems*, pp. –, 2016.

[16] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2015.

[17] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205 – 228, 2015, special Issue: Performance 2015.

[18] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–1, 2016.

[19] M. J. Neely, E. Modiano, and C. P. Li, "Fairness and optimal stochastic control for heterogeneous networks," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 396–409, April 2008.

[20] M. J. Neely, "Energy optimal control for time-varying wireless networks," *IEEE Transactions on Information Theory*, vol. 52, no. 7, pp. 2915–2934, July 2006.

[21] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, June 2012.

[22] S. Li, Y. Zhou, L. Jiao, X. Yan, X. Wang, and M. R. T. Lyu, "Towards operational cost minimization in hybrid clouds for dynamic resource provisioning with delay-aware optimization," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 398–409, May 2015.

[23] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *arXiv preprint arXiv:1605.05488*, 2016.

[24] A. Destounis, G. S. Paschos, and I. Koutsopoulos, "Streaming big data meets backpressure in distributed network computation," *CoRR*, vol. abs/1601.03876, 2016.

Smart Scene Management for IoT-based Constrained Devices Using Checkpointing

François Aïssaoui[1], Gene Cooperman[1,2], Thierry Monteil[1], Saïd Tazi[1]
[1]LAAS-CNRS, Université de Toulouse, CNRS, INSA, UT1 Capitole, Toulouse, France
[2]College of Computer and Information Science Northeastern University, Boston, MA / USA
Emails: aissaoui@laas.fr, gene@ccs.neu.edu, {monteil, tazi}@laas.fr

***Abstract*—Typical devices of the Internet of Things are usually under-powered, and have limited RAM. This is due to energy and cost concerns. Yet, IoT applications require increasingly complex programs with increasingly large amounts of data. In principle, an application could manage the increasing data within the limited RAM by saving and loading data from the file system as needed. But managing the use of RAM in this way is both time-consuming and error-prone for the code developer. We propose instead a novel architecture in which different semantic scenes are implemented as independent operating system processes. As the need arises to switch from one scene to another, the currently running process, which represents the current scene, is checkpointed and a process representing the new scene is restarted from a checkpoint image. This solution employs checkpointing to provide a simpler framework for the end programmer, while at the same time resulting in higher performance. For example, experiments show that restarting an old process from a checkpoint image is about 25 times faster than starting a new process. When using an mmap-based optimization (deferring the paging in of virtual memory pages until runtime), restarting an old process is about 500 times faster. Overall, checkpoint and restart each execute in less than 0.2 seconds on a Raspberry Pi B.**

***Index Terms*—Semantics, Scene Management, Advances Driver Assistance System, Internet of Things, Checkpointing, Constrained Devices**

I. Introduction

The Internet of Things (IoT) is fast becoming a critical technology in the evolution toward a "connected world". However, IoT faces some challenges in terms of optimization on performance-constrained devices and gateways. The CPUs are of low performance, with minimal associated RAM. This is intrinsic to many IoT scenarios, such as Advanced Driver Assistance Systems (ADAS: "smart cars"), drones, etc., which are constrained by limited battery size and by cost concerns.

In many applications, IoT software is consuming increasing amounts of memory, due in part to increasingly complex behavior with increasingly complex semantics as software requirements continue to evolve. Virtual memory is usually not available, since IoT devices involve real-time computation, and paging in virtual memory makes running times difficult to predict. So, the central question is:

> *how to write a large program with large data that does not naturally fit in the small RAM available.*

We propose a novel software architecture that is well-adapted to RAM-constrained devices for IoT. Specifically, we will target a simplistic model of Advanced Driver Assistance System (ADAS) in order to illustrate the architectural benefits of easier scene-management for the end programmer and improved performance through efficient checkpoint-restart. We implement this model on the ARM-based Raspberry Pi Model B computer, as an example of a low-performance CPU configuration with limited RAM.

The proposed software architecture consists of smart scene management, using semantic modelling and rule-based reasoning. Each scene is represented by an operating system process, and a checkpointing mechanism is applied to save the state of the process in a checkpoint image file.

Such a scene represents a partial view of the context. It contains information about the spatial context, the road conditions, the participants, etc. Since the memory for a single scene can be huge, one typically does not have sufficient RAM to load two scenes at the same time within small or embedded computers such as the Raspberry Pi. The novelty here consists of using checkpoint-restart to manage RAM by saving the old scene and restore a new scene. This performs better than writing a monolithic program that includes all scenes, which would have to rely on the random memory access of virtual memory to page in the memory of scenes on demand.

Semantic web technologies such as the Web Ontology Language (OWL) formalism allow one to represent knowledge using a description logic. It allows for the creation of vocabularies that can be shared, along with a set of rules to apply to the model using *semantic reasoners*. We propose a new model for Scene management, with a specific set of properties and a possible link to application model such as ADAS.

The remainder of the paper is organized as follows. Section II presents a brief overview of semantic web principles and technologies in the context of the IoT and some background on checkpointing. In Section III, the presentation of a Ideal Global Architecture of our Scene system is provided, presenting the models and rules used. Section IV presents the specific contributions of our work. Section V analyzes the experiment results. Then we conclude this paper in Section VI.

II. Background

This section provides an overview of the literature for the technologies employed in this paper. A brief overview of IoT is provided first. This is followed by a discussion

978-1-5090-3216-7/16/$31.00

of the requirements for semantics for IoT. Finally, we discuss checkpointing and introduce Distributed Multi Threaded CheckPointing (DMTCP).

A. IoT Overview

The IoT is an important area for innovation due to the large numbers of possible applications [1]. This presents a vision of a world-wide network of interconnecting physical (sensors, activators, complex objects like cars). The vision also includes virtual objects able to interact and affect the real world create a significant number of challenges [2]. In most cases, these objects have strong constraints in term of energy, communication and/or processing [3].

B. Semantic Web Technologies for IoT

Semantic computing [4] is an emerging and rapidly evolving interdisciplinary field that originated from artificial intelligence. It consists of applying models and standardized technology describing the semantics of the linked objects to enable interactions and interoperability between different components (software or hardware). It is a recommended best practice in the domain of IoT [5]. However, one of the problems facing users of semantic technologies is that the semantic information increases the complexity and processing time, and is therefore unsuitable for dynamic and responsive environments such as IoT. Complex models require greater CPU processing and therefore are not suitable for constrained environments such as IoT. The earlier proposal of the W3C [6] takes this difficulty into account by providing a lightweight ontology specially adapted for IoT.

We use the OWL formalism to represent the data and the associated knowledge. OWL is a description language based on linked data and share vocabularies. Semantic Web Rule Language (SWRL) is a rule-based language that describes what could happen when the knowledge base changes, or when an event happens. It allows one to express abstract rules to be applied in the model.

C. Checkpointing using DMTCP

A checkpointing mechanism consists in creating images (snapshots) of a process and being able to recreate the process from this image.

Checkpointing has a long history in HPC [7]–[10]. In 2012, a cluster of ARM CPUs was tested with respect to checkpointing as a basis for power-efficient HPC [11]. This used the more powerful ARM Cortex-A9 CPU, whereas the current Raspberry Pi Model B uses the less powerful ARM Cortex-A7. In those earlier experiments, checkpoint times from 3.4 to 138 seconds were observed on various NAS parallel benchmarks for MPI — a standard test suite for parallel applications. In comparison, the experiments of this work apply checkpointing only to a single process.

In Section V, DMTCP [12] is used to create checkpoint image files from running processes. DMTCP-style checkpointing is *transparent*, in that the original application binary is not modified, and the target process is not aware of DMTCP.

In this work, the latter approach is used to allow the application to change scenes on demand. The application program can be further modified through the use of DMTCP plugins [13]. Plugins are used to virtualize resources, so that the application can be restarted in a new environment, independently of changing physical names such as pathnames, process ids (PIDs), etc.

DMTCP also supports options for two well-known optimizations that enhance the speed of checkpoint and restart. The first is "*Forked Checkpointing*". DMTCP forks a child process, which executes the checkpoint. This takes advantage of the well-known operating system support for copy-on-write between the parent and child processes. The parent process continues to execute without blocking, while the child process writes memory and other state into the checkpoint image file.

The second optimization option is "*Fast restart*", based on the Linux mmap system call. The mmap call maps the checkpoint image file to RAM, but the data is not actually copied to RAM until the virtual memory subsystem pages it in. Thus, execution begins early after restart, paging in only the actively used pages, and without waiting for all of the checkpoint image file to be loaded.

III. Ideal Global Architecture: Scenes and Semantics

This section presents the semantic concepts associated with the Scene concept, along with the rules used to manage the Scenes.

A. Semantic Models Used

A Scene is defined as a partial view of the context. Several scenes are created according to the needs of the application. Only one Scene is loaded at any given time. The Scene includes the destination of the vehicle, along with a possible path. It also contains a partial representation of the map ontology used in [14]. Since the entire map is split among different scenes, this lowers the system complexity through rules to navigate from one scene to another.

Figure 1 shows a semantic model with some example relations in the semantic class Scene. The last relation of the Scene is its *Specificity*. This relation represents, for example, a location specificity, or a time-of-day specificity (e.g., day and night). This associates with the Scene specific characteristics that enable the reasoner to choose the best target scene to switch to.

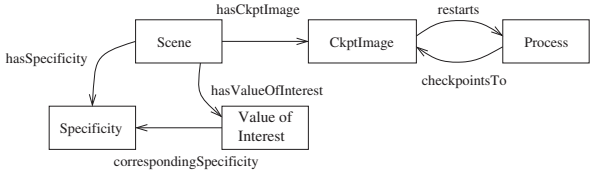


Fig. 1. Overview of Scene representation and link between Scenes and Checkpointing Mechanism

A second model that is linked to the Scene is used to guide the *Checkpointing*. Figure 1 shows the link to the Check-

pointing model classes. The class *ValueOfInterest* is used to characterize the values that are important for other Scenes. They are linked to the *Specificity* class by a *corresponding-Specificity* property for that class. This property links a value to a specific type of scene. Thus, the Scene is linked to a *CkptImage* class by the *hasCkptImage* object property. This allows the reasoner to identify the available checkpoint images for a Scene. The checkpoint image is then linked to a process. Two types of relations are possible: 1) the process has been checkpointed into a CkptImage (shown via *checkpointsTo*); or 2) a CkptImage is used to restart a process (a *restarts* relation is created between the CkptImage and the restarted process).

B. Scene Hierarchy

Since each scene represents a partial view of the global state, a classification of the scenes is needed. A hierarchy is used in which each scene (except for the root scene) has a parent scene.

A "child" scene inherits parameters and rules from its parent scene and adds additional, more specialized information. That information might be, for example, information about the type of location (e.g., what city, or what neighborhood in a vehicular context) and is considered to be *static* in the sense that it does not change over time. In contrast, each specialized scene also has *dynamic* information. An example is the specific road conditions, which might depend on road work in progress.

A hierarchical classification of this type allows one to create lightweight scenes, each of which has more specialized information than the parent in the hierarchy.

C. Rules for Scene Management

Two models are considered in Section III-A. Specific rules for each model are used and will be described.

The first model includes a set of rules that affects the vehicle and its actions. This model is used to analyze the car sensor data (e.g., its position). It will allow the system to react according to the current context.

The second rule-based model from Section III-A is used to guide the checkpointing mechanism for the scene management. This model is in charge of gathering enough information from the system to infer that a change of scene is required. In this case, the rules cause the process in charge of the scene management to checkpoint the current scene and then to load the second one.

D. Shared Information

The checkpointing mechanism allows the state of a running process to be serialized into a file. But some information and knowledge acquired by the first scene must then be passed to the second scene.

As described in Section III-B, the scenes are derived from a hierarchical classification. This classification allows the system to provide relevant information to the next scene. For instance, the whole system shares information from the car sensors and geographical location. This general information is stored and defined by the root scene of the system, which will be shared by all sub-scenes.

With such a mechanism, the system is able to share information between different scenes, according to the relevance of the data for the next scene. Such a mechanism allows one to reduce the amount of information handled by the system and the reasoner. This mechanism is implemented using the DMTCP plugins discussed in Section II-C.

The information to share can be retrieved from the model using the property *hasValueOfInterest* of the Scene. This relation is shown in Figure 1.

IV. Efficient RAM Management for IoT and Embedded Systems

In principle, the use of scenes within a large, global hierarchy can be implemented as a single large process. However, typical IoT-based embedded systems are restricted to small RAM without any virtual memory. For this reason, we represent each scene of the global hierarchy as a separate operating system process. Only one process (the current scene) runs at a time. We demonstrate that switching between scenes can be made efficient through the use of checkpointing. The original scene (with all of its internal state) is checkpointed, and a new scene is restarted from a previous checkpoint image.

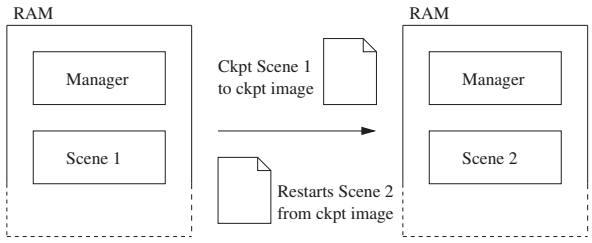


Fig. 2. Proposal of new architecture for Scene Management. Each rectangle represents a process.

Figure 2 illustrates the proposed architecture. The data to be handled is split into multiples scenes, which contain information and rules as described in Section III-A. Each scene is represented as an individual process. A *Scene Manager* is used to checkpoint and restart the process that represents a scene.

This enhancement provides a simpler way for the end programmer to design the architecture and the data handling of its program and is evaluated in the next section. We demonstrate the efficiency of such a system compared to a standard initialization of a process.

V. Experimental Evaluation

In this section we evaluate the system presented in Section III and Section V. Here, we discuss the additional time needed when a checkpoint is invoked, and the time needed to restart a scene from a checkpoint image file. Then we compare this restart time to a traditional approach, which consists of dynamically reading the data files. Finally, we discuss the runtime overhead introduced when the process is executed under the control of DMTCP, as opposed to executing the process natively.

A. Experimental Environment

These experiments use a *Raspberry Pi 2 Model B* with one GB of RAM. In these experiments, we emphasize the limited RAM of a constrained embedded system by restricting ourselves to a more limited 256 MB of RAM. This was also the RAM provided with the earlier Pi 1 Model A+. The files containing the scenes and the images files for the experiment are stored in the file system of the SD card of the Raspberry Pi.

For the checkpointing software, DMTCP version 3.0.0 was used, available at its github repository.

B. Checkpoint and Restart

As a first case for evaluation, we analyze the checkpoint and restart times on the Raspberry Pi. The size of the input files is varied in order to find the relation between the size of the files and the checkpoint-restart time.

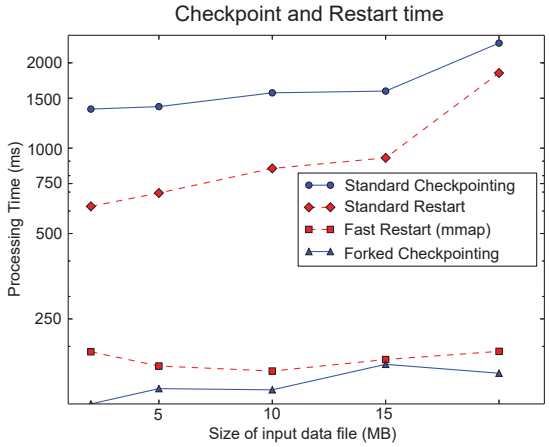


Fig. 3. Checkpoint and restart time

Two sets of experiments are discussed. First the standard checkpointing and restart mechanism is used. In Figure 3, the two lines at the top of the graph show the time needed for a standard checkpoint and restart the Java program with the Jena library and the data files loaded. The "standard" times refer to the case when the DMTCP optimizations of forked checkpoint and mmap-based fast restart (see Section II-C) are not used. The times vary as the size of the scene file is varied. Note that a logarithmic y-axis is used for the checkpoint and restart times. The largest file used is about 20 MB, which serves as a placeholder for the actual scene-related data that would be used in a realistic ADAS application. It is assumed that the operating system must execute in RAM along with the application in a real-time system. Recall that the goal of these experiments is to simulate a low-cost embedded device, with only 256 MB of RAM.

The time to checkpoint and restart grows slightly when the size of the scene-related data increases. This is expected, since DMTCP must map the process image to the checkpoint file (or reverse for restart operation) and this operation is slower if there is more data to save to a file (or to load from a file). The unoptimized checkpointing times of Figure 3 vary from 1.5 s to about 2 s. This is reasonable for energy-constrained devices such as the Raspberry Pi, but it can be improved to be more responsive. Similarly, the unoptimized restart times vary from about 600 ms to 1.5 s.

In order to further improve responsiveness, a second experiment (also presented in Figure 3) shows the impact of using the two DMTCP optimizations discussed in Section II-C: forked checkpointing and mmap-based fast restart. These optimizations improve the checkpoint/restart times (and hence the responsiveness) by a further factor of ten.

The first line from the bottom of Figure 3 shows the time for the Forked Checkpointing. This Forked Checkpointing operation is about 5 to 10 times faster than the Standard Checkpointing and allows the running process to be available more time — since the Checkpointing operation freezes all threads to avoid any error in the memory of the process. The checkpoint operation is done by the child process and the time to make this operation is equivalent to the Standard Checkpointing. The times are reduced to about 150 ms to 200 ms for the running process. Since the times are close to the minimum quantum of times given to the thread, we expect some variations in the checkpoint time, as exemplified by the slightly higher checkpoint time for a file size of 15 MB.

The Fast Restart time is the second curve from the bottom in Figure 3. The time for fast restart operation is nearly constant as the file size varies. This is the mmap optimization defers loading of most of the virtual memory pages. From our experiment, we see that the Fast Restart operation is about 3 to 10 times faster than the Standard Restart.

TABLE I
SIZE OF CKPT IMAGE DEPENDING ON INPUT FILE

Input file (MB)	2.0	5.1	10.2	15.4	20.5
Ckpt image (MB)	86.8	98.4	143.5	157.1	179.7

Table I shows the checkpoint image size as a function of the input file size. The checkpoint image size increases with the size of the input file, since the file data has been loaded into RAM during initialization. The image is large compared to the 2 MB input file, since the process is Java-based. The JVM must be checkpointed along with the loaded classes. The checkpoint image file size is also large because of the large Java classes running in the JVM. The size of the checkpoint image file increases more in absolute terms than the increase in size of the input file. This is because the data loaded are submitted to a semantic reasoner. This reasoner infers new knowledge that has been stored into the RAM and then must be saved as part of the checkpoint image.

C. Startup Times

In the second experiment, we discuss the difference in execution times between a restart and launching a fresh, new process that need to load data from a file.

Figure 4 shows the execution times in different situations. The diamond-shaped and square plotted points represent the restart times for a checkpoint image. The square plot uses the

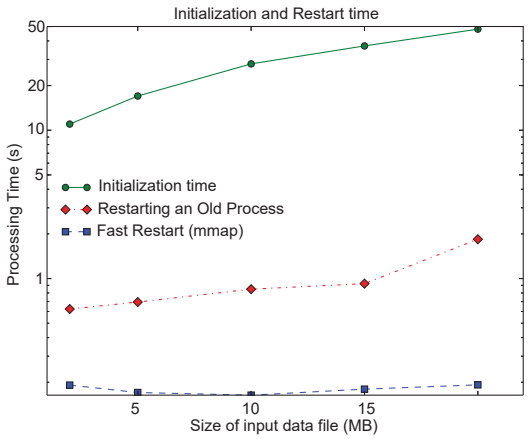


Fig. 4. Initialization of a new process versus restart of a previously checkpointed process. This compares the time for restarting a new process using the techniques of this work, versus the traditional alternative of starting (initializing) a new process for each new scene. Restarting an old process is about 25 times faster (and 500 times faster when using the mmap-based fast restart optimization). This is because restart avoids any data initialization that is executed by the Scene framework itself before it gives control to the end programmer.

mmap-based Fast-Restart option. The round plot represents the initialization time of the process when reading the data from the file. The initialization and restart times grow with the size of the input file that is loaded. Of the total initialization time, about 2 to 4 seconds is required solely to start the JVM before reaching the "main" method of the ADAS framework. The remaining time is used to load the Java-based semantic libraries and the input data.

Collecting together JVM startup, semantic library startup and loading the initial data, Figure 4 shows that "Restarting an Old Process" is about 25 times faster than the standard execution startup of a new process in the ADAS framework. Further, the Fast Restart method is about 500 times faster than the standard initialization.

VI. Conclusion and Future Work

A new software architecture was presented that allows one to manage the RAM usage efficiently for Internet of Things (IoT) devices, and more generally for performance-constrained devices. A mechanism is used to checkpoint a process in order to make RAM available for a new process, which will be restarted from a checkpoint image file. A large, monolithic process would not be a good alternative, since the delays due to virtual memory paging are not consistent with real-time programming. We demonstrated that the proposed architecture is about 25 times faster than the standard startup of a new process (see Figure 4). When used with mmap-based fast restart (thus deferring paging in of virtual memory until runtime), the proposed architecture can even be 500 times faster. This work has been applied to an Advanced Driver Assistance Systems (ADAS) domain as an example, but the scene concept is generic and can be equally well applied to other problems in the Internet of Things.

This work has simulated the operating system characteristics and expected performance of an example scene-based architecture for ADAS as a proof-of-principle. The ADAS example itself is not intended as a realistic system for production. In future work, we will apply this to a full-fledged domain in the Internet of Things integrating management of the connectivity of multiple devices and real time constraints.

Acknowledgments

This work has been supported by a "Chaire d'Attractivité" of the IDEX Program of the Université Fédérale de Toulouse Midi-Pyrénées under Grant 2014-345, and by the National Science Foundation under Grant ACI-1440788.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] A. Whitmore, A. Agarwal, and L. Da Xu, "The Internet of Things — a survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[3] K.-C. Chen and S.-Y. Lien, "Machine-to-machine communications: Technologies and challenges," *Ad Hoc Networks*, vol. 18, pp. 3–23, 2014.

[4] P. C. Y. Sheu, H. Yu, C. V. Ramamoorthy, A. K. Joshi, and L. A. Zadeh, *Semantic Computing*. John Wiley and Sons, 2010.

[5] M. Serrano, P. Barnaghi, F. Carrez, P. Cousin, O. Vermesan, and P. Friess, "Internet of Things IoT semantic interoperability: Research challenges, best practices, recommendations and next steps," IERC: European Research Cluster on the Internet of Things, Tech. Rep., 2015, http://www.internet-of-things-research.eu/pdf/IERC_Position_Paper_IoT_Semantic_Interoperability_Final.pdf.

[6] M. Bermudez-Edu *et al.*, "IoT-lite ontology, a submission to the W3C," Nov. 2015, https://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/.

[7] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny, "Checkpoint and migration of UNIX processes in the Condor distributed processing system," Technical Report, Tech. Rep., 1997.

[8] P. Hargrove and J. Duell, "Berkeley Lab Checkpoint/Restart (BLCR) for Linux clusters," *Journal of Physics Conference Series*, vol. 46, pp. 494–499, Sep. 2006.

[9] J. Cao, G. Kerr, K. Arya, and G. Cooperman, "Transparent checkpoint-restart over InfiniBand," in *Proc. of the 23rd Int. Symp. on High-performance Parallel and Distributed Computing*. ACM Press, 2014, pp. 13–24.

[10] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," *Supercomputing Frontiers and Innovations*, vol. 1, no. 1, pp. 5–28, 2014.

[11] K. L. Keville, R. Garg, D. J. Yates, K. Arya, and G. Cooperman, "Towards fault-tolerant energy-efficient high performance computing in the cloud," in *2012 IEEE International Conference on Cluster Computing*. IEEE, 2012, pp. 622–626.

[12] J. Ansel, K. Aryay, and G. Cooperman, "DMTCP: Transparent checkpointing for cluster computations and the desktop," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, may 2009, pp. 1–12.

[13] K. Arya, R. Garg, A. Y. Polyakov, and G. Cooperman, "Design and implementation for checkpointing of distributed resources using process-level virtualization," in *Proc. of 2016 IEEE Computer Society International Conference on Cluster Computing*. IEEE Press, 2016, to appear.

[14] L. Zhao, R. Ichise, S. Mita, and Y. Sasaki, "Ontologies for Advanced Driver Assistance Systems," in *The 35th Semantic Web & Ontology Workshop (SWO)*, Mar. 2015, pp. 1–6, http://www.ei.sanken.osaka-u.ac.jp/sigswo/papers/SIG-SWO-035/SIG-SWO-035-03.pdf or http://sigswo.org/papers/SIG-SWO-035/SIG-SWO-035-03.pdf.

Byzantine Reliable Broadcast in Sparse Networks

Sisi Duan
Oak Ridge National Laboratory
Email: duans@ornl.gov

Lucas Nicely
University of Tennessee, Knoxville
Email: lnicely@vols.utk.edu

Haibin Zhang
University of Connecticut
Email: haibin.zhang@uconn.edu

***Abstract*—Modern large-scale networks require the ability to withstand arbitrary failures (*i.e.,* Byzantine failures). Byzantine reliable broadcast algorithms can be used to reliably disseminate information in the presence of Byzantine failures.**

We design a novel Byzantine reliable broadcast protocol for loosely connected and synchronous networks. While previous such protocols all assume correct senders, our protocol is the first to handle Byzantine senders. To achieve this goal, we have developed new techniques for fault detection and fault tolerance. Our protocol is efficient, and under normal circumstances, no expensive public-key cryptographic operations are used.

We implement and evaluate our protocol, demonstrating that our protocol has high throughput and is superior to the existing protocols in uncivil executions.

***Index Terms*—Byzantine broadcast, reliable broadcast, fault detection, fault tolerance, sparse networks.**

I. Introduction

Byzantine failures occur when a participant in a distributed system deviates arbitrarily from the protocol specification, *e.g.,* due to a software bug or a cyber attack. Protocols detecting or tolerating such failures are particularly appealing for modern distributed systems and network applications that increasingly grow larger (*e.g.,* clouds, cryptocurrency systems).

We study how to efficiently achieve reliable broadcast in sparse networks that are subject to Byzantine failures. A sparse network is a network with a low number of links. Examples of sparse networks include sensor, robotic, and *most* real-world networks [19], [27].

Previous Byzantine broadcast protocols for sparse networks [23]–[25] assume that senders (*i.e.,* source nodes) who broadcast messages are *correct*, while a fraction of non-source nodes can be Byzantine. These protocols are essentially Byzantine variants of *best-effort broadcast* protocols, which guarantee that all correct processes deliver the same set of messages only if the senders are correct. If the sender is faulty or behaves maliciously, some nodes may deliver the message while others may not. Our goal is to build stronger broadcast protocols in the customary sense of *reliable broadcast*, one that handles Byzantine senders.

Before proceeding to our protocol properties, we will review and clarify models of distributed systems in terms of channel and graph connectivity.

Channel. When considering Byzantine failures, we need methods to provide correct sender identification. One may generally assume *authenticated channels*: if a correct process delivers a message with a correct sender, the message was previously sent by the sender. This model is also known as the *full Byzantine model*. Basing protocols on this model is desirable because it maximizes the fault models. One may choose either cryptographic techniques (such as message authentication codes (MACs) and digital signatures), or non-cryptographic techniques [33].

A number of works (*e.g.,* [23]–[25]) regard protocols designed in the full Byzantine model as "cryptography-free protocols." This view is slightly problematic, because authenticated channels using digital signatures, message authentication codes, or a combination of both (*e.g.,* SSL, TLS) dominate Internet communication. (But the view is correct in the sense that it does not rely on any *specific* cryptographic primitive such as digital signatures.)

It is more efficient to implement the protocols using authenticated channels with MACs because MACs are much less expensive than digital signatures. Therefore, while maximizing the number of fault models to which the protocols can be applied, it is desirable to design Byzantine resilient protocols assuming authenticated channels due to efficiency concerns. It is less convincing to argue that protocols in full Byzantine model "do not require a trusted infrastructure," because authentication requires an extensive setup—there must be an agreed upon setup for the authentication model.

Lastly, we must clarify four points. First, most Byzantine resilient protocols that explicitly use MACs, such as PBFT [9] "[c]an be modified easily to rely only on point-to-point authenticated channels," as commented in [9, pp. 402]. Second, it is unnecessarily true that all Byzantine resilient protocols using MACs also work in the full Byzantine model, because it is difficult, if not impossible, to transform a handful of protocols which use MACs in a more complex manner [2], [16], [35] into ones assuming only authenticated channels. Third, many protocols that claim to be "cryptography-free" are the most efficient cryptographic solutions in practice because they can be implemented simply using MACs. For instance, the broadcast protocol that tolerates Byzantine non-source nodes in sparse networks [25] leads to the most efficient MAC based protocol. Four, if MACs are inadequate for designing a cryptographic solution, another design choice (see, *e.g.,* [2], [21]) is to optimize the gracious execution (*i.e.,* the case without failures) and to use signatures only for uncivil executions (*i.e.,* the case with failures).

Graph connectivity. We briefly review the graph model for the Byzantine broadcast case. Most of these protocols are designed for completely connected graphs [6], [9], [22], which

978-1-5090-3216-7/16/31.00

attempt to tolerate a *maximum* number of Byzantine failures. Dolev [13] was the first to show that in order to tolerate k Byzantine failures, it is necessary and sufficient that the network is $(2k+1)$-connected, given that there is at least $3k+1$ nodes. Later, Nesterenko and Tixeuil [26] generalized the result in a manner that the topology is unknown and the environment is asynchronous.

Other groups have considered the *density* of Byzantine failures. These cases can be divided into two categories—those for dense networks [5], [20], [28] and those for sparse networks [23]–[25]. The protocols for dense networks are not adaptable for use in sparse networks, because if they were, the number failures that they can tolerate would be very small. In sparse networks, the failures are best measured as the distance between any two Byzantine nodes. A significant drawback of the protocols in sparse networks is that they *all* consider only non-faulty senders, and thereby are not secure in the *customary* sense of reliable broadcast. Assuming that the sender is always correct apparently limits the scope of deployment of these broadcast protocols in practice. Our primary goal is to build a reliable broadcast protocol that also tolerates Byzantine senders in an efficient manner.

Our contributions can be summarized as follows:
1) We present the first Byzantine reliable broadcast protocol in sparse networks that also handles Byzantine senders in synchronous environments. Our protocol is based on Maurer and Tixeuil (MT) [25], and has the following features:

- Our protocol provides multi-tiered security. Namely, when the networks are synchronous, it tolerates Byzantine senders; in settings where the networks may be asynchronous and senders are correct, our protocol still provides meaningful consistency guarantees.
- Our protocol is optimal in its gracious execution where there are no failures. Even in its uncivil execution, our protocol remains more efficient than the existing protocols with similar goals.

2) We develop novel techniques in both fault tolerance and fault detection, which are of independent interests and may be applicable to some other scenarios such as secure routing.
3) We implement and evaluate our protocol. Our experimental evaluation shows that our protocol has high throughput and high failure resilience.

II. Related Work

Byzantine broadcast: Additional related work. In the context of Byzantine failures, two classic broadcast notions are *consistent broadcast* and *reliable broadcast*. The notion of consistent broadcast was implicit in earlier papers on the topic [6], [7], [34]. Byzantine consistent broadcast ensures only that the *delivered requests* are the same for all receivers. Byzantine reliable broadcast, also known as the "Byzantine generals problem" [22], additionally guarantees that either all correct parties deliver some request or none delivers any. For instance, Bracha's broadcast [6], one that assumes only authenticated channels, is a well-known implementation of Byzantine reliable broadcast for complete graphs.

For sparse graphs, previous works [23]–[25] all assume correct senders. They fail to provide any meaningful reliability properties if senders become faulty: these protocols are not consistent broadcast, let alone reliable broadcast. One should be aware, however, that the problem of broadcast in the presence of Byzantine non-source nodes is still highly non-trivial, as Byzantine non-source nodes can disseminate fake messages and lie to the network.

Byzantine fault detection. Our protocol developed new techniques of Byzantine fault detection. In contrast to crash failures, Byzantine failures are *not* context-free, and therefore it is impossible to define a general failure detector in Byzantine environments, independently of the algorithm itself [15].

Almost all protocols for Byzantine fault detection (and fault diagnosis) [1], [18], [29]–[32], [37] use the idea of collecting a *proof of misbehavior* by executing modified Byzantine resilient protocols. However, the approach requires a (large) number of rounds and a huge volume of exchanged messages to collect the necessary information to provide such a proof. An adversary can easily render the system even less practical by intermittently following and violating the protocol specification. Similarly, PeerReview [17] can detect and deter failures by exploiting accountability. It also uses a "sufficient" number of witnesses to discover faulty replicas. An exception is BChain [16] which does not need to regularly collect evidence. However, BChain is an atomic broadcast protocol for a complete graph. The technique developed in our protocol deviates significantly from that of BChain.

Self-stabilizing Byzantine broadcast. Self-stabilization [14] is a powerful approach to obtaining correct behavior regardless of the consistency of initial states. Specifically in sparse networks, Maurer and Tixeuil [25] combine Byzantine tolerance and self-stabilization to deal with both a fraction of permanent Byzantine failures and an arbitrary number of transient Byzantine failures. This generalizes the traditional Byzantine broadcast. As our experimental evaluation shows, if the senders are Byzantine, the time needed to recover from transient failures may be prohibitively long. Our protocol, instead, can be well applied to this scenario, yielding a more robust and efficient protocol.

III. Preliminaries

We represent the network using an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Let D be the network diameter (*i.e.,* the maximal distance between any two nodes). Let Δ be the maximum degree of G. We begin by describing several definitions.

Definition 1. (Path). A sequence of nodes $X = (p_1, p_2, \cdots, p_n)$ is a path if $\forall i \in \{1, 2, \cdots, n-1\}$, p_i and p_{i+1} are neighbors.

Definition 2. (Same source disjoint paths). Two paths $X = (p_1, p_2, \cdots, p_{n_1})$ and $X' = (q_1, q_2, \cdots, q_{n_2})$ are same source disjoint if $p_1 = q_1$ and $(X \backslash p_1) \cap (X' \backslash q_1) = \emptyset$; we write $\mathrm{DP}(p_1, X, X') = 1$ to denote two paths X and X' with the same source p_1. Generally, for n paths $X_1, X_2, \cdots, X_n$, if

they share the same source node p and every two paths are same source disjoint, we write $\text{DP}(p, X_1, \cdots, X_n) = 1$.

Definition 3. (Distance). The distance between two nodes p_i and p_j, denoted $distance(p_i, p_j)$, is the smallest number of edges between them.

The following definition is novel and vital to this work.

Definition 4. (Neighbor zone) Given an integer Z, the neighbor zone of a node p, denoted as $\text{NZ}(p)$, is a set of connected nodes $\{p_1, \cdots, p_n\}$, where $p \in \text{NZ}(p)$ and $\forall i, j \in \{1, \cdots, n\}$ and $i \neq j$, $distance(p_i, p_j) \leq 2Z$.

We consider the problem of reliable broadcast in a sparse network that can be decomposed into cycles. We borrow the following definitions from MT [25].

Definition 5. (Cycle). A set of nodes is a cycle if there exists a path $X = (p_1, p_2, \cdots, p_n)$ that contains all the nodes in the set and p_1 and p_n are neighbors. The *diameter* of a cycle is $n/2$ if n is even, and $(n-1)/2$ if n is odd.

Definition 6. (Connected set of cycles). A set of cycles S is connected if, $\forall\{C, C'\} \subseteq S$, there exists a sequence of cycles $(C_1, \cdots, C_n)$ such that $C = C_1, C' = C_n$ and, $\forall i \in \{1, \cdots, n-1\}$, C_i and C_{i+1} have at least two nodes in common.

Definition 7. (Resilient decomposition). An arbitrary set of cycles S of the network is a resilient decomposition if, for each pair of nodes p and q, there exists a connected set $S(p, q) \subseteq S$ of at most Δ cycles such that 1) Each cycle contains p and not q; 2) Each neighbor of p (distinct from q) belongs to a cycle of $S(p, q)$.

Definition 8. (Z-resilient network). A network is Z-resilient if there exists an arbitrary set of cycles S of the network such that S is a resilient decomposition, and the diameter of the cycles of S is at most Z.

Definition 9. ((p, q)-valid set of sequences). Let Ω be a set of sequences, where a sequence $(p_1, \cdots, p_n)$ is a valid path and $n \leq 4D\Delta^2 Z$. Let $G(\Omega)$ represent a subgraph (V, E) of G such that: 1) V is the set of node identifiers in Ω; 2) For any sequence $(p_1, \cdots, p_n)$ and $i \in \{1, \cdots n\}$, there exists an edge in G such that $p = p_i$ and $q = p_{i+1}$. We say that Ω is (p, q)-valid if: 1) $\forall(p_1, \cdots, p_n) \in \Omega, p_1 = p$ and $p_n = q$; 2) The graph $G(\Omega)$ can be decomposed into a connected set of cycles $(C_1, \cdots, C_m)$ such that $\forall i \in \{1, \cdots, m\}$, C_i and C_{i+1} has at least two nodes in common.

IV. Our Protocol

System model. We assume a Z-resilient network where the minimum distance between any two Byzantine nodes is strictly greater than $2Z$. A key property is that in *any* neighbor zone of any sender, there is at most one faulty node. If, for instance, the sender is faulty, all the other nodes in its neighbor zone must be correct. We use both MACs and signatures, but signatures are only needed in case of failures. Let $\langle M \rangle_i$ denote an authenticated message for M using signatures, signed by a node p_i. Let $[M]$ denote an authenticated message for M using MACs.

Property 1. *Validity:* If a correct node broadcasts a message m, then every correct node eventually delivers m.
Property 2. *No Duplication:* No message is delivered more than once.
Property 3. *No Creation:* If a correct node delivers a message m with sender p_s, then m was previously broadcast by p_s.
Property 4. *Agreement:* If a message m is delivered by some correct nodes, then m is eventually delivered by every correct node.

Fig. 1: Byzantine reliable broadcast specification.

Definition of Byzantine reliable broadcast. A Byzantine reliable broadcast algorithm ensures that the correct nodes agree on the set of messages, even when the senders of these messages behave arbitrarily. It is characterized by the four properties in Fig. 1 (see also [8]).

```
00   on receiving m_0 = [s, m, (p_1, p_2, ..., p_n)]
01     if n ≥ Z then discard m_0                                  ⇐ [MT1]
02     if p_n is a neighbor of p_i then add m_0 to p_i.Rec, multicast m_0
03-1   pred ← ∃X, X' s.t. DP(p_s, X, X') = 1 for m                ⇐ [MT1]
03-2   pred ← ∃(p_s, p_i)-valid set for m                         ⇐ [MT2]
04     if pred then accept m from p_s, remove (s, m', X') from p_i.Rec
```

Fig. 2: The MT algorithms.

The underlying MT algorithms. MT [25] presented two simple and elegant Byzantine broadcast algorithms for sparse networks—MT1 (a broadcast protocol) and MT2 (a self-stabilizing broadcast protocol), as depicted in Fig. 2 in pseudocode. Neither of them can tolerate faulty senders.

For both algorithms, a sender p_s multicasts a message m to all its neighbors which then multicast the message to their neighbors, and so forth. When multicasting a message, each node appends its identity to the message and the nodes form a *travel path* X.

For MT1, a node delivers a message, if it receives a matching message from two disjoint paths and the number of nodes in each path does not exceed Z. MT2 removes the limit of Z nodes identifiers. A node q waits until it receives a same message from p_s with several different paths. If the fusion of the paths form a (p_s, q)-valid set of sequences, q accepts the message from p. As a self-stabilizing broadcast protocol, MT2 does not terminate: A node q that have already accepted any message from a node p_s, can accept another message from p_s, and so forth.

Overview. Our protocol trades network environments for stronger reliability, while preserving high throughput for both fault-free and failure scenarios. To handle sender equivocation, we combine fault detection and fault tolerance techniques. If a sender fails to send any message to some nodes in time,

it is convenient to regard that the sender sends an "empty" message ϵ that is different from any messages in the usual message space $\mathcal{M}$, i.e., we consider an extended message space $\mathcal{M} \cup \{\epsilon\}$. We divide sender equivocation into three types such that no matter what the graph connectivity is, any equivocation behavior falls into one of them:

Type I: The source node sends at least three different messages to three neighbors.
Type II: The source node sends only two different messages and each such message reaches at least two neighbors.
Type III: The source node sends only two different messages and one of the messages reaches only one neighbor.

10 *cond*:{$\exists(p_i,p_j,p_k) \in \{X_i,X_j,X_k\}$ & DP$(p_s,X_i,X_j,X_k)=1$ & NZ(p_s,p_i,p_j,p_k)&$M.p_i \neq M.p_j \neq M.p_k$} **or** {$\exists(p_i,p_j,p_k,p_l) \in \{X_i,X_j,X_k,X_l\}$ & DP$(p_s,X_i,X_j,X_k,X_l)=1$ & NZ(p_s,p_i,p_j,p_k,p_l) &$M.p_i = M.p_j = m$ & $M.p_k \neq m$ & $M.p_l \neq m$}
20 **on receiving** $[\text{MSG}, p_s, m, X]$
21 **add** $[\text{MSG}, p_s, m, X]$ to $p_i.Rec$
22 **if** $p_i = \tilde{p}_s$&m is new **or** $p_i \neq \tilde{p}_s$ **then**
23 forward $[\text{MSG}, p_s, m, X]$ to neighbors
24 **if** m is new **then** start timer T_2
25 **if** m is new & $p_i = \tilde{p}_s$ **then** start timer T_1
26 **if** $X = p_s$ & $run(T_1)$ **then** cancel timer T_1
27 **if** $\exists(X_i,X_j) \in p_i.Rec$ & DP(p_s,X_i,X_j) & $M.X_i \neq M.X_j$ **then**
28 **send** $\langle \text{ALERT}, p_s, p_i, p_i.Rec\rangle_i$
30 **on receiving** $\langle \text{ACCUSE}, p_s, p_j, p_j.Rec\rangle_j$
31 **add** $\langle \text{ACCUSE}, p_s, p_j, p_j.Rec\rangle_j$ to $p_i.Acc(p_s)$
32 **if** $\exists(p_j,p_k) \in p_i.Acc(p_s)$ *s.t.* NZ(p_s,p_i,p_j) **then**
33 cancel timers, block p_s
34 **forward** $\langle \text{ACCUSE}, p_s, p_j, p_j.Rec\rangle_j$ and $\langle \text{ACCUSE}, p_s, p_k, p_k.Rec\rangle_k$
35 **if** *cond* **then send** $\langle \text{ACCUSE}, p_s, p_i, p_i.Rec\rangle_i$, cancel timers
40 **on receiving** $\langle \text{ALERT}, p_s, p_i, p_j.Rec\rangle_j$
41 **add** $\langle \text{ALERT}, p_s, p_i, p_j.Rec\rangle_j$ to $p_i.Ale$
42 **if** *cond* **then**
43 **send** $\langle \text{ACCUSE}, p_s, p_i, p_i.Rec\rangle_i$, cancel timers
44 remove $\langle \text{ALERT}\rangle$ from $p_i.Ale$ and $\langle \text{ACCUSE}\rangle$ from $p_i.Acc$
50 **on** timeout T_1 & $p_i = \tilde{p}_s$
51 add $[\text{MSG}, p_s, \epsilon, p_s]$ to $p_i.Rec$, **send** $\langle \text{ALERT}, p_s, p_i, p_i.Rec\rangle_i$
60 **on** timeout T_2
61 **if** $pred$ **then** deliver m, remove $[\text{MSG}, p_s, m, X]$ from $p_i.Rec$

Fig. 3: Our protocol.

Our protocol can be based on either MT1 or MT2, leading to a Byzantine reliable broadcast protocol or self-stabilizing broadcast protocol. Under normal circumstances, nodes run MT1 or MT2 with MACs based authenticated messages [MSG]. We use two types of signed messages—$\langle$ALERT$\rangle$ and $\langle$ACCUSE$\rangle$ to cope with failures. An $\langle$ALERT$\rangle$ message is triggered if a node receives mismatching messages from disjoint paths. However, a faulty node might issue an $\langle$ALERT$\rangle$ to frame the sender. A correct node thus needs to rely on $\langle$ALERT$\rangle$ messages *in the same neighbor zone* to rule out this possibility. After a node is certain that the source node is faulty, it generates an $\langle$ACCUSE$\rangle$ message. A node discards all the messages related to p_s if it already generates an $\langle$ACCUSE$\rangle$ message or receives two $\langle$ACCUSE$\rangle$ messages from nodes in the same neighbor zone.

Our protocol. The protocol is depicted in Fig. 3. Security of our protocol is *multi-tiered*: when the networks are synchronous, it satisfies the definitions in Fig. 1; when the networks are asynchronous but senders are correct, our protocol meets all the definitions except validity, in which case correct nodes still always agree on the same set of messages.

Let X be the travel path of the message, $M.p_i$ denote the message that p_i receives from a source node p_s, $M.X$ be the message p_i receives from a path X, $M.X_i = M.X_j$ be the case where the messages from path X_i and X_j are matching, $run(T_i)$ represent if the timer T_i has been started, and $\tilde{p}_s$ be a neighbor of p_s. Each node stores three sets of messages, including $p_i.Rec$ for the [MSG], $p_i.Ale$ for the $\langle$ALERT$\rangle$ messages, and $p_i.Acc$ for the $\langle$ACCUSE$\rangle$ messages.

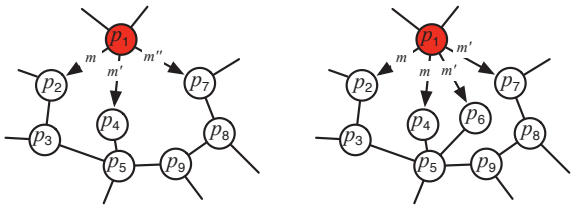
The algorithm proceeds as follows. A sender p_s multicasts a message $[\text{MSG}, p_s, m, X]$ to its neighbors. When a node p_i receives the message, it appends its id p_i to X and forwards the message to its neighbors, as shown in lines 22-23. Note that if a neighbor p_i receives a message from a source node p_s, it forwards the message only if the message is new. After receiving a [MSG], p_i may start two fixed timers T_1 and T_2. T_1 is used for the neighbors of p_s to monitor if they have received the [MSG] from p_s. The timer will be canceled if the neighbor of p_s receives the [MSG] from p_s. Instead, T_2 is a timer used for any nodes to see if a desired condition $pred$ can be met before the timer expires (see lines 60-61). Recall that we used the same notation $pred$ when we describe MT1 and MT2 in Fig. 2. If our protocol is instantiated using, say, MT1, then $pred$ is defined as in Fig. 2, *i.e.,* $pred$ returns 1 if a node receives matching messages from two disjoint paths.

Line 10 specifies a condition, which is used by nodes to verify if the source node is faulty based on their own $\langle$ALERT$\rangle$ and $\langle$ACCUSE$\rangle$ sets. The first and second OR clauses match Type I and Type II equivocation, respectively. The first clause aims to check if there exists at least three nodes in some NZ(p_s) such that their $\langle$ALERT$\rangle$ and $\langle$ACCUSE$\rangle$ sets contain three inconsistent messages. Note that there exists at most one faulty node in any neighbor zone of p_s. If p_s sent consistent messages to all its neighbors, it would be impossible that three of its neighbors (two of which must be correct) claim they received inconsistent messages. The second clause checks if there exists at least four nodes in some NZ(p_s) such that two of them received m while another two of them received a different message m'. Similarly, if the source node is correct, the condition will not be satisfied.

We do not need to worry about the Type III sender failures, because this type of failures are effectively masked by our protocol (an example coming shortly.)

When $cond$ is satisfied, a node sends a message $\langle \text{ACCUSE}, p_s, p_i, p_i.Rec\rangle_i$ to all the neighbors. When a node receives two $\langle$ACCUSE$\rangle$ messages from two nodes in the same neighbor zone, it can also confirm that the source node is faulty. From then on, it ignores any messages which are from

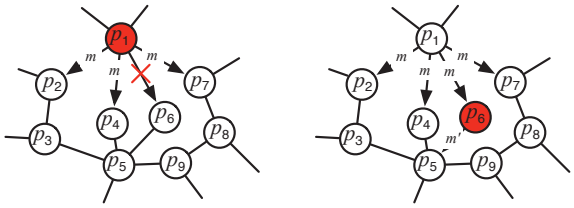
p_s and ⟨ALERT⟩ and ⟨ACCUSE⟩ messages related to p_s, and discards the corresponding ⟨ALERT⟩ and ⟨ACCUSE⟩ sets.



(a) An example for Type I equivocation. (b) An example for Type II equivocation.

Fig. 4: Examples for Type I and Type II failures.

Examples. Consider an example in Fig. 4 and suppose that all the nodes are in NZ(p_1). Fig. 4a illustrates Type I equivocation, where p_1 sends m to p_2, m' to p_4, and m'' to p_7. All the nodes will send an ⟨ALERT⟩ message, because they will all receive inconsistent messages from disjoint paths, and all the nodes will receive ⟨ALERT⟩ messages satisfying the first clause in *cond*. Thus, all the correct nodes can confirm that p_1 is faulty and generate an ⟨ACCUSE⟩ message. Fig. 4b illustrates Type II equivocation, where p_1 sends m to two of its neighbors and m' to two other neighbors. Likewise, all the nodes will receive conflicting messages and generate an ⟨ALERT⟩ message. For instance, p_2 receives m from $X = \{p_1\}$ and m' from the path $X = \{p_1, p_6, p_5, p_3\}$. After the nodes receive the ⟨ALERT⟩ messages, they can all verify *cond* and generate ⟨ACCUSE⟩ messages.



(a) The source p_1 is faulty. (b) Node p_6 is faulty.

Fig. 5: Example for a Type III failure.

We point out that it is "impossible" to detect a Type III faulty sender, if we consider the case where source nodes multicast messages without using signatures. The reason is that Type III is indistinguishable from the case with a faulty non-source node. Let's consider an example in Fig. 5. In Fig. 5a, we assume that the source node p_1 is faulty. It sends m to nodes p_2, p_4, and p_7, and sends m' to p_6. In Fig. 5b, we assume that p_1 is correct and but one of its neighbors p_6 is faulty. If p_6 changes m to m' and sends m' to other nodes, then no one can discover if conflicting messages are due to p_1 or p_6. Fortunately, this type of failures can be effectively tolerated by MT protocols and also our protocol.

V. CORRECTNESS PROOF

We begin by recalling several lemmas in MT. Lemma V.1 and Lemma V.2 are the properties of the Z-resilient networks, while Lemma V.3 are the properties of MT1 and MT2.

Lemma V.1. *Let p_s be a Byzantine node. Let p_i and p_j be two neighbors of p_s. There exists a correct path of at most $\alpha = \Delta Z$ hops connecting p_i and p_j.*

Lemma V.2. *Let p_i and p_j be two correct nodes. There exists a correct path of at most $\beta = 2D\Delta$ hops connecting p_i and p_j.*

Lemma V.3. *Let p_i and p_j be two non-neighbors correct nodes. Node p_j accepts message m from p_i within at most γ time and never accepts another message from p_i. For MT1, $\gamma = 8D\Delta^2 ZT$ and for MT2, $\gamma = 12D\Delta^2 ZT$, where T is the upper bound on the channel transmission time.*

Our protocol can be based on either MT1 or MT2. We focus on the case of MT1 and the other case is similar. The validity, no duplication, and no creation properties essentially follow from MT. The crux is to prove the correctness of the agreement property.

Theorem V.4. *(Agreement) If a message m is delivered by some correct nodes, then m is eventually delivered by every correct node.*

Proof. MT shows that agreement is satisfied if the sender is correct. We demonstrate that any sender equivocation behavior can be either *eventually* and *accurately* detected by all the correct nodes or tolerated by our protocol.

To this end, we prove the following claims in the rest of the section: 1) Type I and Type II failures can be eventually identified by all the nodes; 2) Correct nodes never accept any messages from senders who exhibit Type I or Type II failures; 3) If the sender is correct, it will never be accused by correct nodes; and 4) Type III failures do not introduce any inconsistency. The agreement property of our protocol will then easily follow from the above claims and the agreement property of MT. ■

Below we prove the four claims for Theorem V.4.

Theorem V.5. *Type I and Type II Byzantine senders can be always detected by at least two correct nodes in some neighbor zone. After at most $(3\alpha+\beta)T+T_1$ time, all the correct nodes learn that the sender p_s is faulty.*

Proof. We first prove in Lemma V.6 that Type I and Type II failures can be effectively detected by nodes in some neighbor zone. Then we show that eventually all the correct nodes learn the fact and we upper bound the time in Lemma V.7.

Lemma V.6. *Type I and Type II Byzantine senders can be always detected by at least two correct nodes in some neighbor zone NZ(p_s). Nodes in the network will all receive the ⟨ACCUSE⟩ messages.*

Proof. We first show that in the presence of Type I and Type II Byzantine senders, all the correct nodes in NZ(p_s) will generate ⟨ALERT⟩ messages and we distinguish two cases:

1) Assume that none of messages which the faulty source node sends is an empty message. We claim that correct nodes will receive conflicting messages from disjoint paths in NZ(p_s). Indeed, according to Lemma V.1, there exists a correct path

between any two correct nodes. Therefore, correct nodes will then generate ⟨ALERT⟩ messages.

2) Assume otherwise there exists at least one correct node (say, p_i) which the sender sent an empty message. Recall that if p_i learns a non-empty message from any other node in NZ(p_s), it starts a timer T_1. If p_1 does not receive any message from p_s before the timer expires, it sends an ⟨ALERT⟩ message. This additional step allows p_i to know if p_s sent some message to other neighbors but did not send any message to it. (The rest of the scenario is now the same as the above one.)

We now show that correct nodes in NZ(p_s) will generate ⟨ACCUSE⟩ messages. Note that ⟨ALERT⟩ messages contain the [MSG] received from p_s and their travel paths. For Type I senders, there exist three correct nodes whose ⟨ALERT⟩ messages contain three inconsistent messages from three disjoint paths in NZ(p_s). This will satisfy the first clause of $cond$ and these nodes will generate ⟨ACCUSE⟩ messages. Likewise, for Type II senders, there exist at least two correct nodes which generate ⟨ACCUSE⟩ messages. Since there exists a correct path between any two correct nodes, it is easy to see that all correct nodes in the network will receive ⟨ACCUSE⟩ messages. □

Lemma V.7. *All the correct nodes learn that p_s is faulty after at most $(3\alpha + \beta)T + T_1$ time.*

Proof. We consider the worst case where some nodes in NZ(p_s) need to wait for the timer T_1 to verify if the source node sends an empty message. In this case, a node p_i in NZ(p_s) needs at most ①αT time to learn that another node p_j in NZ(p_s) receive at least a message, say m (according to Lemma V.1), and then waits for ②T_1 time before p_i send an ⟨ALERT⟩ message. For those nodes in NZ(p_s) that have already received m, it takes another ③αT time for them to receive the ⟨ALERT⟩ message and generate their ⟨ALERT⟩ messages. After the nodes in NZ(p_s) generate ⟨ALERT⟩ messages, it takes ④αT time for the nodes to receive the ⟨ALERT⟩ messages and generate ⟨ACCUSE⟩ messages. For the rest of nodes in the graph, according to Lemma V.2, there exists a correct path of at most β hops connecting any two correct nodes, so it takes at most ⑤βT time for the rest of nodes to learn that p_s is faulty after receiving two ⟨ACCUSE⟩ messages. Notice that since T represents the transmission time in the network, T_1 can be set to T for a neighbor node to detect the failure of the source. Summing up ① to ⑤, the maximum time is $(3\alpha + \beta)T + T_1 = (3\alpha + \beta + 1)T$. As shown in Lemma V.3, nodes will not accept any other messages after γ time. Therefore, the timer T_2 can be set to be *max*(γ, $(3\alpha + \beta + 1)T$). This guarantees that before T_2 times out, all the correct nodes can receive ⟨ACCUSE⟩ messages. □

The theorem now follows. ■

Theorem V.8. *Correct nodes never accept any messages from senders who exhibit Type I or Type II failures.*

Proof. Setting the timer T_2 as the upper bound in the above lemma, the theorem trivially follows, as our protocol requires each node to wait for T_2 time to deliver the messages. ■

Theorem V.9. *If the source p_s is correct, it will never be accused by correct nodes.*

Proof. Suppose a correct node accused a source node p_s. According to our protocol, this node either received messages that satisfy $cond$ (type I) or received two ⟨ACCUSE⟩ messages from nodes in the same neighbor zone (type II). For type II, since there is one faulty in NZ(p_s), one of the two nodes that sent ⟨ACCUSE⟩ messages must be correct. This indicates that the correct node either received messages that satisfy $cond$, or received two ⟨ACCUSE⟩ messages from some other nodes in the zone. Inductively, we can prove that for type II, there exists some correct node that received messages that satisfy $cond$. Therefore, in both cases, there exists some correct node that received messages that satisfy $cond$ and for this reason it sent an ⟨ACCUSE⟩ message.

For this correct node, if the first clause of $cond$ was satisfied, there exist three nodes whose ⟨ALERT⟩ messages contain three inconsistent messages from three disjoint paths in NZ(p_s). In this case, either p_s sent three inconsistent messages or at least two of them are faulty. As there is at most one faulty node in each NZ(p_s), we know that p_s is faulty. On the other hand, if the second clause of $cond$ was satisfeid, either p_s sent two messages and each of them reached at least two nodes or at least two of them are faulty. Likewise, we can conclude that p_s is faulty. The theorem now follows. ■

Theorem V.10. *Type III failures do not introduce inconsistency.*

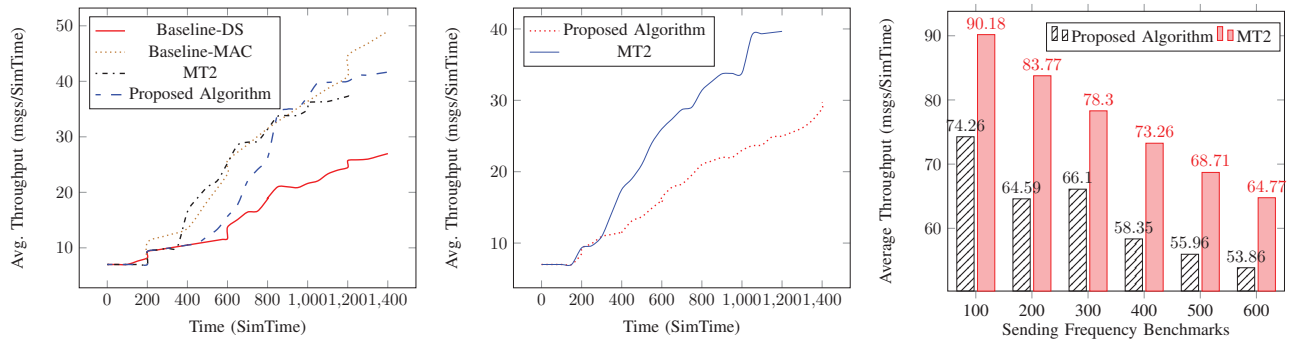
Proof. If p_s only sends a message m' to one neighbor and m to all other neighbors, a correct node will not receive a message from two disjoint paths. This is because there is at most one correct node in NZ(p_s) and the message is sent to at least two neighbors so that a node can accept m'. ■

VI. Implementation and Evaluation

We used the OMNeT++ Discrete Event Simulator [36] to model our algorithm. Each simulation begins with an initialization stage, during which the nodes send initialization messages in order to set up the network topology and to detect their neighbors.

We implemented the following algorithms and compared them with our algorithm. 1) Baseline-MAC: A MAC-based multicast protocol where nodes use MACs for message authentication in authenticated channels; 2) Baseline-DS: A digital signature-based multicast protocol, where nodes use digital signatures for message authentication; 3) MT algorithms.

For Baseline-MAC and Baseline-DS, we implemented the conventional gossip algorithm. Namely, both protocols are based on a multicast protocol, where a source node will send a message to each of its neighbors, which in turn will forward that message to each of their neighbors until all nodes have received the message. Baseline-MAC is efficient but does not provide meaningful fault tolerant guarantees. Instead, Baseline-DS is more robust and can detect sender equivocation, but still it fails to handle cases such as sender crashes.



(a) Faulty source node detection overhead. (b) Throughput for faulty non-source nodes. (c) Throughput of various benchmarks.

Fig. 6: Protocol evaluation.

We utilized HMAC [3] and RSA-FDH [4] to implement the underlying MAC and digital signature, respectively.

We implemented our protocol on top of MT1 and compare with MT2 in failure scenarios. Indeed, both our protocol and MT2 can deal with faulty senders, though via very different perspectives and with different properties. We implemented a failure injection mechanism where a number of random nodes are selected during each simulation. The faulty nodes can be further specified as either faulty source nodes or faulty non-source nodes or both. If a node is the source node, it simply equivocates to the neighbors by generating inconsistent messages with the same timestamp. If a node is a non-source node, it tampers with or falsifies the content of the messages and forwards to the neighbors.

We generate a random cyclic topology with a minimum of 3 nodes in each cycle. We use several benchmarks to evaluate a cyclic network with different traffic. For the x benchmark, each node multicasts a message to its neighbors every x ms. We evaluate throughput as the number of messages per simulation time (SimTime), and delivery rate as the percentage of messages received versus messages sent. Messages sent by the source node to the network are considered meaningful messages. All other invalid messages that correct nodes will not deliver are considered meaningless.

Faulty source node detection overhead. We compare failure-free cases for the Baseline-MAC and Baseline-DS algorithms with the faulty source node case for our algorithm. We compare Baseline-MAC and our algorithm to evaluate the overhead for our algorithm. Although Baseline-MAC does not handle any failures, we can compare Baseline-MAC in the failure-free case and our algorithm in the failure case to evaluate the overhead of our faulty source node detection algorithm. We then compare Baseline-DS and our algorithm to show our algorithm's efficiency in utilizing digital signatures, where in Baseline-DS, nodes are able to detect a faulty source node due to receiving mismatched messages.

We evaluate the three algorithms using a 200 benchmark and a single faulty source node. The benchmark determines the frequency that a faulty node sends false messages. Fig. 6a shows that our algorithm has a consistently higher throughput than Baseline-DS. This is caused by the fact that digital signatures are more expensive than MACs and our algorithm uses digital signatures when failures occur. The introduction of ⟨ALERT⟩ and ⟨ACCUSE⟩ messages in the presence of the faulty node also introduces more network traffic. Our algorithm achieves similar throughput, however, with Baseline-MAC. This shows that in the case where there are fewer failures, our algorithm generates low overhead for failure detection.

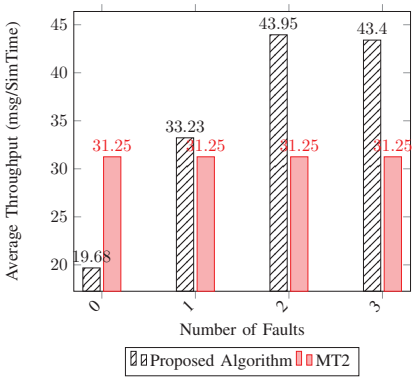


Fig. 7: Average throughput vs. number of faults.

Throughput. We compare MT and our algorithm in both the case where source nodes are faulty and non-source nodes are faulty. We use 200 benchmark and one failure in the network. Fig. 6a shows the case where the source node is faulty. Our algorithm achieves lower throughput than MT in the beginning and higher throughput later in the experiment. This is due to the ⟨ALERT⟩ and ⟨ACCUSE⟩ messages that are introduced into the network. As the faulty nodes begin to send false messages with an increased frequency, more ⟨ALERT⟩ and ⟨ACCUSE⟩ messages are generated. The lower the frequency, the faster we can confirm that there is a failure which results in a lower throughput initially. We also compare the performance where non-source nodes are faulty. Notice that the non-source nodes can "frame" correct source nodes by altering the content of once correct messages. It can be observed in Fig. 6b that our algorithm achieves lower throughput in this situation.

Additionally, we evaluate the case with various benchmarks that represent faulty node sending frequency. We randomly select 3 faulty nodes in a network of 12 nodes. As shown in Fig. 6c, the average throughput of our algorithm in each

benchmark is lower than MT. This is due to there being more false messages in the network as the frequency increases. Our algorithm detects the faulty node and alerts other nodes, which in turn do not forward any of the false messages. This results in a lower average throughput. We later test the case where nodes behave as faulty source nodes and multicast false messages. As shown in Fig. 7, our algorithm has a higher throughput in the presence of failures. The throughput is lower when there are no failures because our algorithm does not repetitively send the same message. We observe that our algorithm allows nodes perform more efficiently after a failure has been detected.

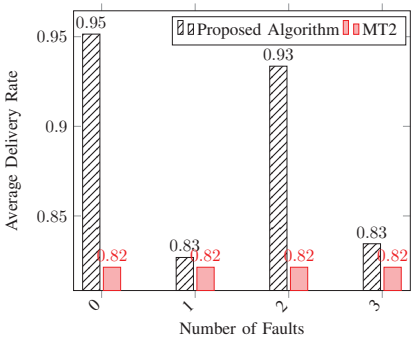


Fig. 8: Delivery rate vs. number of faults.

Delivery rate. As shown in Fig. 8, our algorithm consistently achieves a higher delivery rate than MT. This is because any decrease in delivery rate caused by a faulty node is balanced by the introduction of ⟨ALERT⟩ and ⟨ACCUSE⟩ messages.

VII. Conclusion

We presented the first Byzantine reliable broadcast protocol for sparse networks that can tolerate Byzantine senders in synchronous environments. We developed new techniques for fault detection. Our protocol is efficient for both fault-free and failure scenarios; in particular within gracious executions no public key cryptographic operations are needed. Finally, we implemented and evaluated our protocol. Our experimental evaluation shows that our protocol has high throughput and high failure resilience.

VIII. Acknowledgments

Sisi Duan was supported in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the Department of Energy. Lucas Nicely was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship program. Haibin was supported in part by NSF grant CNS-1413996 for the MACS project.

References

[1] J. Adams and K. Ramarao. Distributed diagnosis of Byzantine processors and links. *ICDCS*, pp. 562–569, 1989.
[2] P-L. Aublin, R. Guerraoui, N. Knezevic, V. Quema, and M. Vukolic. The next 700 BFT protocols. *TOCS*, vol. 32, issue 4, 2015.
[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *CRYPTO 1996.*
[4] M. Bellare and P. Rogaway. The exact security of digital signatures — How to sign with RSA and Rabin. *EUROCRYPT 1996*, pp. 399–416.
[5] V. Bhandari and N. Vaidya. On reliable broadcast in a radio network. *PODC*, pp. 138–147, 2005.
[6] G. Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation 75*, pp. 130–143, 1987.
[7] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM* 32(4), 824–840, 1985.
[8] C. Cachin, R. Guerraoui, and L. Rodrigues. Introduction to reliable and secure distributed programming. Springer, 2011.
[9] M. Castro and B. Liskov. Practical Byzantine fault tolerance. *ACM Trans. Comput. Syst*, 20(4): 398–461, 2002.
[10] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2):225–267, 1996.
[11] S. Chaudhuri. Agreement is harder than consensus: set consensus problems in totally asynchronous systems. *PODC*, pp. 311–324, 1990.
[12] S. Chaudhuri, M. Herlihy, N. A. Lynch, and M. R. Tuttle. A tight lower bound for k-set agreement. *FOCS*, 1993.
[13] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.
[14] S. Dolev. Self-Stabilization. MIT Press, 2000.
[15] A. Doudou, B. Garbinato, R. Guerraoui, and A. Schiper. Muteness failure detectors: Specification and implementation. *EDCC*, 1999.
[16] S. Duan, H. Meling, S. Peisert, and H. Zhang. BChain: Byzantine replication with high throughput and embedded reconfiguration. *OPODIS 2014.*
[17] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: practical accountability for distributed systems. *SOSP*, pp. 175–188, ACM, 2007.
[18] H. Hsiao, Y. Chin, and W. Yang. Reaching fault diagnosis agreement under a hybrid fault model. *IEEE Trans. on Computers*, vol. 49, no. 9, 2000.
[19] M. Humphries and K. Gurney. Network 'small-world-ness': A quantitative method for determining canonical network equivalence. *PLoS One* 3(4): e2051, 2008.
[20] C.-Y. Koo. Broadcast in radio networks tolerating Byzantine adversarial behavior. *PODC*, pp. 275–282, ACM, 2004.
[21] K. Kursawe and V. Shoup. Optimistic asynchronous atomic broadcast. *ICALP 2005*, pp. 204–215, 2005.
[22] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
[23] A. Maurer and S. Tixeuil. Limiting Byzantine influence in multihop asynchronous networks. *ICDCS*, pp. 183–192, 2012.
[24] A. Maurer and S. Tixeuil. On Byzantine broadcast in loosely connected networks. In *DISC*, pp. 253–266, 2012.
[25] A. Maurer and S. Tixeuil. Self-stabilizing Byzantine broadcast. *SRDS*, 2014.
[26] M. Nesterenko and S. Tixeuil. Discovering network topology in the presence of Byzantine nodes. *IEEE TPDS*, 20(12):1777–1789, 2009.
[27] M. Newman. The structure and function of complex networks. *SIAM Review* 45, pp. 67–256, 2003.
[28] A. Pelc and D. Peleg. Broadcasting with locally bounded Byzantine faults. *Inf. Process. Lett.*, 93(3):109–115, 2005.
[29] F. Preperata, G. Metze, and R. Chien. On the connection asssignment problem of diagnosable systems. *IEEE Trans. on Elec. Comp.*, 16(6): 848–854, 1967.
[30] K. Ramarao and J. Adams. On the diagnosis of Byzantine faults. *Proc. Symp. Reliable Distributed Systems*, pp. 144–153, 1988.
[31] M. Serafini, A. Bondavalli, and N. Suri. Online diagnosis and recovery: on the choice and impact of tuning parameters. *IEEE TDSC*, 4(4): 295–312, 2007.
[32] K. Shin and P. Ramanathan. Diagnosis of processors with Byzantine faults in a distributed computing system. *Proc. Symp. Fault-Tolerant Computing*, pp. 55–60, July 1987.
[33] G. J. Simmons. A survey of information authentication. *Contemporary Cryptology, The Science of Information Integrity*, IEEE Press, 1999.
[34] S. Toueg. Randomized Byzantine agreements. *PODC 1984*, 1984.
[35] R. van Renesse, C. Ho, and N. Schiper. Byzantine chain replication. *OPODIS 2012.*
[36] A. Varge. OMNeT++. In *Modeling and Tools for Network Simulation*, 91–104, 2010.
[37] C. Walter, P. Lincoln, and N. Suri. Formally verified on-line diagnosis. *IEEE Trans. Software Eng*, 23(11): 684–721, 1997.

Evaluating Reliability Techniques in the Master-Worker Paradigm

Evgenia Christoforou*†, Antonio Fernández Anta*, Kishori M. Konwar‡, Nicolas Nicolaou*
† Universidad Carlos III, Madrid, Spain. evgenia.christoforou@imdea.org
* IMDEA Networks Institute, Madrid, Spain. antonio.fernandez@imdea.org, nicolas.nicolaou@imdea.org
‡ MIT, Cambridge MA, USA. kishori@csail.mit.edu

***Abstract*—A distributed system is considered that carries out computational tasks according to the master-worker paradigm. A master has a set of computational tasks to resolve. She assigns each task to a set of workers over the Internet, instead of computing the task locally. For each task each worker reply to the master with the task result. Since the task was not computed locally, the master can not trust the result for two main reasons: (i) workers might deliberately provide an incorrect result, (ii) the result is corrupted due to some hardware or software failure during the execution of the task. Given the above, we can model our workers as either "altruistic", always willing to provide the correct result to each task, or "troll" that are trying to provide an incorrect result to each task. Moreover we model the failure of the worker to comply with her intended behavior, as an error probability ϵ. The goal of the master is to compute the correct result of all the tasks with high probability. In the literature two techniques have been used to achieve this goal: (i) "voting", that determines the correct result of a task given multiple replies of distinct workers; (ii)"challenges", that are tasks whose result is known and can be used to detect altruistic workers. What separates our work from the current literature is the realistic modelling of the worker's behavior and the fact that we do not restrict the task result to a binary set of answers; the domain of possible replies for a task can have multiple correct and multiple incorrect results. Given the above we evaluate the performance of the two techniques described in the literature in the scenario where $\epsilon = 0$ and when $\epsilon > 0$. Performance is measured in terms of: (1) time, i.e., the number of rounds performed by an algorithm for the computation of all the tasks, and (2) work, i.e., the number of total task computations performed by the workers. The case where $\epsilon = 0$ is used as a best case scenario that provides the optimal time and work bounds of the problem. In the case where $\epsilon > 0$ we propose two "natural" algorithms: one using a combination of both voting and challenges, and a second one using only voting. Both algorithms assume that certain system parameters are known. Since this might not always be the case we also provide an algorithm that estimates correctly these parameters with high probability.**

I. Introduction

Distributed computing systems following the master-worker paradigm are increasing in popularity over the past decades. In the literature, this paradigm is encountered under several names like: volunteer computing, desktop grid computing, public resource computing and so on. This work focuses on master-worker task computations, where a master entity has a set of tasks to be computed that is unable or unwilling to compute locally. Hence, she assigns these tasks over the Internet to worker entities willing to perform the task and reply back to the master with a result. The inherent limitation of this load distribution scheme is the unreliable nature of the workers. We assume that there is no mean of verifying an answer provided by a worker unless we know the set of solutions for the particular task.

One classical example of the master-worker paradigm is SETI@home [13] that uses the BOINC [1], [3] infrastructure. In the case of SETI@home, the project is collecting and analysing signals from space in a search for extraterrestrial forms of life. Volunteers willing to help the search register their machines, receive tasks to be executed (when available), and report the results back to the master.

Evidence exist though that volunteers might actually misreport values [1], [2], [10], [11]. The most straight forward explanation is that workers might have ulterior motives for misreporting a result. Another reason might be that actually a hardware or software failures happened during the computation of the task that was not detected.

Drawing from the above example and the work of Kondo et al. [11], where they have characterized errors in BOINC systems, we can infer the existence of two type of workers. **(i) Altruistic**[1] **workers:** This type of worker is positive towards executing the task, and willing to provide the correct result. In the case of a BOINC system, a "positive" worker will let its machine execute the task and report back the result. **(ii) Troll**[2] **workers:** This type of worker is negative towards executing a task and wants to convey an incorrect result to the master. Hence, it can miscalculate a task on purpose and tries to report an incorrect result. In this work we do not assume any type of intelligent strategy to fool the system, nor that a troll has any information on the actions taken by the master or the other workers.

We can safely assume that both types of workers can be susceptible to a small error probability ϵ that is related to

Supported in part by FP7-PEOPLE-2013-IEF grant ATOMICDFS No:629088, Ministerio de Economia y Competitividad grant TEC2014-55713-R, Regional Government of Madrid (CM) grant Cloud4BigData (S2013/ICE-2894, co-funded by FSE & FEDER), NSF of China grant 61520106005 and Spanish Ministry of Education grant FPU2013-03792 .

978-1-5090-3216-7/16/$31.00

[1]For historical reasons, to match with original work in the field of master-worker task computing.

[2]Historically they are called malicious workers, but since we are not assuming any intelligent behavior to harm the system here we call them troll workers.

hardware or software failures or to any other factor that would force them to deviate from their intended behavior. Thus an altruistic worker, will have a high probability of reporting a correct result, while a troll will have a low probability of reporting a correct result.

The goal of the master is to identify the correct result for each task assigned with high probability. As far as we know, two techniques are used (individually or combined) for increasing reliability. **(i) Voting:** The master uses redundancy by assigning the same task to multiple workers. After all the task replies are collected, the master uses a voting scheme [4], [15], [14] to decide on the correct result. This may fail to provide optimal reliability since a high concentration of incorrect results reported may lead to a decision on the task result that is incorrect. **(ii) Challenges:** The master uses a set of tasks with known solutions to identify the workers that are replying correctly. Again this approach can not guarantee optimal reliability if the same workers in different time intervals provides correct and incorrect results due to an error during the computation of the task (as we discussed above). Similar or identical concepts to this approach are encountered by the name of spot-checking, auditing or quiz [16], [18].

Both techniques add an extra load on the computation of the task. On the one hand, with voting the same task needs to be executed by multiple workers, thus the resources of the system are not used in an optimal way. On the other hand, challenges require that the workers compute tasks with known solutions, thus not only resources are not used in an optimal way but also the execution time of a task increases.

An added difficulty when using these techniques is related to the nature of the task. If you have tasks that can have multiple correct and multiple incorrect results, maybe you can not guarantee that the result of each task will be correct with high probability. The common assumption in the literature is that tasks have a binary set of solutions (i.e. only one correct and one incorrect solution are feasible).

Voting and challenges have been widely used either alone or in combination to provide the correct task result for each task with high probability. What is yet unclear is the advantage of one technique over the other or the combination of these techniques in terms of time and work complexity, given tasks with multiple correct and multiple incorrect results.

Related Work: In this section we review a few of the most representative works that use voting and challenges in the master-worker paradigm. In the work of Fernández et al. [9], [8] two voting mechanisms are presented under the assumption of a binary set of possible responses to the master. Targeting at a high reliability while minimizing redundancy, the authors provide analytical results assuming that the number of malicious workers or the probability of a worker acting maliciously is known. In the work of Konwar et al. [12] these last assumptions on having information on the malice are removed and new algorithms are proposed to approximate these probabilities. The lower bounds on the amount of work necessary, when the set of responses is binary, are comparable to the work complexity of our proposed algorithms. These works assume that all workers might fail in every round with a certain probability. In our work we take a more realistic assumption, considering that workers might reply incorrectly with a different probability related to their nature (that is, we assume two types of workers' failures).

In his work, Sarmenta [16] assumed the presence of malicious and altruistic workers, with only the malicious workers having a constant probability of submitting an erroneous result and binary set of response. Based on these assumptions he introduced sabotage-tolerance mechanisms that used voting and a spot checking technique (what we are calling challenges). Given the assumption that altruistic workers will always reply with the correct task result, the mechanisms combine voting with challenges to create a reputation for each worker and increase the systems' reliability while trying to keep redundancy low.

In their work, Zhao and Lo [18] compare voting to challenges (called Quiz) under two assumptions: that all malicious workers return the same incorrect result or that malicious workers return distinct results. They use as performance metrics the accuracy and overhead, and through simulations they show the trade-off among these performance metrics and the two reliability techniques. Their work is mostly experimental and does not provide any complexity analysis. Additionally they do not assume a density of solutions nor an error probability on the altruistic workers.

Finally, Sonnek et al. [17] design algorithms for efficient task allocation based on the reputation of each worker. Hence the reliability of each worker is a statistical property that depends on the results submitted by each worker. Each task is being redundantly assigned to a group of workers with a targeted reliability. These groups are formed based on different algorithms. Their algorithms are evaluated through simulations. In contrast, we provide analytical bounds on the performance metrics of our proposed algorithms.

Contributions: Our contributions can be summarized as follows:

- We model the master-worker paradigm in the presence of altruistic and troll workers using five parameters $\langle \epsilon, f_a, s, r, T \rangle$ (Section II): (i) ϵ is the probability that an altruistic worker may reply with an incorrect result or a troll worker with a correct result, (ii) f_a is the fraction of altruistic workers over the set of workers, (iii) s, r are the number of correct and incorrect answers for a task respectively, leading to the more realistic assumption that tasks are not only binary but rather may have solutions in a broader domain, and (iv) $T \subseteq \{C, V\}$ the set of reported result evaluation techniques, i.e., *challenges* and *voting*. Additionally we define two measures: (i) "time" and (ii) "work", for evaluating the complexity of the proposed algorithms.
- We fix $\epsilon = 0$ in Section III, i.e., the simple case where altruistic workers always reply with a correct result and troll workers always with an incorrect result. Given this idealistic scenario, we identify asymptotically optimal bounds on the time and work complexity when challenges and voting are

used separately. While it is clear that when challenges are used the master can receive the correct task result with probability one, this is not always the case when voting is used alone. In the case of voting, we were able to show a negative result giving conditions on the parameters of s, r, n_a and n_t, where the master will not always be deciding on the correct task result with probability one. This result reveals that the domain of reported results is important even in the simple case where $\epsilon = 0$.

- We then make the realistic assumption that $\epsilon > 0$ (Section IV) and we provide two algorithms MWMIX and MWVOTE that solve correctly all the tasks whp. Both algorithms assume that s, ϵ and f_a are known. Algorithm MWMIX uses both challenges and voting, and can be applied if $1 - \epsilon > \frac{s}{s+1}$. Algorithm MWVOTE uses only voting, and can be applied if $f_a(1 - \epsilon) + (1 - f_a)\epsilon > \frac{s}{s+1}$. What is interesting to observe is that these algorithms have a log factor overhead compared to the case where $\epsilon = 0$, which as shown in [12], is a necessary price to pay when voting is used.
- Finally, in the case where ϵ and f_a are not known, in Section V we provide algorithm E_1 that estimates these parameters within tight bounds *whp*.

II. Model

Our setting consists of a master process M and a set $W = \{w_1, \ldots, w_n\}$ of n worker processes. Workers might be unreliable and produce an incorrect result.

Definition 2.1 (Problem statement.): The master must guarantee with high probability (see below) the correct result for each task t_i where $t_i \in \mathcal{T} = \{t_1 \cdots t_n\}$, without computing the task locally.

To keep the pseudocode simple for the purpose of exposition we assume that $|\mathcal{T}| = n$. (The algorithms presented in this paper can be easily extended to run for $|\mathcal{T}| = O(poly(\log n))$ without violating the statements of correctness.)

Computation and Communication model: Processes in the system communicate by exchanging messages via reliable communication channels. Computation proceeds in synchronous rounds. For a process p a round consists of the following steps: (i) receive incoming messages, (ii) perform computation on the received messages and produce a set of outgoing messages, and (iii) send the produced messages. During a round each worker can compute only one task from the master, and report back the result of the task. A synchronous algorithm A is a collection of processes, and its state is defined over the vector of the states of each process in that collection. An execution ξ of A is an (infinite) sequence of states. We denote by $execs(A)$ the set of executions of A.

Performance Measures: An algorithm A is evaluated in terms of: (i) *time*, and (ii) *work*. Time is defined as the number of rounds that an algorithm A requires in order to determine the result of all n tasks in $\mathcal{T}$. Work represents the number of aggregated results computed by each worker in algorithm A. More formally, let $comp_\xi(w, t)$ be the number of times a worker w computes task t during an execution ξ of algorithm A. The task t can be one the tasks in $\mathcal{T}$ or a task chosen from a set $\mathcal{C}$ of challenge tasks available to the master M. Then,

$$work_\xi = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} comp_\xi(w_i, t_j) + \sum_{t \in \mathcal{C}} comp_\xi(w_i, t) \right)$$

is the work of all the workers in ξ. Thus, the work of an algorithm A is defined as $Work(A) = \max_{\xi \in execs(A)}(work_\xi)$ over all executions of A. The work captures the redundancy used by an algorithm (i.e., the number of workers that compute the same task), as well as the computation performed on challenges.

Density of Solutions: A reported result v for a task t may take values from a domain $D(t)$. Let $S(t) \subset D(t)$ be the set of *correct solutions* and $R(t) = D(t) \backslash S(t)$, the set of reported results that are incorrect solutions for task t. We only consider tasks t that have at least one correct solution, i.e., $|S(t)| \geq 1$.

Given $D(t)$ and $S(t)$ we define the *density of solutions* for task t as $\frac{|S(t)|}{|D(t)|}$. The density of solutions affects the techniques used to determine the correct task result. For simplicity of presentation, we will assume that the size of the domain $d = |D(t)|$, the number of correct solutions $s = |S(t)|$, and the number of incorrect solutions $r = |R(t)|$ are the same for every $t \in \mathcal{T}$. Hence, the density of solutions is the same for all tasks t.

Worker Types: We assume that each worker $w_i \in W$ is either *altruistic* or a *troll* and its type remains the same throughout the execution of the algorithm. Let $W_a \subseteq W$ be the set of altruistic and $W_t \subseteq W$ the set of trolls. We use $n_a = |W_a|$ and $n_t = |W_t|$ to denote the sizes of these sets. We assume that $n_a \geq 1$. Observe that $W = W_a \cup W_t$ and $n = n_a + n_t$. We also use $f_a = \frac{n_a}{n}$, to denote the fraction of altruistic workers. Altruistic workers are "positive" towards executing a task, always aiming at returning a correct result. Trolls are "negative" towards the task, always trying to provide an answer from the set of incorrect solutions. All workers are subject to an *error probability* ϵ that deviates a worker from its specification. For simplicity of presentation we will assume that ϵ is the same for all workers. In particular, an altruistic worker replies with a correct result with probability $1 - \epsilon$ and with an incorrect result with probability ϵ, while for a troll holds the contrary. For all workers, the correct and incorrect results are selected uniformly at random from the respective sets $S(t)$ and $R(t)$ for a task t. When $\epsilon = 0$ an altruistic worker always replies with the correct result and a troll always replies with an incorrect result.

Result Evaluation: We assume that the master can use two techniques to determine the correct task result: *challenges* (C) and *voting* (V) . A *challenge* is a task of which the master knows the correct result. When a challenge is used, the master knows if a worker replied with a correct or an incorrect result. This information can help the master determine the correct result for each task in $\mathcal{T}$. When *voting* is used, the same task is assigned to multiple workers, and the master uses some voting technique to decide upon the correct task result.

Definition 2.2 (Environmental Parameters.): A master-worker system environment can be characterized by the

parameters (ϵ, s, r, f_a, T) where: (i) ϵ is the worker error probability, (ii) s is the set of correct replies for each task t, (iii) r is the set of incorrect replies for each task t (iv) f_a the fraction of altruistic workers, and (v) $T \subseteq \{C, V\}$ the technique used for the evaluation of the reported results.

Probabilities: We use the common definition of *an event $\mathcal{E}$ occurring with high probability* (*whp*) to mean that $\mathsf{Pr}[\mathcal{E}] = 1 - O(n^{-\alpha})$ for some constant $\alpha > 0$.

III. Exact Worker Behavior ($\epsilon = 0$)

In this section we examine the simple case where it is given that the error probability of each worker is $\epsilon = 0$. Hence, altruistic workers always reply with a correct result and trolls always reply with an incorrect result. This scenario is somewhat idealistic in practical master-worker systems, but it is used here to provide the best case analysis of our problem. The results of this section will be used as a reference from the next section, in order to evaluate the performance of the proposed algorithms compared to this optimistic scenario. Notice that any algorithm requires at least $\frac{n}{n_a}$ rounds to complete the computation of all the $|\mathcal{T}| = n$ tasks, and needs to perform n work, if all tasks have to be computed correctly with full reliability.

Algorithm 1 Simple algorithm MWSIMPLE_0 where $\epsilon = 0$ and $T = \{C\}$.

```
1: Send challenge task t to all workers in W
2: R[j] ← result received from w_j ∈ W, j ∈ [1, |W|]
3: U_a ← {w_i | R[i] is correct}
4: for i = 1 : |U_a| : n do          ▷ for loop increments i by |U_a|
5:     Send task t_{i+k-1} to kth worker in U_a, k ∈ [1, |U_a|]
6:     Add received result for t_{i+k-1} into Results[i + k]
7: end for
8: return Results
```

In the simple algorithm MWSIMPLE_0 that appears in Algorithm 1, the set of altruistic workers is not known and thus we use challenges ($T = \{C\}$) to determine it in a single round. Once the altruistic workers are identified the master makes unique task assignments to each of them. Hence the master needs $1 + \frac{n}{n_a}$ rounds to decide for n tasks, and requires $2n$ amount of work.

Theorem 3.1: Algorithm MWSIMPLE_0 has asymptotically optimal time $\Theta(\frac{n}{n_a})$ and optimal work $\Theta(n)$, and compute all the n tasks with probability 1, when $\epsilon = 0$.

Looking more closely at the general case of algorithms that only use voting ($T = \{V\}$) we have found that it is possible to solve all the tasks efficiently if $n_a > s \cdot n_t$. The algorithm MWVOTE_0 that solves the problem is given in Algorithm 2. In this algorithm the master sends the first task t_1 to all the workers. No incorrect returned value can appear more than n_t times, while from the pigeonhole principle at least one correct value appears at least $n_a/s > n_t$ times. Then, the workers that return values with multiplicity larger than n_t are all altruistic. These workers are stored in U_a and used to solve the rest of tasks. The size of U_a is at least $n_a - n_t(s-1) > n_t$, and hence the master needs at most $1 + \frac{n-1}{n_a - n_t(s-1)}$ rounds and $2n - 1$ work.

Algorithm 2 Simple algorithm MWVOTE_0 where $\epsilon = 0$, $n_a > s \cdot n_t$, and $T = \{V\}$.

```
1: Send task t_1 to all workers in W
2: Add worker w_j to set R[v] if it replied with value v
3: U_a ← ⋃_{v:|R[v]|>n_t} R[v]
4: Results[1] ← any value v : |R[v]| > n_t
5: for i = 2 : |U_a| : n do          ▷ loop increments i by |U_a|
6:     Send task t_{i+k-1} to kth worker in U_a, k ∈ [1, |U_a|]
7:     Add received result for t_{i+k-1} into Results[i + k - 1]
8: end for
9: return Results
```

From the above we have the following theorem:

Theorem 3.2: The algorithm MWVOTE_0 compute all the n tasks with probability 1 when $\epsilon = 0$ and $n_a > s \cdot n_t$. It has time $O(\frac{n}{n_t})$ and optimal work $\Theta(n)$.

In the case that $n_a = s \cdot n_t$ we have a negative result.

Theorem 3.3: If $\epsilon = 0$ and $n_a = s \cdot n_t$, then for any $r > 0$ there exists no algorithm that allows the master node to returns the correct result of a task t with probability greater than $\frac{s}{s+1}$ in any execution.

Proof: Lets assume that there exists such an algorithm A_m. The master M assigns the task t to a subset W' of the set of workers W. We assume w.l.o.g. that it sends the task to all the workers in W, since A_m can disregard the replies from the workers not in W'. Note that the master does not have any prior information on the correct answers or the answers that each worker returns. So we can examine possible executions for the same task t, where the incorrect results and the troll workers are different. Since $\epsilon = 0$, an altruistic worker always returns a correct answer and a troll returns an incorrect answer. We assume that the same worker returns the same answer if asked to compute the task more than once.

Consider now an execution ξ_1 of A_m constructed as follows. Let $D(t) = \{a_1, \ldots, a_s, a_{s+1}, \ldots, a_{s+r}\}$ be the domain of possible replies for t, where $S(t) = \{a_1, \ldots, a_s\}$ is the set of correct results for t and the rest of the answers are incorrect. Since $\epsilon = 0$ then the master receives n_a correct answers and n_t incorrect answers. Let us assume, that the troll workers are the set $\{w_{n_a+1}, \ldots, w_{n_a+n_t}\}$. Furthermore in ξ_1, let each answer $a_i \in S(t)$ be returned by $\frac{n_a}{s}$ workers. W.l.o.g assume that workers $\{w_{(i-1)\frac{n_a}{s}+1}, \ldots, w_{i\frac{n_a}{s}}\}$ reply with answer $a_i \in S(t)$. Observe that all the trolls reply with the same incorrect answer a_{s+1}. Since, $n_a = s \cdot n_t$, it follows that $n_t = \frac{n_a}{s}$ troll workers reply with a_{s+1}. Since, according to our assumption, algorithm A_m returns a correct answer with probability $p_c > \frac{s}{s+1}$, then it follows by the pigeonhole principle, that A_m will return some answer $a_i \in S(t)$ with probability $Pr[a_i] > \frac{s}{(s+1)s} = \frac{1}{s+1}$, in ξ_1. Let w.l.o.g. a_1 be that answer.

Assume now a second execution ξ_2 which is similar with ξ_1 with the difference that the troll workers are the set $\{w_1, \ldots, w_{\frac{n_a}{s}}\}$ and the correct answers is the set

$S(t) = \{a_2, \ldots, a_{s+1}\}$. Each answer is returned by the same set of workers as in ξ_1. Note that correct workers $\{w_{n_a+1}, \ldots, w_{n_a+n_t}\}$ all reply with a_{s+1}. Also the troll workers in ξ_2 all reply with a_1. Since all the workers reply with the same answers to the master in both ξ_1 and ξ_2, and since the master does not have any prior info on the correct answers, then M will not be able to distinguish ξ_1 from ξ_2. Thus, if according to A_m, M returned a_1 with probability $Pr[a_1] > \frac{1}{s+1}$ in ξ_1 then M will return a_1 with the same probability in ξ_2 as well. Since a_1 is an incorrect answer in ξ_2, then M returns a correct answer with probability $\mathbf{Pr}[$ return $a_i \in S(t)] = 1 - Pr[a_1] < 1 - \frac{1}{s+1} = \frac{s}{s+1}$. This however contradicts our initial assumption that A_m returns a correct answer with probability greater than $\frac{s}{s+1}$ in ξ_2 and completes our proof. ∎

Looking at the above two results one may conjecture that $n_a = s \cdot n_t$ is in fact the boundary between solvability and unsolvability. However, this is not the case, since, for instance, if $r = 1$ and n_a is not a multiple of n_t, even if $n_a < s \cdot n_t$ it is possible to solve all tasks efficiently. Using the opposite logic the incorrect value will appear n_t times, while from the pigeonhole principle at worst one correct value appears $n_a \mod n_t$ times. Then, the workers that return values with cardinality smaller than n_t are all altruistic. These workers like in Algorithm 2 can be stored in U_a and used to solve the rest of tasks following an analogous algorithm. The size of U_a is at least $n_a - \left\lfloor \frac{n_a}{n_t} \right\rfloor \cdot n_t$, and hence the master needs at most $1 + \frac{n-1}{n_a - \left\lfloor \frac{n_a}{n_t} \right\rfloor \cdot n_t} < n$ rounds and $2n - 1$ work. Thus, an algorithm analogous to Algorithm 2 has time $O(n)$ and optimal work $\Theta(n)$.

IV. Probabilistic Worker Behavior ($\epsilon > 0$)

We are now moving to the case where the error probability of each worker is $0 < \epsilon < \frac{1}{2}$. In this case an altruistic worker may reply with an incorrect result, or a troll with a correct result with probability ϵ. Note that we do not consider the case when $\epsilon \geq \frac{1}{2}$, because in that case essentially the roles are switched. We provide algorithms that cover the full spectrum of values for the density of solutions $\rho \in (0, 1)$ and the fraction of altruistic workers $f_a \in (0, 1]$. Notice that the algorithms presented here also apply in the case where $\epsilon = 0$, but they may induce extra performance overhead.

Under this model the master receives the correct result from a randomly selected worker with probability at least $f_a(1 - \epsilon)$. We provide two different algorithms for this setting: (i) algorithm MWMIX that uses both challenges and voting, i.e. $T = \{C, V\}$, and (ii) algorithm A_2 that uses only voting, i.e. $T = \{V\}$, which is possible only when the density of solutions satisfy a given bound.

Note that in this section we do not present an algorithm that only relies on challenges to compute all tasks in $\mathcal{T}$ with high probability. This is so, because even if the set W_a of altruistic workers is known, and only these workers are used, the value returned by the execution of a task is correct only with constant $1-\epsilon$ probability. An algorithm that does not use some form of voting will not execute a task more than once, and cannot improve this probability.

Algorithm 3 The pseudo-code for algorithm MWMIX, at the master, with n workers W computing the results of n tasks in $\mathcal{T}$, where $\frac{s}{s+1} < 1 - \epsilon$ and $T = \{C, V\}$.

```
   Phase 1
 1: R[1..n] ← ∅^n                ▷ R[j] is the list of results from worker w_j
 2: for i = 1 : ⌈c log n⌉ do
 3:     Send challenge task t to all workers in W
 4:     Add received result from worker w_j to R[j]
 5: end for
 6: for i = 1 : n do
 7:     if # correct results in R[i] ≥ ⌈½ c log n⌉ then
 8:         U_a ← U_a ∪ {w_i}
 9:     end if
10: end for
   Phase 2
11: F[i] ← ∅                     ▷ initially empty for all 1 ≤ i ≤ n
12: for j = 1 : ⌈k log n⌉ do
13:     for i = 1 : |U_a| : n do          ▷ loop increments i by |U_a|
14:         Send task t_{i+k-1} to kth worker in U_a, k ∈ [1, |U_a|]
15:         Add received result for t_{i+k-1} to F[i + k − 1]
16:     end for
17: end for
18: for i = 1 : n do
19:     Results[i] ← plurality(F[i])
20: end for
21: return Results
```

In this section we assume that the algorithms know the parameters s, f_a and ϵ. For the case that this is not true, we provide algorithm E_1 in the next section, that uses challenges to estimate them. Due to lack of space the proofs of correctness and performance of the algorithms are omitted.

A. Algorithm MWMIX: Challenges and Voting

In this section we present algorithm MWMIX, that uses a combination of challenges and voting. The general idea of the algorithm is to identify the workers in W_a, correctly *whp* and then use the estimated set U_a to compute *all* the tasks correctly *whp*. Below we describe algorithm MWMIX, with the pseudo code in Algorithm 3, using this strategy. As explained earlier, it is preferable to avoid using *challenge* method ($T = \{C\}$) because challenges imply computational burden on the master. Therefore, the algorithm uses challenges only to estimate U_a, and afterwards it uses voting ($T = \{V\}$) to determine the correct result for each task, i.e. $T = \{C, V\}$.

Description of algorithm MWMIX*:* Algorithm MWMIX consists of two phases. During the first phase, MWMIX computes an estimate of the set W_a, denoted by U_a, by using the challenge method. Phase 1 has $c \log n$ rounds, for a constant $c > 0$ that depends on ϵ (L:2). During each round the master sends out a distinct challenge task to every worker in W (L:3), and upon receiving the reponses from the workers stores the results in an array of lists $R[1], R[2], \cdots, R[n]$, where $R[i]$ denotes the list of results received from process $w_i \in W$ (L:4). Next based on the results in $R[\ ]$, the ID of

Algorithm 4 Algorithm MWVOTE, at the master process, performs n tasks using n workers for the case $\frac{s}{s+1} < f_a(1-\epsilon)+(1-f_a)\epsilon$ and $T=\{V\}$.

1: $F[i] \leftarrow \emptyset$ ▷ initially empty for all $1 \le i \le n$
2: **for** $i = 1$ to $\lceil k \log n \rceil$ **do** ▷ for some constant $k > 0$
3: Choose a random permutation $\pi \in \Pi_n$
4: Send each task $t_j \in \mathcal{T}$ to worker $w_{\pi(j)}$
5: Add received result from worker $w_{\pi(j)}$ to $F[j]$
6: **end for**
7: **for** $i = 1 : n$ **do**
8: $Results[i] \leftarrow plurality(F[i])$
9: **end for**
10: **return** $Results$

any worker that answered the majority of the challenge tasks correctly is included in the set U_a (L:8).

During the Phase 2, only the workers in U_a are used to compute the n tasks. Each task is executed $\lceil k \log n \rceil$ times. The results sent back by the workers in U_a are stored in the array of lists $F[1], F[2], \cdots, F[n]$ (L:15), where the results for task t_i are stored in list $F[i]$. Finally, the master decides for every task t_i the plurality of results in $F[i]$ to be the correct result. The results of the tasks in $\mathcal{T}$ are hence stored in the array variable $Results$, where $Results[i]$ is the result of task t_i (L:19).

Correctness and Performance Analysis: Now, we prove the correctness of all the tasks *whp* and the complexity of results. In Lemma 4.1, we show that at the end of Phase 1 every altruistic worker and only altrusitic workers are included in U_a. Using this lemma, we prove Theorem 4.1 which states that every task in $\mathcal{T}$ is computed correctly *whp*.

Lemma 4.1: In any execution of MWMIX, at the end of Phase 1 we have $U_a = W_a$, whp.

Theorem 4.1: If $\frac{s}{s+1} < 1-\epsilon$, then Algorithm MWMIX computes all n tasks correctly, whp.

Theorem 4.2: Algorithm MWMIX runs in $\Theta(\frac{n}{n_a}\log n)$ synchronous rounds and performs $\Theta(n \log n)$ work.

B. Algorithm MWVOTE: Use Voting Alone

In this section, we present algorithm MWVOTE that uses voting mechanism alone (i.e., when $T = V$) to compute all the n tasks *whp*. Algorithm MWVOTE can be used when $\frac{s}{s+1} < f_a(1-\epsilon)+(1-f_a)\epsilon$. Note here that, without identifying the altruistic workers, the probability that a randomly selected worker will reply with the correct answer for a task t is $f_a(1-\epsilon)+(1-f_a)\epsilon$; i.e. receive the correct answer from an altruistic worker with probability $1-\epsilon$ or to receive the correct answer from a troll with probability ϵ.

Description of algorithm MWVOTE*:* Below we present our algorithm MWVOTE, and the pseudo-code for the algorithm is in Algorithm 4. The basic idea of MWVOTE is to exploit the fact that if a troll worker picks an answer for a task t randomly from $R(t)$ and $\frac{s}{d}$ is small, then the likelihood of an incorrect result being a majority or plurality among all the results is small. To apply this idea, the master distributes the n tasks according to a random permutation from Π_n, which is the set of all permutations over $[n] = \{1, 2, \ldots, n\}$. In other words, if the random permutation is π then the j^{th} task t_j is delegated to worker $w_{\pi(j)}$ (L:4). The whole process is repeated $\lceil k \log n \rceil$ rounds (L:2-6). The constant k is used to tune the exponent $\ell > 0$ in the denominator of $\frac{1}{n^\ell}$, required for the high probability guarantee. The results for each task t_j is accumulated in the multiset $R[j]$. When the **for** loop in lines 2–6 terminates, the result for each task $t_j \in \mathcal{T}$ is chosen to be the one that forms a plurality of results in $R[j]$ (L:8).

Correctness and Performance Analysis: The following theorems state that algorithm MWVOTE computes all tasks correctly *whp* under the assumed case.

Theorem 4.3: If $\frac{s}{s+1} < f_a(1-\epsilon)+(1-f_a)\epsilon$, Algorithm MWVOTE computes all n tasks correctly *whp*.

Theorem 4.4: Algorithm MWVOTE runs in $\Theta(\log n)$ synchronous rounds and performs $\Theta(n \log n)$ work.

Remark 4.1: Note that the algorithms presented in this section have a log factor overhead on the optimal work (Theorem 3.1). According to [12] a factor of log work overhead is the least amount of work required per task when voting is used. Thus, we can safely conclude on the optimality of both algorithms MWMIX and MWVOTE.

V. Algorithm E_1: Tightly Estimating f_a and ϵ

As shown in Section IV, algorithms MWMIX and MWVOTE are possible only if we know parameters of the system, like the probability of error ϵ, the fraction of altruistic workers f_a, and the number of correct results s, to check applicability. In this section, we assume s is known, but that neither the value of f_a and ϵ are known *a priori*. (Note that it is reasonable to assume that the number of correct answers s is known.) Hence, we provide an algorithm to estimate f_a and ϵ, *whp*. As a byproduct, the algorithm also estimates $f_a(1-\epsilon)+(1-f_a)\epsilon$, *whp*. Our goal is to estimate all these values, with user defined bounds, in a manner called (ε, δ)-approximation. By choosing $\varepsilon, \delta \in O(\frac{1}{n^c})$ for some $c > 0$ we can provide a tight estimate of the value within a $\pm\varepsilon$ factor and with high probability (greater than $1-\delta$).

Formally, let Z be a random variable distributed in the interval $[0, 1]$ with mean μ_Z. Let $Z_1, Z_2, \ldots$ be independently and identically distributed according to the Z variable. We say that an estimate $\tilde{\mu}_Z$ is an (ε, δ)-approximation of μ_Z if $\mathbf{Pr}[\mu_Z(1-\varepsilon) \le \tilde{\mu}_Z \le \mu_Z(1+\varepsilon)] > 1-\delta$. Estimating the value of μ_Z may be done by collecting *sufficient* samples and selecting the majority as the outcome. However, such a solution is not feasible if the number of samples are not known *a priori*.

The Stopping Rule Algorithm: Algorithm 6 is an algorithm for calculating an (ε, δ)-approximation of the desired parameters, where the error tolerance bounds δ and ε are $O(\frac{1}{n^c})$, for some $c > 0$. The core idea behind E_1 is based on the Stopping Rule Algorithm (*SRA*) of Dagum *et al.* [6]. For completeness we reproduce in Algorithm 5 the *SRA* for estimating the mean of a random variable with support in $[0, 1]$, with (ε, δ)-approximation. *SRA* computes an (ε, δ)-approximation with an

Algorithm 5 The Stopping Rule Algorithm (*SRA*) for estimating μ_Z.

input parameters: (ε, δ) with $0 < \varepsilon < 1$, $\delta > 0$
Let $\Gamma = 4\lambda \log(\frac{2}{\delta})/\varepsilon^2$ ▷ $\lambda = (e-2) \approx 0.72$
Let $\Gamma_1 = 1 + (1+\varepsilon)\Gamma$
initialize $N \leftarrow 0, S \leftarrow 0$
while $S < \Gamma_1$ **do**
    $N \leftarrow N + 1$
    $S \leftarrow S + Z_N$
end while
return $\tilde{\mu}_Z \leftarrow \frac{\Gamma_1}{N}$

optimal number of samplings, within a constant factor [6]. Thus *SRA*-based method provides substantial computational savings. Let us define $\lambda = (e-2) \approx 0.72$ and $\Gamma = 4\lambda \log(\frac{2}{\delta})/\varepsilon^2$. Now, Theorem 5.1 (slightly modified, from [6]) tells us that *SRA* provides us with an (ε, δ)-approximation with the number of trials within $\frac{\Gamma_1}{\mu_Z}$ *whp*, where $\Gamma_1 = 1 + (1+\varepsilon)\Gamma$.

Theorem 5.1: [*Stopping Rule Theorem*] Let Z be a random variable in $[0,1]$ with $\mu_Z = \mathbb{E}[Z] > 0$. Let $\tilde{\mu}_Z$ be the estimate produced and let N_Z be the number of experiments that *SRA* runs with respect to Z on inputs ε and δ. Then, (i) $\mathbf{Pr}[\mu_Z(1-\varepsilon) \leq \tilde{\mu}_Z \leq \mu_Z(1+\varepsilon)] > 1-\delta$; (ii) $\mathbb{E}[N_Z] \leq \frac{\Gamma_1}{\mu_Z}$, and (iii) $\mathbf{Pr}[N_Z > (1+\varepsilon)\frac{\Gamma_1}{\mu_Z}] \leq \frac{\delta}{2}$.

Description of algorithm E_1: The idea behind algorithm E_1 is to sample two binary random variables: (i) $Z^1 \in \{0,1\}$, whose mean is "close" to f_a; and (ii) $Z^2 \in \{0,1\}$, whose mean is $f_a(1-\epsilon) + (1-f_a)\epsilon$. Then E_1 creates (ε, δ)-approximation estimates for both of these means, using the *SRA* algorithm for $\delta, \varepsilon \in O(\frac{1}{n^c})$, for some $c > 0$. Using these estimates, it solves for a (ε, δ)-approximation for the different parameters. Below we explain the sampling process.

Z^1 is defined as follows: the master randomly picks a worker w from W, sends ℓ (a positive integer, explained later) challenges to w, and collects and verifies the results. If the majority of the results R are correct then $Z^1 = 1$, otherwise $Z^1 = 0$. We use $CorrMaj(R)$ to denote that the majority of the results in R are correct. Clearly,

$$\begin{aligned}\mathbb{E}[Z^1] &= \mathbf{Pr}[w \in W_a] \cdot \mathbf{Pr}[CorrMaj(R) | w \in W_a] \\ &\quad + \mathbf{Pr}[w \notin W_a] \cdot \mathbf{Pr}[CorrMaj(R) | w \notin W_a] \\ &= f_a \cdot \mathbf{Pr}[CorrMaj(R) | w \in W_a] \\ &\quad + (1-f_a) \cdot \mathbf{Pr}[CorrMaj(R) | w \notin W_a]\end{aligned}$$

Next, by exploiting the fact that $\epsilon < \frac{1}{2} - \zeta$, where $\zeta > 0$ is a constant, we choose ℓ appropriately, such that, $\mathbf{Pr}[CorrMaj(R) | w \in W_a] \approx 1$ (i.e., $1 - O(\frac{1}{n^c})$, for some $c > 0$) and $\mathbf{Pr}[CorrMaj(R) | w \notin W_a]$ becomes very small (i.e., $O(\frac{1}{n^c})$, for some $c > 0$). Hence $\mathbb{E}[Z^1]$ approximated suitably enough $f_a = \mathbf{Pr}[w \in W_a]$. Lines 4–15 for algorithm E_1 implements the *SRA* algorithm to estimate $\mathbb{E}[Z^1]$.

Z^2 is defined as follows: the master randomly picks a worker w from W, assigns a challenge to w, and verifies the reported result. If the result is correct then $Z^2 = 1$, otherwise $Z^2 = 0$. Note that

$$\begin{aligned}\mathbb{E}[Z^2] &= \mathbf{Pr}[w \in W_a] \cdot \mathbf{Pr}[result\ is\ correct | w \in W_a] \\ &\quad + \mathbf{Pr}[w \notin W_a] \cdot \mathbf{Pr}[result\ is\ correct | w \notin W_a] \\ &= f_a(1-\epsilon) + (1-f_a)\epsilon.\end{aligned}$$

Lines 16–24 for algorithm E_1 implement the *SRA* algorithm to estimate $\mathbb{E}[Z^2]$.

Algorithm 6 Algorithm E_1 to estimate f_a, ϵ, and $f_a(1-\epsilon) + (1-f_a)\epsilon$.

1: Let $\delta = \frac{1}{n^c}$ and $\varepsilon = \frac{1}{n^c}$ for $c > 0$
2: Let $\Gamma = (4\lambda \log(\frac{2}{\delta}))/\varepsilon^2$ and $\Gamma_1 = 1 + (1+\varepsilon)\Gamma$
3: Let $\ell = \lceil k \log n \rceil$, for some $k > 0$
4: $N \leftarrow 0, S \leftarrow 0$
5: **while** $S < \Gamma_1$ **do**
6: $N \leftarrow N + 1$
7: pick a worker w randomly uniformly from W
8: **for** $i = 1$ to ℓ **do**
9: send challenge task t_i to w
10: $R[i] \leftarrow$ result received from w
11: **end for**
12: **if** $CorrMaj(R)$ **then** $Z^1_N \leftarrow 1$ **else** $Z^1_N \leftarrow 0$ **end if**
13: $S \leftarrow S + Z^1_N$
14: **end while**
15: $\tilde{p} \leftarrow \frac{\Gamma_1}{N}$
16: $N \leftarrow 0, S \leftarrow 0$
17: **while** $S < \Gamma_1$ **do**
18: $N \leftarrow N + 1$
19: pick a worker w randomly uniformly from W
20: send challenge task to w
21: **if** result received from w is correct **then** $Z^2_N \leftarrow 1$ **else** $Z^2_N \leftarrow 0$ **end if**
22: $S \leftarrow S + Z^2_N$
23: **end while**
24: $\tilde{q} \leftarrow \frac{\Gamma_1}{N}$
25: **return** $\left(\tilde{p}, \frac{\tilde{q}-\tilde{p}}{1-2\tilde{p}}, \tilde{q}\right)$

Analysis of the algorithm: In the following theorem we state that E_1 provides suitable approximation for the different parameters of the system.

Theorem 5.2: The estimates $\tilde{p}$, $\frac{\tilde{q}-\tilde{p}}{1-2\tilde{p}}$, and $\tilde{q}$ returned by E_1 are (ε, δ)-approximations of the parameters f_a, ϵ, and $f_a(1-\epsilon) + (1-f_a)\epsilon$, respectively, where $\varepsilon, \delta \in O(\frac{1}{n^\gamma})$, for some $\gamma > 0$.

Theorem 5.3: The number of rounds or the work for algorithm E_1 is $n^c \log n$ for $c > 0$ *whp*.

Proof: The number of times the **while** loop iterates is the value of N at the end of the looping. The value of N at the end of any of the **while** loop is $n^c \log n$ *whp*, which can be proved by substituting $\frac{1}{n^c}$ for δ and ε in the Γ_1 (see Algorithm 6) and applying Theorem 5.1(iii). Now, in the first **while** the **for** loop runs for $\Theta(\log n)$ iterations. Therefore, the rounds and work are $O(n^c \log^2 n)$, *whp*. ■

VI. Conclusions and Future Work

This work considers the master-worker paradigm for Internet-based computation of tasks, in the presence of altruistic and troll workers subject to an error probability that

can alter their intended behavior. A generic model capturing the parameters of the master-worker paradigm is presented; deviating from the usual conventions made in the literature. The error probability that workers might have and moving away from the usual assumption that a task can only have one correct and one incorrect result made also our modelling more realistic.

There are still many aspects of the system that are not captured by our proposed model. For example, we do not consider the possibility that workers might be unavailable at stages in the execution, or that they may express dynamic behavior by experiencing different error probability over time. Moreover, it would be interesting to study algorithms that adapt when each task has a different number of correct and incorrect results. We have assumed for simplicity of presentation that ϵ is the same for all workers, this assumption also helped to gain a better understanding of our master-worker setting. In the future we plan to remove this assumption and consider that each worker has it's own error probability that can even change over time. Thus, we will adapt our algorithms to reputation techniques designed to help us estimate the reliability of each worker. Such additional considerations will allow our model to capture more complex environments like, for example, the behavior of crowdsourcing systems.

REFERENCES

[1] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[2] D. P. Anderson. Volunteer computing: the ultimate cloud. *ACM Crossroads*, 16(3):7–10, 2010.

[3] D. P. Anderson. BOINC, 2016. http://boinc.berkeley.edu/.

[4] D. M. Blough and G. F. Sullivan. A comparison of voting strategies for fault-tolerant distributed systems. In Proc. of *RDS'90*, pages 136–145, 1990.

[5] G. Casella and R. L. Berger. *Statistical Inference*. Duxbury Advanced Series, second edition, 2001.

[6] P. Dagum, R.M. Karp, M. Luby, and S. Ross. An optimal algorithm for monte carlo estimation. In *Proceedings of the Foundations of Computer Science*, pages 142–149, 1995.

[7] W. de Zutter. BOINC stats, 2016. http://boincstats.com/en/forum/10/4597.

[8] A. Fernández, C. Georgiou, L. López, and A. Santos. Reliably executing tasks in the presence of malicious processors. In *Distributed Computing*, pages 490–492. Springer, 2005.

[9] A. Fernández Anta, C. Georgiou, L. López, and A. Santos. Reliable internet-based master-worker computing in the presence of malicious workers. *Parallel Processing Letters*, 22(01), 2012.

[10] E. M. Heien, D. P Anderson, and K. Hagihara. Computing low latency batches with unreliable workers in volunteer computing environments. *Journal of Grid Computing*, 7(4):501–518, 2009.

[11] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello. Characterizing result errors in internet desktop grids. In *Euro-Par 2007 Parallel Processing*, pages 361–371. Springer, 2007.

[12] K. M. Konwar, S. Rajasekaran, and A. A. Shvartsman. Robust network supercomputing with unreliable workers. *J. Parallel Distrib. Comput.*, 75:81–92, 2015.

[13] E. Korpela, D. Werthimer, D. P Anderson, J. Cobb, and M. Lebofsky. Seti@ homemassively distributed computing for seti. *Computing in science & engineering*, 3(1):78–83, 2001.

[14] A. Kumar and K. Malik. Voting mechanisms in distributed systems. *Reliability, IEEE Transactions on*, 40(5):593–600, 1991.

[15] M. Paquette and A. Pelc. Optimal decision strategies in byzantine environments. *Journal of Parallel and Distributed Computing*, 66(3):419–427, 2006.

[16] L. FG Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems*, 18(4):561–572, 2002.

[17] J. Sonnek, M. Nathan, A. Chandra, and J. Weissman. Reputation-based scheduling on unreliable distributed infrastructures. In *IEEE ICDCS06*, pages 30–30, 2006.

[18] S. Zhao, V. Lo, and C G. Dickey. Result verification and trust-based scheduling in peer-to-peer grids. In *IEEE P2P05*, 2005.

NoSQL Undo: Recovering NoSQL Databases by Undoing Operations

David Matos Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

***Abstract*—NoSQL databases offer high throughput, support for huge data structures, and capacity to scale horizontally at the expense of not supporting relational data, ACID consistency and a standard SQL syntax. Due to their simplicity and flexibility, NoSQL databases are becoming very popular among web application developers. However, most NoSQL databases only provide basic backup and restore mechanisms, which allow recovering databases from a crash, but not to remove undesired operations caused by accidental or malicious actions. To solve this problem we propose NoSQL Undo, a recovery approach and tool that allows database administrators to remove the effect of undesirable actions by undoing operations, leading the system to a consistent state. NoSQL Undo leverages the logging and snapshot mechanisms built-in NoSQL databases, and is able to undo operations as long as they are present in the logs. This is, as far as we know, the first recovery service that offers these capabilities for NoSQL databases. The experimental results with MongoDB show that it is possible to undo a single operation in a log with 1,000,000 entries in around one second and to undo 10,000 incorrect operations in less than 200 seconds.**

I. Introduction

Most NoSQL databases aim to provide high performance for large-scale applications [1], [2]. Their ability to split records and scale-out horizontally allows them to maintain performance when dealing with high traffic loads and peaks. In comparison with traditional relational databases, NoSQL databases offer better performance and availability, over strong consistency, relational data and ACID properties [1]. These characteristics make NoSQL databases a good choice for applications with high availability and scalability requirements, but no need of strong consistency and complex transactions.

Currently there are many NoSQL databases.[1] Some of the best known are: Cassandra [3], MongoDB [4], Hadoop HBase [5], Couchbase [6], DynamoDB [7], and Google BigTable [8]. These databases vary mainly in the format of stored data, which can be key-value [7], columnar [3], [8], [9], or document oriented [4]. In terms of scalability, all these systems can be deployed in large clusters and have the ability to easily extend to new machines and to cope with failures.

Most NoSQL databases offer simple recovery mechanisms based in *local logs* and *snapshots* that support data recovery when a server crashes. They also use *global logs* that keep data consistency across replicas. These mechanisms are useful but not sufficient to remove the effect of *faulty operations* from the system state, e.g., of an exploit to a website vulnerability or an incorrect update command by a database administrator that changes or deletes the wrong document. If an incorrect operation is executed and corrupts the database, an administrator may restore an old snapshot that does not include the faulty operation. However, although this solution removes the faulty operation from the database it also discards correct state changes. Even worse, if the faulty operation is detected late, the amount of data lost may be huge. A better solution is to manually execute a command, such as an update or a delete, that removes the effects of the incorrect operation, but this is difficult and time consuming for the administrator. The time it takes to recover a database is critical; if a recovery takes too long it could be impossible to successfully recover the data without collateral damage.

This paper presents NoSQL Undo, a recovery approach and tool that allows database administrators to automatically remove the effect ("undo") of faulty operations. NoSQL Undo is a client-side tool in the sense that it does not need to be installed in the database server, but runs similarly to other clients. Unlike recovery tools in the literature [10]–[12], NoSQL Undo does not require an extra server to act as proxy since it uses the built-in log and snapshots of the database to perform recovery. It also does not require extra meta-data or modifications to the database distribution or to the application using the database. The tool offers two different methods to recover a database: *Full Recovery* that performs better when removing a large amount of incorrect operations; and *Focused Recovery* that requires less database writes when there are just a few incorrect operations to undo. NoSQL Undo supports the *replicated* (primary-secondary) and *sharded* architecture of NoSQL databases. The tool provides a graphical user interface so that a database administrator is able to easily and quickly find faulty operations and perform a recovery.

To evaluate NoSQL Undo, we integrated it with MongoDB and conducted several experiments using YCSB [13]. The latter is a benchmark framework for data-servicing services that provides realistic workloads that represent real-world applications. It allows configuring the amount of operations, records, threads and clients using the database. With the experimental evaluation we wanted to compare both approaches to undo incorrect operations (Focused Recovery and Full Recovery), and how both methods perform with different sets of operations. The experimental results show that it is possible to undo a single operation in a log with 1,000,000 entries in around one second and to undo 10,000 incorrect operations in less than 200 seconds.

[1] We use "databases" for short to mean database management systems.

978-1-5090-3216-7/16/$31.00

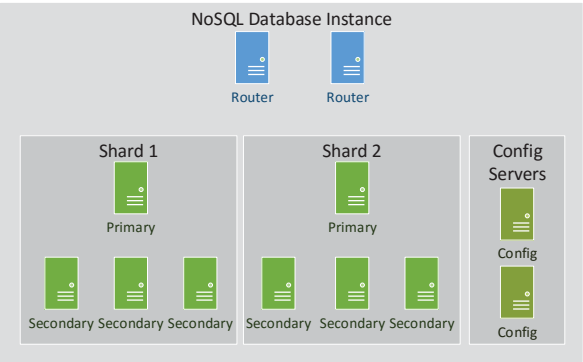


Fig. 1: Example of NoSQL instance with two shards.

This work provides the following contributions. First, a novel recovery approach and tool that supports the distributed – replicated and sharded – architecture of NoSQL databases, with two recovery methods, that does not require a proxy to log operations, and does not require modifications to the service. This is the first system combining these characteristics. It is also the first that supports such a replicated and sharded architecture, that does not require a proxy, and that does not require modifications to the service to support recovery. Second, a detailed experimental evaluation of the tool and comparison of the two recovery methods using YCSB.

II. NoSQL Databases

Most NoSQL databases provide replication, horizontal scaling, unstructured data storage and simple backup and restore capabilities. There are different NoSQL databases, but there are many common elements in their architectures. In some databases several elements can be incorporated in the same server [6], whereas others require that each component of the system is placed in a dedicated server [3], [4]. Some databases [3], [9] may require additional components, such as Zookeeper [14] to manage group membership. Despite their differences, NoSQL databases that provide replication and horizontal scaling tend to have a similar architecture.

A. Architecture

Figure 1 represents the architecture of a common NoSQL database instance with two *shards*. In this configuration each piece of data is divided into slices that are stored in the shards. Each shard is a collection of servers that store the same data, i.e., that are *replicas*. This redundancy provides fault tolerance (no data is lost in case a replica fails) and more performance (any of the servers can respond to read operations). In each replica a server acts as *primary* and coordinates the replication actions of the remaining, *secondary*, servers. The primary server is responsible for keeping data consistent inside a shard.

In order to correctly split data in shards, a special server (or a collection of servers, depending on the complexity of the instance) is responsible for redirecting the requests to the correct shards and divide the records. These servers are usually called *routers*. The routers are the components of the system that interact with the application. If there are no routers alive then the database is inaccessible. To prevent this usually there are several routers. Besides increasing availability of the database, having several routers also increases the performance since if reduces the bottleneck of having a single server responding to every request of the application.

Some NoSQL databases have extra servers that are responsible for recording configuration information of the instance (*configuration servers* in the figure). This allows separation of concerns: while some server are only responsible for storing data, other are responsible for storing metadata and configuration parameters of the instance.

This architecture is very similar to a distributed MongoDB instance. The main difference is that the configuration servers in a Mongo database are grouped in a replica set (primary with a set of secondary servers) which means that all the configuration servers store the exact same information. Cassandra also has a similar architecture except that the configuration servers are not present. The metadata and configuration parameters are stored in the data servers. HBase has a slightly different architecture. Master servers are in a group and manage how data is partitioned to the slave servers (also called region servers). This approach does not separate the servers by data partitions, but instead separates the servers in two groups: master servers and region servers. In Couchbase it is possible to aggregate every component in every server. This way all the servers perform the same tasks. It is also possible to deploy a Couchbase database having each server responsible for a specific task. This way the architecture of a Couchbase database would be like the one in Figure 1.

B. Logging Mechanisms

Most databases have a logging mechanism that records database requests and take periodic snapshots, allowing the recovery of the system in case of failure. NoSQL databases also have logging mechanisms and take snapshots to allow the recovery of an individual server. However, these logs are specific to an individual server, so if the entire database (all servers) fails it is difficult to recover it using the local logs of each server. Besides the local log of each server, most NoSQL databases also have a global log that is used to maintain consistency across all the servers. This global log is not intended to recover the database, but instead to guarantee that all the servers receive the same set of operation in the correct order.

1) Local logs: Any cluster should be prepared for single servers failing unexpectedly. After a failure, a server has to perform fail-over, i.e., to take the required actions to come back to work without interfering with the remaining servers. In order to do that the server must have a diary in which it records every operation it writes to disk, so in case of failure the server only needs to repeat every operation and it should reach the state right before the failure. In some cases this diary is not sufficient since it only contains recent operations, so it is necessary to use a snapshot (a full copy of the server in

a previous moment in time) of the server and complete the missing information with the entries in the diary. The diary in which the server stores the operations is a local log and the information it contains is usually non human readable and specific to the server itself. A simple query can be decomposed in several local log entries corresponding to all the disk writes necessary to execute that query.

2) Global logs: In order to keep data consistent across servers the requests have to be delivered in total order, i.e., all requests delivered in the same order to all servers. On the contrary, simply sending the requests to all the servers might not work since some messages could be lost in the network or delivered out of order. To prevent this most databases use a global log in which they store every request they receive. This log is then used to propagate the operations to all the servers. If a server fails to receive an operation it can later consult the global log and execute it. Some databases have a fixed storage limit for this global log, i.e., it is implemented as a circular array (older operations are overwritten by new ones to prevent the log from growing indefinitely). The way the operations are stored in the global log is usually similar to the way data is stored in the database, meaning that is is possible and fairly easy to perform normal queries over the global log. Random values are converted into deterministic values to guarantee that every operation in the log is idempotent. Besides the operation itself, each log entry contains also a numeric value that allows ordering the executed operations. This numeric value guarantees that every server in the instance is able to reach the same state, since they all execute the operations in the same order.

In most NoSQL databases the global log has a format that is close to the format of the requests, contains the executed queries, is in a human readable form, and stores the operations in the order they were executed. Therefore, in NoSQL Undo we use this log to perform recovery.

III. NoSQL Undo

NoSQL Undo is a client side tool that only accesses a NoSQL database instance when the database administrator wants to remove the effect of some operations from the database, e.g., because they are malicious. The client does not need to be connected to the server in run time since it uses the database built-in logs to do recovery.

A. Undo vs Rollback

Every database provides rollback capabilities, meaning that if an incorrect operation is detected and needs to be removed then it is possible to revert the entire database to a previous point in time prior to the execution of that incorrect operation. The problem with this approach is that every single correct operation executed after the point in time to recover is lost. Figure 2 represents the state of a database with three different documents (D1, D2 and D3). All three documents were updated 6 times, i.e., there are 6 versions of each of these documents in the log of the database. Two of these documents (D1 and D2) were corrupted by malicious operations (red dots). D1 is later updated with valid operations meaning that part of the document may be corrupted while the other part is valid. To use a traditional rollback, the administrator needs to select a point in time prior to any malicious operation, which in this case is when all documents were in version 2. After the rollback the database is clean, however all the documents were reverted to older versions. Every correct operation executed after version 2 is then lost. By using any of the algorithms of NoSQL Undo, the administrator is able to clean both D1 and D2 and still keep every correct operation that was executed after, since both algorithms correct the corrupted documents by removing the effects of the malicious operations, instead of replacing them with older versions.

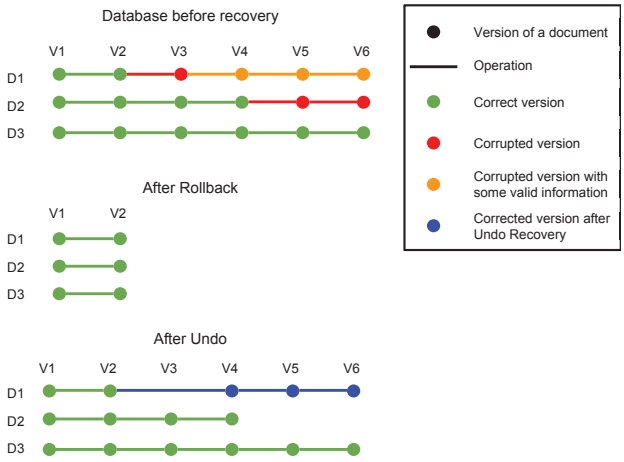


Fig. 2: Comparison of using rollback to recover a corrupted database with using undo to revert incorrect operations.

B. Architecture

Figure 3 presents an example of a NoSQL instance with NoSQL Undo. The architecture is logically divided in two layers: the database layer in which the database runs without any modification to the configuration or to the software; and the support layer, which includes optional modules that can also be deployed to improve the capabilities of NoSQL Undo.

In order to undo undesired operations there has to be a log with every executed operation and snapshots with previous versions of the database. Most recovery systems, such as Operator Undo [10] and Shuttle [15], implement a proxy that intercepts every request to the database and saves these requests in a specific log, independent from the DBMS. That approach may impact performance since every operation must pass through a single server [15]. The proxy may also be a single point of failure; if it fails, the clients become unable to contact the DBMS. It is possible to circumvent this limitation by replicating the proxy, however this introduces more complexity in the system. NoSQL Undo handles this issue using the built-in logs and snapshots to do recovery, so it does not require an additional server (proxy).

Since NoSQL Undo only accesses the built-in log to perform recovery it does not have control of when log entries

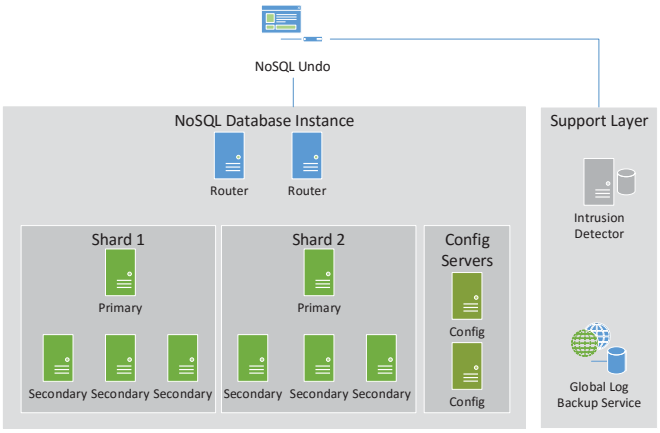


Fig. 3: A NoSQL database with NoSQL Undo.

are discarded. A database administrator may define a high storage threshold for the log, but an unpredictable peak of traffic may be enough to exhaust that limit. To guarantee that any operation can be undone, the log has to be saved regularly. NoSQL Undo does this using a service that runs along with the database instance and listens to changes in the log, the *Global Log Backup Service* (Figure 3). This service is a daemon that is constantly listening to the database for changes and keeping a copy of every log entry in an external database. Every time an operation is executed in the database instance, a new log entry is created and the global log backup service is notified, then it copies the global log record to another database that should be stored in a different server for availability purposes. This backup operation executes concurrently with the clients' operations and does not require the database to lock until the record was successfully stored, so it does not interfere with the original database functioning.

The last component of the architecture is the Intrusion Detector, which provides assistance with the process of identifying operations that need to be undone. We postpone the explanation of this component to Section III-D.

C. Recovery Mechanisms

NoSQL Undo uses two methods to undo the effects of incorrect operations leaving the database in a correct state: full recovery and focused recovery. Both methods take as input a list with operations to undo.

1) Full recovery: The full recovery algorithm is the simplest recovery method among the two. It works by loading the most recent snapshot of the database, then it updates the state by executing the remaining operations, which were previously recorded in a log. The algorithm takes as input a list of incorrect operations that it is suppose to ignore when it is executing the log operations.

Algorithm 1 shows the full recovery procedure. The algorithm takes as input the most recent snapshot before the first incorrect operation and a list with the incorrect operations to undo. In line 1 a new database instance is created using

Algorithm 1 Full recovery algorithm.

```
1: recoveredDatabase ← snapshot
2: logEntries ← getLogEntries(snapshot)
3: for logEntry ∈ logEntries do
4:     if logEntry ∉ incorrectLogEntries then
5:         recoveredDatabase.execute(logEntry)
6:     end if
7: end for
8: return recoveredDatabase
```

that snapshot. The log entries are fetched from the global log (line 2) using the *getLogEntries* method. This method returns an ordered list with every log entry after *snapshot*. Correct operations are executed in line 5. When the algorithm finishes (line 8), *recoveredDatabase* is a clean copy of the database without the effects of incorrect operations. This algorithm is simple and effective, but is not efficient when there are a small number of operations to undo, since it requires every correct operation in the log after the snapshot to be re-executed.

2) Focused recovery: The idea behind Focused Recovery is that instead of recovering the entire database just to erase the effects of a small set of incorrect operations, only compensation operations are executed. A compensation operation is an operation that corrects the effects of a faulty operation. The algorithm works basically the following way. For each faulty operation the affected record is reconstructed in memory by NoSQL Undo. When the record is updated, NoSQL Undo removes the incorrect record and inserts the correct one in the database. On the contrary of Algorithm 1, this algorithm only requires two write operations in the database for each faulty operation.

Algorithm 2 describes the process of erasing the effect of incorrect operations. The algorithm iterates through every incorrect operation (line 1). For each incorrect document it fetches every log entry that affected this record (line 3). For simplicity the pseudo-code assumes that there is no older snapshot and that every operation executed is in the log, therefore the recovered document is initialized empty (line 4). If there was a snapshot, then the recovered document would be initialized as a copy of the incorrect document in the snapshot. Then the reconstruction begins and every correct operation is executed in memory in the recovered record, not in the database (lines 4 to 6). Once every correct operation is executed, the record is ready to be inserted in the database. First the incorrect record is deleted (line 10) and finally the correct one is inserted (line 11).

3) Comparison of the two recovery schemes: Both methods are capable of removing the effects of undesirable operations, but there are differences. Focused Recovery does not require a new database to be created (*recoveredDatabase*) because it does compensation operations in the existing database. For each record affected by incorrect operations, it does two write operations in the database: one to remove the corrupted record, and another to insert the fixed record. On the contrary, with Full Recovery every correct operation executed after the last correct snapshot is executed in a new database instance. In terms of writes in the database, Focused Recovery is much

Algorithm 2 Focused recovery algorithm.

```
1: for incorrectOperation ∈ incorrectOperations do
2:     corruptedRecord ← incorrectOperation.getRecord()
3:     documentLogEntries ← getRecordLogEntries(corruptedRecord)
4:     recoveredRecord ← {}
5:     for recordLogEntry ∈ recordLogEntries do
6:         if recordLogEntry ≠ incorrectOperation then
7:             recoveredRecord ← updateRecord(recoveredRecord, recordLogEntry)
8:         end if
9:     end for
10:    database.remove(corruptedRecord)
11:    database.insert(recoveredRecord)
12: end for
```

lighter if the number of incorrect operations is reasonably small. On the contrary, if the number of incorrect operations is greater than the number of correct operations in the log, then using the Full Recovery will be more efficient because there are less write operations to execute in the database (see Section V-B).

Although the algorithms leave the database in a consistent state, a user that has read a corrupted document and suddenly reads the corrected document may believe that the state is inconsistent. To solve this problem, the tool can be configured to leave a message to the users so that they understand why the state suddenly changed [10].

Both algorithms are able to remove the effects of faulty operations but they require the database to be paused, i.e., not executing operations while recovering. If the database keeps serving clients, then data consistency after recovery cannot be guaranteed.

D. Detecting Incorrect Operations

NoSQL Undo provides an interface for administrators to select which operations should be discarded during recovery. This interface provides searching capabilities making it easier to find incorrect operations. An interesting case to use this tool is to do recovery from intrusions.

One of the problems in recovering faulty databases is to detect when the database became corrupted in the first place. Detecting the incorrect operations in a log with millions of operations can be a difficult and error prone task. Searching for regular expressions of possible attacks may not be sufficient since a database administrator may not remember every search pattern to all possible malicious operations.

To cope with this, it is possible to deploy alongside with the NoSQL database an Intrusion Detection System (IDS). This IDS permanently listens to the requests to the database and if they match a certain signature, it logs this operation as suspicious. The most conspicuous case of requests that match signatures are attempts of doing NoSQL injection attacks, which are similar to SQL injection attacks but target NoSQL databases [16], [17]. Later, when NoSQL Undo is being used it first consults all the suspicious operations and suggests them to the database administrator who then decides if they should be removed from the database. This automates the identification process and reduces the time to recovery, which can be critical in a highly accessible database. An example of an IDS that can be deployed in this manner is Snort [18]. Different solutions to detect malicious operations in the log can be used, such as [19]–[21].

E. Recovery Without Configuration

An interesting feature of NoSQL Undo is that it does not have to be configured *a priori* to be able to recover a database. If an incorrect operation is detected soon enough, i.e., while it is still present in the log, then it is possible to remove the effects of this faulty operation without any previous configuration of NoSQL Undo. This is interesting as many organizations do not take preventive measures to allow later recovery.

This approach however has some limitations in comparison to the full-fledged recovery scheme presented in the previous sections. When a recovery service uses specific logging mechanism it is able to store additional information useful for later recovery (e.g., dependencies between operations, origin of the operations and versions of the affected documents). With this extra information it is possible to improve the recovery process as well as provide more information to the database administrator to help him find the faulty operations.

IV. Implementation with MongoDB

To evaluate the proposed algorithms, a version of NoSQL Undo was implemented in Java. This instance of the tool allows recovering MongoDB databases.

A. MongoDB

MongoDB supports replication to guarantee availability if servers fail, and horizontal scaling to maintain performance when the traffic load increases [22], [23]. A set of servers with replicated data is called a *replica set*. In each replica set there is a server that coordinates the replication process called *primary*, while the remaining servers are called *secondary*. Each secondary server synchronizes its state with the primary. It is possible to fragment data records into MongoDB instances, called *shards*, to balance load. A shard can be either a single server or a replica set. Data in MongoDB is structured in *documents*. Each document contains a set of key-value pairs. A set of related documents is a *collection*. The documents in a collection do not need to have the same set of key-value pairs nor the same type, as opposed to relational DBMSs that impose a strict structure to the records (rows) of a table.

MongoDB uses two logging mechanisms: *journaling*, which is the *local log* used to recover from data loss when a single server crashes; and *oplog*, which is the *global log* that ensures data consistency across replicas of a replica set. From time to time a database copy, a *snapshot*, is saved in external storage and the journal logs are discarded, otherwise they would grow indefinitely.

In relation to journaling, MongoDB uses a local log called journal to recover a single server that failed unexpectedly. Every time MongoDB is about to write to disk it first logs the

write to a journal file. The journal is then used to recover a single replica that failed without intervention by other servers. The journal file contains non-human readable, binary, data so it is hard to process.

Oplog is the global log used by MongoDB. The primary server uses it to log every operation that changes the database. The oplog collection has limited capacity, so MongoDB removes older log entries. The oplog is stored in a (MongoDB) database, not in a file as logs usually are.

B. NoSQL Undo with MongoDB

The integration of NoSQL Undo with MongoDB is pretty straightforward and follows the architecture in Figure 3. We installed the Global Log Backup Service in the same network of the database. NoSQL Undo accesses both the database and the backup of the log in order to perform recovery and undo operations.

We used two scenarios in the experiments. *Scenario 1* corresponds to Figure 3. It is a fully distributed instance of MongoDB that was installed in 10 EC2 machines of *Amazon AWS*. The database is divided in two shards, each one containing 3 servers (one primary and two secondary). The configuration servers are grouped in a replica set since that is how MongoDB uses the configuration servers. Finally there is a single router (unlike the 2 in the figure), which is a MongoDB server that is responsible for redirecting the requests to the correct servers. All the servers have the exact same configuration: t2.small instances with a 1 vCPU and 2GB of memory and running Ubuntu 14.04LTS.

We also used a second configuration in our experimental evaluation for diversity: *Scenario 2*. In that scenario NoSQL Undo was deployed in Google Compute Engine. The deployment was composed by a single replica set with 4 machines: 1 primary and 3 secondaries. Each machine had 1 vCPU and 4GB of memory. The OS used was Debian 8.

V. Experimental Evaluation

The objective of the experimental evaluation was to answer the following questions using the implementation of NoSQL Undo for MongoDB: (1) what is the performance trade-off between Focused Recovery and the Full Recovery mechanism? (2) How long does it take to undo different numbers of operations? (3) How does the number of versions of a file affects the time to recover?

To inject realistic workloads we used YCSB [13]. YCSB is a framework to evaluate the performance of different DBMSs using realistic workloads. Some examples of DBMSs supported by YCSB are MongoDB, Cassandra [3], Couchbase [6], DynamoDB [7], and Hadoop HBase [5]. We choose this framework because it is widely adopted for benchmarking NoSQL DBMSs, it provides realistic workloads, and has several configuration options (number of operations, amount of records to be inserted, distribution between reads and writes).

The YCSB workloads used in the experiments were: (A) update heavy, composed by 50% reads and 50% writes; (B) read mostly, with 95% reads and 5% writes; (C) only read, without write operations; (D) new records inserted and the most read, simulating social networks and forums where users consult the most recent records; (E) short ranges, where read operations fetch a short range of records at a time, like a conversation application, blogs and forums; and (F) records updated right after read, simulating social networks when users update their profile.

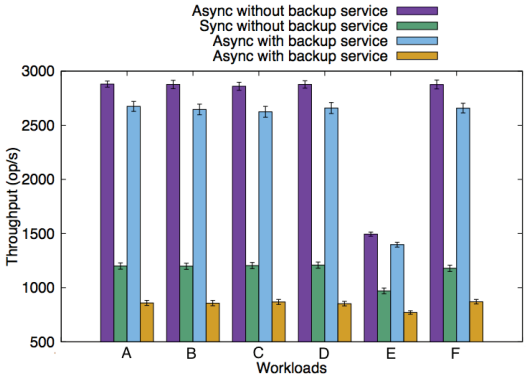


Fig. 4: Overhead of using the Global Log Backup Service with a confidence level of 95%.

MongoDB offers two different interfaces: *synchronous*, where only one operation can be submitted at a time, and *asynchronous*, where operations can be executed in parallel.

A. Global Log Backup Service Overhead

The Global Log Backup Service runs alongside with the database. It listens for changes in the log, so when an operation is executed in the database server, the Global Log Backup Service is notified and saves the operation in external storage. To evaluate the throughput overhead of using this backup service we executed several workloads of YCSB 10 times each using Scenario 2.

Figure 4 presents the average throughput (operations per second) of 10 executions of several workloads of YCSB using both the asynchronous and the synchronous driver with a confidence level of 95%. The cost of having this additional service varies from 6% to 8% when using the asynchronous driver, and from 20% to 30% when using the synchronous driver. The overall throughput seems acceptable given the advantages of storing every executed operation in the database.

In terms of storage, after executing every workload of YCSB the database occupied 100MBs in disk while the global log backup occupied 120MBs. The overhead in terms of storage is considerable (120%), but this is an unavoidable cost of supporting state recovery.

B. Focused Recovery versus Full Recovery

To evaluate how Focused Recovery performs in comparison with Full Recovery a set of operations were undone from the database using both algorithms in Scenario 1. The size of this set varied from 1 to 10,000. Each case was repeated 10 times. The goal of using different sets of incorrect operations to undo was to understand how both algorithms perform when they

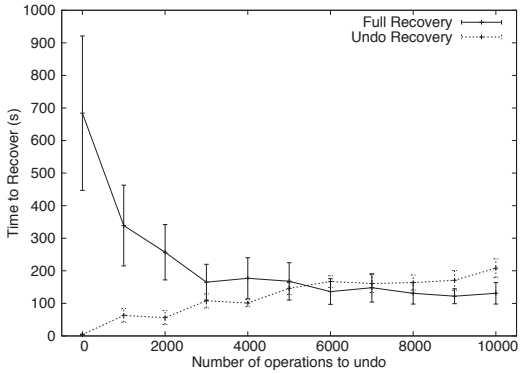


Fig. 5: Focused Recovery performance vs Full Recovery time with a confidence level of 95%.

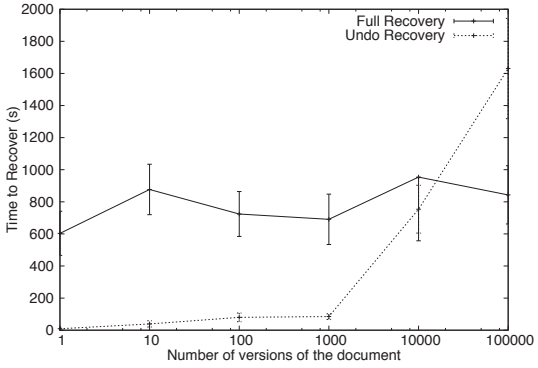


Fig. 6: Focused and Full recovery a document with different versions with 95% confidence level.

undo from just a few operations to almost every operation in the database.

Figure 5 shows the average time to recover using both methods, with a confidence level of 95%. The Full Recovery method performs better as the number of operations to remove increases, which makes sense as less operations have to be executed. On the other hand, the Focused Recovery method performs a lot better when there are just a few operations to be removed and it degrades the performance linearly as the number of operations increases. Focused Recovery takes almost a second to remove 1 operation, whereas Full Recovery takes around 700 seconds. Both methods achieve a similar performance around 5,000 operations to be removed. This result shows that for a small number of operations to be removed Focused Recovery is a better choice, but if more than 60% of the operations are incorrect than the Full Recovery method should be used.

C. Recovery with Different Versions

The focused recovery method reconstructs every document affected by incorrect operations, meaning that if a document has a thousand versions and one of them is incorrect, then the focused recovery method needs to re-execute the remaining 999 operations in order to reconstruct the document correctly. To evaluate how focused recovery is able to remove incorrect operations in documents with different number of versions, we executed both the focused and the full recovery methods in a document varying the number of versions from 1 to 100,000 in steps of 10. Each recovery execution was repeated 10 times. These experiments were conducted in Scenario 1. The average recovery time of each execution can be seen in Figure 6. The time to recover increases exponentially for the Focused Recovery, while it remains almost constant for the Full Recovery. This result was expected since the Full Recovery needs to undo one incorrect operations in every case. Focused Recovery has more work to do if the number of versions of a document increases. This is because it needs to reconstruct the affected record.

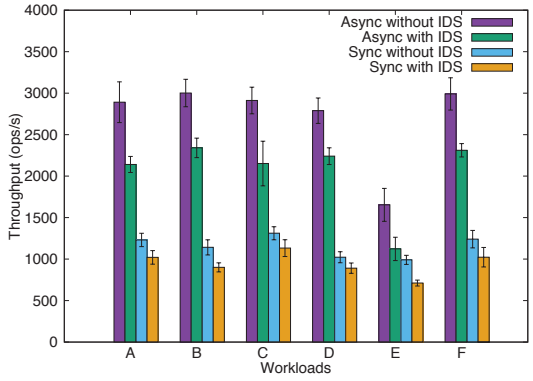


Fig. 7: Overhead of using Snort to detect incorrect operations in a MongoDB Cluster.

D. Intrusion Detection Overhead

Using an IDS to tag incorrect operations facilitates the database administrator job. When an alarm is triggered the administrator consults the IDS log and can immediately undo incorrect operations, instead of browsing the entire database log for incorrect operations. To evaluate the cost of using an IDS to detect incorrect operations, we set up an extra machine (with the same characteristics of the others) running Snort in Scenario 1. We then added 10 rules to Snort and executed every YCSB workload.

Figure 7 shows the overhead of using Snort. The throughput is degraded by 10 to 30%. It is a considerable cost given the benefits of allowing the database administrator to recover a database immediately without loosing time searching in the database log for incorrect operations.

VI. Related Work

The use of logs and snapshots to recover databases after a crash is far from new and is covered in textbooks in the area, e.g., [24]. This work follows a more recent line of work on recovering databases [11], [25], operating systems [12], web applications [15] and other services [10] by eliminating the effect of undesirable operations, but not of the rest of the operations. We discuss some of these works next.

Operator Undo seems to be the first work in this line [10]. It is an architecture that aims to provide administrators the ability to undo operations. The authors argue it is generic, but it has been applied specifically to email servers. To remove the effect of an operation –a verb– from the state, the system is rollback to the moment immediately before that verb (Rewind), the verb is removed (Repair), then every entry in the log after that point is re-executed until the present moment (Replay).

ITDB is a self-healing database built on top of a commercial DBMS [25]. It detects intrusions and is able to isolate the effect of attacks. It also provides recovery mechanisms that repair the effects of intrusions in useful time. ITDB provides a repair mechanism that removes the effect of intrusions similarly to NoSQL Undo.

Phoenix is another recovery system for databases [11]. The operations executed on a database may depend on each other, e.g., a write operation may modify record a using values read from record b. Phoenix tracks these dependencies and considers them during recovery. It consists in a PostgreSQL-based database that gathers record dependencies while logging requests.

Shuttle is a recent recovery system form applications deployed in Platform-as-a-Service clouds [15]. It combines the use of snapshots with selective re-execution of log operations to recover a web application and undo the effects of intrusions. It considers the existence of more than one server (application servers) and a back-end database, unlike the previous systems.

NoSQL Undo is inspired in these systems but has several crucial differences. Firstly, it is the first to support a distributed DBMS, that can be replicated for fault tolerance (replica set) and performance (sharding), where most of the other systems consider a single server. Secondly, it is the only system of the kind that does not require a proxy or modifications to the service to support recovery (MongoDB in this case). Instead, it takes advantage of the built-in log and uses an external component to guarantee that no log operations are lost. Thirdly, except for Shuttle it is the only system that supports two modes of recovery and NoSQL databases.

VII. Conclusion

This paper presents a tool that allows the database administrator to remove incorrect operations from a MongoDB database. It runs as a client of the DBMS and uses its built-in log and snapshots to do recovery. The tool provides two different approaches to recover a database: Focused and Full Recovery. Both methods are capable of recovering databases, but there is a trade-off between performance and number of operations to undo.

Acknowledgements This work was supported by the European Commission through project H2020-653884 (SafeCloud) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

[1] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[2] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 2013, pp. 15–19.

[3] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[4] "MongoDB." [Online]. Available: http://www.mongodb.org

[5] T. White, *Hadoop: The Definitive Guide*. O'Reilly, 2009.

[6] M. Brown, *Getting Started with Couchbase Server*. O'Reilly, 2012.

[7] S. Sivasubramanian, "Amazon DynamoDB: a seamlessly scalable non-relational database service," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012, pp. 729–730.

[8] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, p. 4, 2008.

[9] M. Vora, "Hadoop-HBase for large-scale data," in *Proceedings of the 2011 IEEE International Conference on Computer Science and Network Technology*, 2011, pp. 601–605.

[10] A. Brown and D. Patterson, "Undo for operators: Building an undoable e-mail store," in *Proceedings of the USENIX Annual Technical Conference*, 2003, pp. 1–14.

[11] T. Chiueh and D. Pilania, "Design, implementation, and evaluation of a repairable database management system," in *Proceedings of the 21st IEEE International Conference on Data Engineering*, 2005, pp. 1024–1035.

[12] T. Kim, X. Wang, N. Zeldovich, and M. Kaashoek, "Intrusion recovery using selective re-execution," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010, pp. 89–104.

[13] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, pp. 143–154.

[14] P. Hunt, M. Konar, F. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX Annual Technical Conference*, vol. 8, 2010, p. 9.

[15] D. Nascimento and M. Correia, "Shuttle: Intrusion recovery for PaaS," in *Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems*, 2015, pp. 653–663.

[16] OWASP, "Testing for NoSQL injection," https://www.owasp.org/index.php/Testing_for_NoSQL_injection.

[17] J. Scambray, V. Lui, and C. Sima, *Hacking Exposed Web Applications: Web Application Security Secrets and Solutions*. Mc Graw Hill, 2011.

[18] M. Roesch, "Snort: Lightweight intrusion detection for networks." in *Proceedings of LISA'99: 13th Systems Administration Conference*, 1999, pp. 229–238.

[19] S. Lee, W. Low, and P. Wong, "Learning fingerprints for a database intrusion detection system," in *Computer Security – ESORICS 2002*. Springer, 2002, pp. 264–279.

[20] Y. Hu and B. Panda, "A data mining approach for database intrusion detection," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, 2004, pp. 711–716.

[21] ——, "Identification of malicious transactions in database systems," in *Proceedings of the 7th International Database Engineering and Applications Symposium*, 2003, pp. 329–335.

[22] K. Chodorow, *MongoDB: The Definitive Guide*. O'Reilly, 2013.

[23] "MongoDB Manual." [Online]. Available: https://docs.mongodb.org/manual/

[24] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*, 2nd ed. Pearson, 2008.

[25] P. Liu, J. Jing, P. Luenam, Y. Wang, L. Li, and S. Ingsriswang, "The design and implementation of a self-healing database system," *Journal of Intelligent Information Systems*, vol. 23, no. 3, pp. 247–269, 2004.

Exploiting Universal Redundancy

Ali Shoker
HASLab, INESC TEC & University of Minho
Braga, Portugal
Email: ali.shoker@inesctec.pt

***Abstract*—Fault tolerance is essential for building reliable services; however, it comes at the price of redundancy, mainly the "replication factor" and "diversity". With the increasing reliance on Internet-based services, more machines (mainly servers) are needed to scale out, multiplied with the extra expense of replication. This paper revisits the very fundamentals of fault tolerance and presents "artificial redundancy": a formal generalization of "exact copy" redundancy in which new sources of redundancy are exploited to build fault tolerant systems. On this concept, we show how to build "artificial replication" and design "artificial fault tolerance" (AFT). We discuss the properties of these new techniques showing that AFT extends current fault tolerant approaches to use other forms of redundancy aiming at reduced cost and high diversity[1].**

***Index Terms*—Artificial fault tolerance, redundancy, replication.**

I. Introduction

Fault tolerance (FT) is the central pillar of reliable services [1], [2]. A fault tolerant system must employ some form of redundancy in space or time [2]. *Redundancy in space* is widely used nowadys through consensus protocols and fault detectors [1], [3], [4]. Unfortunately, these techniques are costly and their effectiveness are sometimes questionable due to the *replication factor* and *diversity*.

In particular, in order to tolerate f faults, a FT protocol requires a minimum number of redundant components (i.e., replicas), called the *replication factor*, which is often $2f + 1$ or more [5]–[7]. On the other hand, FT protocols assume independence of failures between replicas. This is usually mitigated by introducing some software and hardware *diversity* in the replicated components on different axes and levels [8]–[11]. Although these approaches improve the reliability of systems through diversity, they are costly and not always effective [9], [11], [12]. For instance, *N-version programming* [1], [9] introduces diversity through coding multiple versions by independent teams and programming languages which is very costly and not always effective since versions originate from a common specification [9], [12]. Other approaches like *proactive recovery* between diverse obfuscated components (generated to be semantically equivalent using a secret key) [10], [13] are only effective in transient failures and when the key is kept secret [8], [14]. *BASE* [11] introduces design diversity using different *Components Off-The-Shelf* (COTS) that have similar behavior, and then uses software wrappers to mimic the state machine behavior. This approach is however limited to the existence of COTS in various programming languages.

In this paper, we revisit the very fundamentals of fault tolerance and introduce *artificial redundancy* considering the "redundant information" inferred through the "action on a component" rather than the "component" itself: a component is artificially redundant to another one if there is a strong correlation between them, even if they are non similar in behavior or semantics. For example, two always opposite buffers $A = -B$ are artificially redundant since there is a perfect (though negative) correlation between them; whereas, "the presence of ice" is artificially redundant, with some *uncertainty*, to "the atmospheric temperature is low" since they are strongly (but not perfectly) correlated.

Artificial replication can then be achieved by making an artificially redundant component an artificial replica, *artira* for short. The idea is to wrap the component by an *adapter* to code (resp., decode) the input (resp., output) of an artira as needed using component-specific (mathematical or probabilistic) transformation functions. Adapters are similar to the *conformance wrappers* used in BASE [11]; however, we apply it to completely independent, but correlated, components instead of those of similar behaviors allowing for some uncertainty (if needed). *Artificial fault tolerance* (AFT) is therefore achieved using replicas and artira, e.g., using voting or agreement, in a similar fashion to current FT protocols. When artira are perfectly correlated, existing FT protocols can be used with higher reliability due to the increased diversity of artiras. On the other hand, if artiras include some uncertainty (bounded or unbounded), new variants of AFT protocols are needed as we show in Section III.

Our approach has many benefits: (1) it exploits new forms of redundancy to reduce the cost of replication; (2) it achieves equivalent or better tolerance to faults than classical FT being built on highly diverse components in terms of behavior, providers, location, etc.; and (3) it makes it possible to achieve lower levels of fault tolerance, e.g., detection, if some uncertainty is accepted by the application and when extra "exact copy" replicas do not exist or are not affordable. This is not uncommon as uncertainty in fault tolerance do exist in practice in areas like *automotive systems*, *clock synchronization*, and *Byzantine approximate agreement* [15]–[17]. We believe that these concepts can be generalized to a wider spectrum of

[1]Project "TEC4Growth - Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER-000020" is financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).

distributed applications and services where even new forms of redundancy can be exploited as we do here. We discuss the feasibility of our approach in Section II-E we explain how AFT can be applied to a large span of applications as in webservices, multithreading, HPC, etc., in the accompanying report [18].

II. Artificial Redundancy and Replication

A. Notations

Consider a component X that is associated with a set of possible actions in A. In general, an action can modify X; however, for ease of presentation, we assume that actions are read-only and we explicitly mention *writes* when needed. We denote by $a(x)$ the output of an action $a \in A$ on a state $x \in X$, and by X^a the range of a on any state in X; we read this "X subject to action a". We also assume that (X^a, d) is a *metric space* with a defined distance d. We say that $x \in S$ is *in the neighborhood* of $y \in S$ if there is a distance r such that: $d(x, y) \leq r$.

B. Artificial Redundancy

Definition 1 (Artificial Redundancy)**.** A component X subject to action a (i.e., X^a) is artificially redundant to component Y subject to action b (i.e., Y^b), denoted $X^a \succcurlyeq Y^b$, iff there is a correlation $\zeta \in]0, 1]$ between them.

The above definition is very relaxed as it makes any two components redundant to each other regardless of their behavior or semantics provided that there is a correlation between them. For instance, the atmospheric "temperature forecast" component on action getTemperature() is strongly correlated to the "snow forecast" component on action isFalling(), and hence, they are artificially redundant. Although, artificial redundancy often makes sense when there is strong or perfect correlation (whether +ve or –ve), we do not explicitly mention the *strength* of correlation ζ to keep the definition general to any correlation method, e.g., Pearson, Spearman, Support Vector Machines, etc [19]. Definition 1 is fine-grained to an individual action of a component (e.g., a function in a service API); however, in practice, services may not be equivalent (i.e., have different APIs); consequently, only parts of a service might be artificially redundant; please refer to [18] for more artificial redundancy properties.

C. Artificial Replication

Artificial redundancy remains useless without the ability to transform artificially redundant components to replicas that can be used in practice. We make this possible by introducing *artificial replication*:

Definition 2 (Artificial Replication)**.** A component X^a is said to be an artificial replica (*artira* for short) of Y^b iff there exists $\alpha \in [0, 1]$, $\epsilon \in Y^b$, and a *transformation function* $\mathcal{F} : X^a \longrightarrow Y^b$ such that $\forall y \in Y^b, \exists x \in X^a$ such that $P\left(d(\hat{x}, y) < \epsilon\right) \geq \alpha$, where $\hat{x} = \mathcal{F}(x)$.

Informally, this means that a component X, subject to an action a, is an *artira* of Y, subject to an action b, if we can find a function $\mathcal{F}$ such that for every state $y \in Y^b$ there is a state $x \in X^a$ such that $\mathcal{F}(x)$ is in the neighborhood of y with some accuracy ϵ (i.e., the error is bounded) and certainty α (i.e., the bound is precise). An artira is defined in a triple $(\mathcal{F}, \alpha, \epsilon)$ whose values must be defined a priori. Notice that, X is an artira of Y means that Y is a reference replica and need not to be an artira of X. In principle, $\mathcal{F}$ is used to transform the output $a(x)$ to a value $\hat{x} = \mathcal{F}(a(x)) \in Y^b$ such that $\hat{x}$ is close, with distance ϵ, to some $y \in Y^b$ with certainty α. The two metrics α and ϵ are strongly related and should be adjusted together: $\epsilon = 0$ reflects 100% accuracy of $\mathcal{F}$ whereas α tells if this is correct all the time. Increasing ϵ makes the accuracy of $\mathcal{F}$ lower but with better certainty α. We show in Section II-E how tuning α and ϵ can bring interesting benefits.

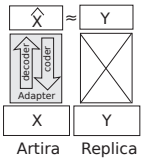


Fig. 1: Artira vs. Replica.

D. Building an Artira

Building an artira X^a of an existing replica Y^b starts by defining the accepted accuracy ϵ and certainty α by the application. If some strong correlation between X^a and Y^b exists X^a can likely be an artira of Y^b; this is possible if a transformation $\mathcal{F}$ can be defined with some accuracy ϵ' and certainty α'. ϵ' and α' can then be adjusted (by incrementing ϵ') to get a higher certainty α'. Finally, X^a is accepted as an artira of Y^b with the triple $(\mathcal{F}, \alpha', \epsilon')$ if: $\alpha' \geq \alpha$ and $\epsilon' \leq \epsilon$.

The transformation logic is then implemented in a wrapper on top of X^a, called *adapter*. Fig. 1 shows the architecture of an artira with an adapter versus a replica. An adapter can be state-full or stateless as *conformance wrappers* which were explained thoroughly in BASE [11], and therefore we skip this discussion here. *Read* operations use a *decoder* that implements $\mathcal{F}$ to transform outgoing values from the artira. *Update* operations, however, use a *coder* to write into the artira, which requires an inverse function $\mathcal{F}^{-1}$ to be defined. In this case, the parameters ϵ' and α' must be adjusted to consider the uncertainty of $\mathcal{F}^{-1}$ if it is not a "perfect" inverse of $\mathcal{F}$ (since a read value will be affected twice by the uncertainty of both $\mathcal{F}$ and $\mathcal{F}^{-1}$). However, this is not required when $\mathcal{F}^{-1}$ is a perfect inverse (e.g., mathematical inverse function) of $\mathcal{F}$, since a value will be read exactly as it was previously written via the adapter (e.g., if $\mathcal{F}(x) = 1/x$ and $\mathcal{F}^{-1}(x) = 1/x$, then $\mathcal{F}(\mathcal{F}^{-1}(x)) = x$). A reasonable cost must be paid while building an artira as discussed in the extended version [18].

E. Artificial Redundancy and Replication Models

We discuss the different artificial redundancy and replication models and their theoretical feasibility by considering a simple abstraction A_i in Table I. More complex abstractions can intuitively be built on top of it, but this is enough to serve

TABLE I: An abstraction of a simple component and the relation between two possible components.

Component:	A_i
val:	a value of any type.
expose(val):	a read-only function that exposes the value of val.
modify(val):	an update function that modifies the value of val.
Relation	A_i and A_j
τ:	Correlation threshold above which $\zeta \geq \tau$ is accepted.
$\mathcal{T}(\mathsf{val}_i, \mathsf{val}_j)$:	a relation to define a transformation function $\mathcal{F}$.

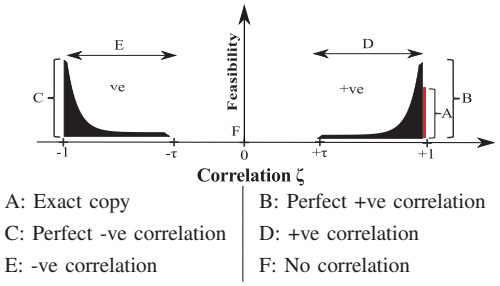


Fig. 2: Feasibility space of artificial redundancy.

for explanation. A_i is composed of a single value val that represents the state of A_i; whereas expose and modify represent the read and write actions, respectively, that are accessible by any other abstraction A_j (which may have different val type and actions). We also represent the relation between A_i and A_j by the correlation threshold τ and the relation $\mathcal{T}$, where τ is the minimum correlation coefficient above which (inclusive) a service accepts components to be artificially redundant, whereas $\mathcal{T}$ materializes the correlation between two components by defining a transformation function $\mathcal{F}$ that may comprise some uncertainty as described. Based on this, we distinguish between interesting artificial replication and redundancy models summarized in Table II.

For ease of presentation, we explain the different models and feasibility with the help of an "imaginary" feasibility spectrum depicted in Fig. 2. Since the ultimate goal is to build fault tolerant systems, which is often the basic defense layer in a service, critical services are likely to adopt very strong correlation (e.g., τ is close to 1) and, gradually, fewer ones accept lower correlation coefficients. Therefore, our conjecture argues that the number of applications decreases (resp., increases) *exponentially* to τ (resp., $-\tau$) as the correlation coefficient ζ approaches zero. Notice that, theoretically, τ can be close to zero; however, it is merely meaningful only when $\tau > 0.5$, i.e., when a strong correlation exists. Now, we discuss the different models.

F. Perfect Artificial Redundancy and Replication (PAR)

Perfect artificial redundancy refers to the case in which application FT requirements only accept perfect positive correlations between components, i.e., $\tau = \pm 1$; meaning that the information inferred by one component through the adapter is the exact information of the other with zero error. Consequently, PAR is the most interesting and desirable model to achieve fault tolerance. The feasibility is depicted in locations A, B, and C on Fig. 2. This is mapped to Table II. A refers to EC case which is the unique acceptable case in current FT. Artificial redundancy expands this case to use other redundancy sources as in PC+ (corr., B) and PC– (corr., C) with the same confidence as if they were exact replicas, as in A. From the perspective of artificial replication, this case refers to the configuration: $(\mathcal{F}, \alpha = 1, \epsilon = 0)$. As shown in the use-cases of settings EC, PC+, and PC–, the function $\mathcal{F}$ transforms val_i to val_j without any error ($\epsilon = 0$) and with 100% certainty ($\alpha = 1$).

G. Strong Artificial Redundancy and Replication (SAR)

Strong artificial redundancy (SAR) refers to the case in which a small bounded error is tolerable as in BSC case in Table II. Though SAR is weaker than PAR, it is useful for some applications to avoid high costs of exact copies when high certainty is acceptable. Of course, such applications are much fewer than those of PAR; however, they do exist in practice as we explain in [18]. In Fig. 2, this is depicted in the gradient color regions D and E. The dense color indicates more applications, showing that the more interesting cases are those when τ is closer to 1. In general, artificial replication is represented by $(\mathcal{F}, \alpha \neq 1, \epsilon \neq 0)$ in SAR case; however, the parameters α and ϵ can be tuned since the inaccuracy is bounded. Thus, it may be suitable to increase ϵ so that a greater certainty $\alpha = 1$ can be achieved and thus SAR becomes $(\mathcal{F}, \alpha = 1, \epsilon \neq 0)$. To explain this, consider the use-case in BSC settings in Table II. In this case, the medical instruments A_i and A_j can infer slightly different cardiac pulse val that is bounded by δ. Then, setting $\epsilon := \delta$ such that $\alpha = 1$ can be a good choice to get high certainty. This actually means that, the artificial replication is 100% accurate with an allowed error of δ cardiac pulses. We show how this is useful in Section III.

III. Artificial Fault Tolerance (AFT)

Artificial fault tolerance (AFT) is the approach used to achieve fault tolerance in a system where at least one artira is used. Without loss of generality, we only address consensus protocols that are the dominant FT protocols used in practice. In particular, we draw an analogy to show how current FT protocols can be adjusted to support artiras, which are the building blocks of AFT protocols. Given the size limits, pedantic details and more fault models can be found in [18].

a) Recalling FT Protocols.: Consider a system of n nodes (e.g., replicas) where f of them can be faulty (regardless of the fault model). To ensure correctness, consensus (or agreement) between nodes must be achieved. To ensure correct Write and Read requests, the intersection of a Read quorum and a Write quorum must be correct (non-faulty). A common approach is to choose the quorum q to be the majority of nodes (also called majority consensus), e.g., $\frac{n}{2} + 1$. An FT protocol is often designed in three main phases: *Propose*, *Accept*, and *Learn*.

- Propose: a value is proposed to agree upon.

TABLE II: The different models, settings, an use-cases of artificial replication and based on Table I.

Model	Correlation	Description	Use-case
PAR	EC	Exact Copy.	A replicated process having $\mathcal{T} = \{\mathcal{F}(\mathsf{val}_i) = \mathsf{val}_j\}$.
	PC+	Perfect +ve Corr. $\tau = +1$	Two weather forecast processes where A_i returns temperature in *Celsius* $\mathsf{val}_i = C^\circ$ and A_j in *Fahrenheit* $\mathsf{val}_j = F^\circ$; then $\mathcal{T} = \{\mathcal{F}(\mathsf{val}_i) = (\mathsf{val}_j - 32) \times 5/9\}$.
	PC–	Perfect –ve Corr. $\tau = -1$	Two processes A_i and A_j having a shared buffer or token and using val as a flag; thus $\mathcal{T} = \{\mathcal{F}(\mathsf{val}_i) = -\mathsf{val}_j\}$.
SAR	BSC	Bounded Strong Corr. $\mid \tau \mid \geq 0.5$	Two medical diagnosis instruments: cardiac pulse meter A_i and Electrocardiogram A_j with sensors val_i and val_j (resp.); since both monitor heart activity, val_i and val_j are strongly correlated with some acceptable error e bounded by δ; therefore, $\mathcal{T} = \{\mathcal{F}(\mathsf{val}_i) = \mathsf{val}_j \pm e \mid e \leq \delta\}$.

- Accept: a proposed value is accepted by nodes if a quorum q of nodes agree on it.
- Learn: the learner (often the requester) accepts the request if a quorum q of replies match, and *learns* the matching value.

This notation is analogous to the phases used in the well-known protocols in literature as Paxos and PBFT [4], [20]. We do not discuss message exchange patterns and delivery assumptions of an FT protocol since they are often the same as in AFT protocols (explained next). Committing a request is also protocol-dependent as it can occur in the Accept or Learn phases. The matching logic ftmatch to approve a request by the acceptors and the requester simply requires a quorum q of responses $\mathsf{r_k}$ to be equal:

$$\mathsf{ftmatch} = \mathsf{card}(\mathsf{R_q}) \geq \mathsf{q};\ \mathsf{R_q} = \{\mathsf{i} \neq \mathsf{j} \mid \mathsf{r_i} = \mathsf{r_j}\} \quad (1)$$

Obviously, since all the quorum's responses are equal, the committed value ftvalue by the acceptors, as well as the learned value by the learner (or requester), is a single value which corresponds to any response in the quorum:

$$\mathsf{ftvalue} = \mathsf{rand}(\mathsf{r_i})\ \mathsf{where}\ \mathsf{i} \in \mathsf{R_q} \quad (2)$$

A. Designing AFT Protocols

Designing an AFT protocol starting from a FT protocol is reasonably not hard since the mechanics of the three phases is almost the same. The only sensitive parts are those which require deterministic behavior. Since in AFT at least one node will be an artira, this can incur some inaccuracy in the response returned by the *decoder* (as explained in Section II-C) which can induce indeterminism in some cases. This can require modifications in the three phases depending on the artificial replication model used. In general, the phases in an AFT protocol are defined as follows:

- Propose: one value is proposed to agree upon.
- Accept: *one or more* proposed values are accepted by nodes if a quorum q of nodes agree on them after following some message exchange pattern.
- Learn: the learner (often the requester) accepts the request if a quorum q of replies *match with some uncertainty*, and learns a *chosen value* according to some *policy*.

The uncertainty induced by the artiras require different matching logic to that in Eq. 1 as well since responses may not always be equal. For an AFT model defined by $(\mathcal{F}, \alpha, \epsilon)$, the general matching criteria is as follows:

$$\mathsf{aftmatch} = \mathsf{card}(\mathsf{R'_q}) \geq \mathsf{q};\ \mathsf{R'_q} = \{\mathsf{i} \neq \mathsf{j} \mid \mathsf{P}(\mathsf{d}(\mathsf{r_i}, \mathsf{r_j}) \leq \epsilon) \geq \alpha\} \quad (3)$$

Equation 3 says that if the distance d (defined in the metric space) between two responses is bounded by ϵ with probability α then these responses are considered matching. On the other hand, choosing a value by the requester in AFT follows an application-dependent policy (e.g., priority, mean value, etc.):

$$\mathsf{aftvalue} = \mathsf{policy}(\mathsf{R'_q}) \quad (4)$$

Depending on the artificial replication model (i.e., PAR or SAR), the properties of the system and aftmatch and aftvalue may change. For instance, in benign (non-Byzantine or malicious) fault models, we distinguish between the following cases:

- In the PAR replication model, an AFT protocol has the same design as a FT protocol. This is the most desired case since it is at least as robust as the existing FT case. (Additional robustness follows from the higher diversity of artiras). In PAR, $\epsilon = 0$ and $\alpha = 1$; thus, the adapter's coding/decoding is perfect which makes the artiras deterministic, and equivalent to replicas in behavior. Therefore, the matching logic and the learned value become as follows:

$$\mathsf{aftmatch} = \mathsf{card}(\mathsf{R''_q}) \geq \mathsf{q};\ \mathsf{R''_q} = \{\mathsf{i} \neq \mathsf{j} \mid \mathsf{d}(\mathsf{r_i}, \mathsf{r_j}) = 0\}$$

$$\mathsf{aftvalue} = \mathsf{rand}(\mathsf{r_i})\ \mathsf{where}\ \mathsf{r_i} \in \mathsf{R''_q}$$

Notice that the above equations are exactly equivalent to ftmatch and ftvalue in Eq. 1 and 2. This makes the AFT protocol phases (Propose, Accept, and Learn) exactly the same as those defined the FT case in Section III-0a. Therefore, in PAR replication model, existing FT protocols can be used.

- In the SAR replication model, the AFT system is defined by $\mathcal{F}_\epsilon^\alpha$ where $\epsilon \neq 0$ and $\alpha = 1$. This means that the error induced by the adapter's coding/decoding is bounded by ϵ with probability 1. Consequently, the matching logic becomes as follows:

$$\mathsf{aftmatch} = \mathsf{card}(\mathsf{R_q}) \geq \mathsf{q}; \mathsf{R'''_q} = \{\mathsf{i} \neq \mathsf{j} \mid \mathsf{d}(\mathsf{r_i}, \mathsf{r_j}) \leq \epsilon\}$$

Due to this bounded indeterminism, we distinguish between Read and Write requests. In ***Write requests***, the proposer node, in the Propose phase, proposes a request value req_p. In the Accept phase, a quorum q of nodes accept req_p (regardless

of the messaging patterns); the request is then committed by having all non-faulty nodes execute req_p in the same order. However, since artiras are indeterministic, the local state of artiras can vary upon execution of req_p within the bound defined by ϵ. This does not affect the Learn phase since an ACK is enough to be sent to the requester. The problem is however reduced to a *Vector Consensus* [21] or *Approximate Agreement* [17] problem as we describe in more details in [18]. Executing ***Read requests*** is similar to those of Write requests case. In some protocols, however, the requester directly sends its Read request to all nodes, which reply back with their local values to the client, without passing through the phases of the protocol described above. In this case, the received values can be treated as a vector, and then a value is chosen depending on the policy. A policy is application-dependent. In some cases, it is enough to choose: one value randomly, based on some criteria (like max or min), or even an aggregate value (e.g., sum, $mean$). We show in [18] that these policies are sometimes more interesting than choosing a single value.

IV. Concluding Remarks

This paper introduces a new form of *artificial redundancy* that is based on the correlations among components rather than on exact copies or similar behaviors. This allows to exploit new sorts of redundancy aiming at reducing the cost of replication and improving independence of failures. In this approach, artificial replicas (artiras) can be used as classical replicas when some uncertainty is tolerated by the application. Additional cost must be paid to find the suitable correlated replica and to build the wrapper. However, this cost will only be paid once which remains less costly than using extra replicas. Artificial fault tolerance (AFT) protocols are similar to classical FT protocols when the correlation between an artira and replica is deterministic; however, the diversity induced by the artira can improve independence of failures and lead better reliability. If the correlation is not perfect, the relation between an artira and replica will no longer be deterministic, but statistical. If the correlation is strong enough, classical FT protocols will need some modifications to take this inaccuracy into consideration.

We argue that this model can be applied in situations where different components are likely to correlate. For instance, the leading Web API directory in [22] shows that dozens of web-services exist in each API category (e.g., currency, weather, dictionaries, BigData, etc.). Given this, it would be interesting to exploit these redundant sources and use them as PAR artiras to design other more reliable services. On a lower level, this approach can also be applied in distributed programming as in Erlang which allows processes to monitor each other for error handling [23]. A similar application can be observed in High Performance Computing when different processes are strongly correlated. A simple example is the multiplication of huge matrices in which Map processes are assigned parts of a matrix like rows/columns/blocks. If there are patterns in the matrix (e.g., sorting), it is not difficult to detect a failure of a Map process if the values emitted by the adjacent Map processes are captured. (Please refer to the extended technical report [18] for more details.) Finally, we believe that it is interesting to derive an empirically study on the feasibility of this approach and the tradeoffs between FT and AFT in terms of fault tolerance, efficiency, and cost in the future.

References

[1] J. Gray and D. P. Siewiorek, "High-availability computer systems," *Computer*, vol. 24, no. 9, pp. 39–48, Sep. 1991.

[2] F. C. Gärtner, "Fundamentals of fault-tolerant distributed computing in asynchronous environments," *ACM Comput. Surv.*, vol. 31, no. 1, pp. 1–26, Mar. 1999.

[3] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, Mar. 1996.

[4] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998.

[5] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: An engineering perspective," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '07. New York,NY,USA: ACM, 2007, pp. 398–407.

[6] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 45–58, Oct. 2007.

[7] Jean-Paul Bahsoun, Rachid Guerraoui, and Ali Shoker, "Making BFT Protocols Really Adaptive," in *In the Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS'15. IEEE-CS, May 2015.

[8] R. R. Obelheiro, A. N. Bessani, L. C. Lung, and M. Correia, "How practical are intrusion-tolerant distributed systems?" 2006.

[9] L. Chen and A. Avizienis, "N-version programming: A fault-tolerance approach to reliability of software operation," in *Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing*, 1978, pp. 3–9.

[10] T. Roeder and F. B. Schneider, "Proactive obfuscation," *ACM Trans. Comput. Syst.*, vol. 28, no. 2, pp. 4:1–4:54, Jul. 2010.

[11] M. Castro, R. Rodrigues, and B. Liskov, "Base: Using abstraction to improve fault tolerance," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 236–269, Aug. 2003.

[12] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Trans. Softw. Eng.*, vol. 12, no. 1, pp. 96–109, Jan. 1986.

[13] F. B. Schneider and L. Zhou, "Distributed trust: Supporting fault-tolerance and attack-tolerance," Cornell University, Tech. Rep., 2004.

[14] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser, "N-variant systems: A secretless framework for security through diversity," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, ser. USENIX-SS'06. Berkeley,CA,USA: USENIX Association, 2006.

[15] W. Ren and R. W. Beard, *Distributed consensus in multi-vehicle cooperative control*. Springer, 2008.

[16] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of the ACM (JACM)*, vol. 32, no. 1, pp. 52–78, 1985.

[17] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl, "Reaching approximate agreement in the presence of faults," *J. ACM*, vol. 33, no. 3, pp. 499–516, May 1986.

[18] A. Shoker, "Exploiting universal redundancy," CoRR, Tech. Rep., September 2016. [Online]. Available: https://arxiv.org/corr/home

[19] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[20] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. Berkeley,CA,USA: USENIX Association, 1999, pp. 173–186.

[21] M. Correia, N. F. Neves, and P. Veríssimo, "From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures," *The Computer Journal*, vol. 49, no. 1, pp. 82–96, 2006.

[22] ProgrammableWeb, "Programmableweb website," 2015. [Online]. Available: http://www.programmableweb.com

[23] F. Cesarini and S. Thompson, *Erlang programming*. " O'Reilly Media, Inc.", 2009.

Towards Designing Reliable Messaging Patterns

Naghmeh Ivaki, Nuno Laranjeiro, Filipe Araujo
CISUC, Department of Informatics Engineering
University of Coimbra, Portugal
naghmeh@dei.uc.pt, cnl@dei.uc.pt, filipius@uc.pt

***Abstract*—Reliable communication is nowadays pervasively supported by TCP, which is poorly adapted for message-based communications, because it offers a streaming channel with no mechanisms to encapsulate messages. Moreover, TCP does not tolerate connection crashes. Thus, whenever reliable message-based communication is needed, developers either use heavy-weight middleware, like Java Message Service (JMS), or develop their own custom error-prone solutions for recovering from crashes. In this paper, we introduce two TCP-based design patterns that address these limitations, and facilitate the development of light-weight and reliable message-based applications. Our design solutions are modular, in the sense that they build on top of each other.**

***Index Terms*—Message-Based Communication, Reliable Communication, Connection Crashes, Design Patterns**

I. Introduction

Reliable messaging lies at the heart of many distributed systems, and is often found in two basic forms: one-way and request-response. One-way messaging, which is the simplest form of communication, is extremely useful in event-based systems, where the information flows in one single direction and does not require any response. Email and chat applications [1], multi-player games [2], social networks [3], group communication systems [4], and complex event processing systems [5] are some of the examples that conceptually need this messaging paradigm.

Over the last few decades, the stream-based Transmission Control Protocol (TCP) [6] has been the most common option for providing reliable communication over the Internet, even for message-based applications. Despite its advantages, TCP has several limitations for reliable message-based communication. TCP is a stream-based protocol, which means that it has no built-in means to place application layer data into an envelope, to be sent and received as a "Message". Also, TCP's reliability guarantees are unfit for one-way messaging, because they do not provide any means for applications to know if application-layer messages are received or processed correctly. Thus, despite being inherently one-way, many applications (e.g., chat) use request-response protocols (e.g., HTTP [10]), to ensure that their messages reach the destination. Using a request-response paradigm in one-way applications, to implement confirmations over TCP, actually changes the interaction pattern from one-way to request-response, which slows down performance due to the waiting time needed for each response. Finally, TCP does not provide means for applications to recover from connection crashes, as they cannot determine which messages did or did not reach the destination, should the TCP connection fail [7].

978-1-5090-3216-7/16/$31.00

A large number of middleware solutions, like Java Message Service [15] or Microsoft MSMQ [16] can overcome these limitations, to provide one-way and other forms of communication. However, these are heavyweight solutions, offering a vast range of services, but targeting very specific platforms and requiring a service provider.

Since reliable communication and message communication are recurring needs in distributed systems, in this paper, we present two messaging design patterns, called "Messenger" and "Reliable Messenger". These patterns should help programmers in the implementation of reliable and very light-weight message-based communication. The Messenger design pattern, in Section II, presents a general design of a message-based application, in which the distributed peers use enveloped messages for communication in a TCP full-duplex channel. This design includes a session-based component that not only sends and receives messages, but also implements the actions required to build (on send) and rebuild (on receive) messages.

The Reliable Messenger, in Section III, extends the Messenger's functionalities, by applying the Connection Handler design pattern [8], to provide the ability for recovering from connection crashes. Thus, besides providing a full-duplex message-oriented communication (implemented in the Messenger), the Reliable Messenger tolerates connection crashes, thus enabling reliable communication across TCP connections.

II. Messenger Design Pattern

In message-based communication, peers exchange data in discrete chunks, known as "messages". A message, which usually includes a header and a body, is placed into an envelope in a predefined format when sent, and it should be exactly the same when read [9]. Since the most popular transport protocols are stream-based (e.g., TCP), it is up to the application layer to determine whether a message has been completely received or not. There are various message-based protocols (e.g., HTTP, SMTP) that define their own messaging format, which besides achieving the functional goal of the protocol, restricts applications to build messages that conform to some format. This format agreement allows the peers to rebuild the messages, after being received, from the stream of bytes.

In this section, we present the Messenger design pattern for synchronous message-based applications. This pattern is

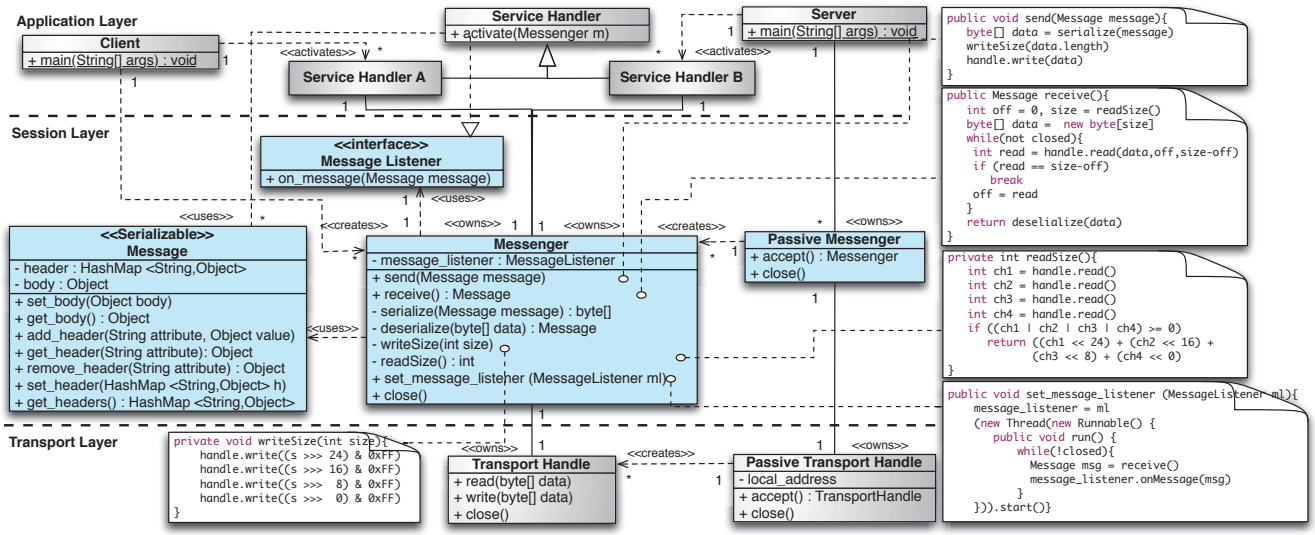


Fig. 1: Messenger Design Pattern for Synchronous Message-Based Applications

independent of the application layer protocol, programming language (as with design patterns in general, it targets object-oriented programming languages), and underlying platform (e.g., operating system).

Figure 1 presents the Messenger design pattern and its partial implementation details as pseudocode. This design includes three layers: application, session and transport. The application layer includes `Service Handlers` to implement the business logic of the application, and a `Client` and a `Server`, to initialize and activate the application peers. The transport layer includes a `Transport Handle` for transmitting byte streams and a `Passive Transport Handle` for receiving connections on the server.

The session layer includes a `Message`, which is a serializable data structure encapsulating application data and any associated meta-data. A message can be interpreted as data, as the description of a command to be invoked, or as the description of an event that occurred (e.g., a mouse click). Each `Message` includes two parts, a header to carry meta-data and a body to carry data. The header of a message contains meta-data about the message (e.g., identifier, size) and any information required for communication, many times depending on the protocol used between the application peers. This information is stored into a structure comprised of various fields and their corresponding values. While the header can be used by the application and session layer, the body contains the application's data and is ignored by the session layer.

The `Messenger` is dedicated to take the necessary actions for sending and receiving the application's messages through the `Transport Handle`. Indeed, the `Messenger` is responsible for sending a `Message` as an array of bytes through the stream-based `Transport Handle`, and also for delivering an array of bytes, read from the `Transport Handle`, to the `Service Handlers` as a `Message`.

When a `Message` is given to the `Messenger` through the method *send()*, the messenger converts (or serializes) the message to an array of bytes, writes the size of the message, and then sends the serialized message. On the receiver side, when the method *receive()* is invoked by the application, the receiver reads the size of the incoming message, receives the message completely, deserializes it from an array of bytes to the `Message`, and delivers it to the `Service Handler`.

Messages can also be delivered to the application using a callback method defined in `Message Listener`. To enable the automatic delivery of messages, rather than having explicit requests for read (through the method *receive()*), the `Service Handlers` must implement the method *on_message()* of the `Message Listener` and pass the reference of this method to the `Messenger` through the method *set_message_listener()*. When this happens, the `Messenger` internally dedicates a new thread for reading the messages and delivering them to the service handler through the method *on_message()*.

III. Reliable Messenger Design Pattern

In this section, we advance the Messenger's design to tolerate connection crashes. To this end, the Connection Handler design pattern [8] is used. The resulting design is presented in Figure 2. As shown, the Reliable Messenger extends the functionalities of the Messenger and the `Connection Handler`. Here, we explain how the `Connection Handler` design pattern is incorporated and integrated with the Messenger, to ensure recovery from connection crashes.

The `Connection Handler` provides an interface to implement all actions required to establish a connection and reestablish a failed one. Each `Connection Handler` has a unique server-generated identifier. Peers exchange this identifier in the method *handshake()*, once a connection is successfully created. The actions to reconnect and resend the lost data must be implemented in method *reconnect()*. The most challenging part of the recovery process in the server side is to replace a failed connection with a new one. The `Handlers Synchronizer` solves this problem,

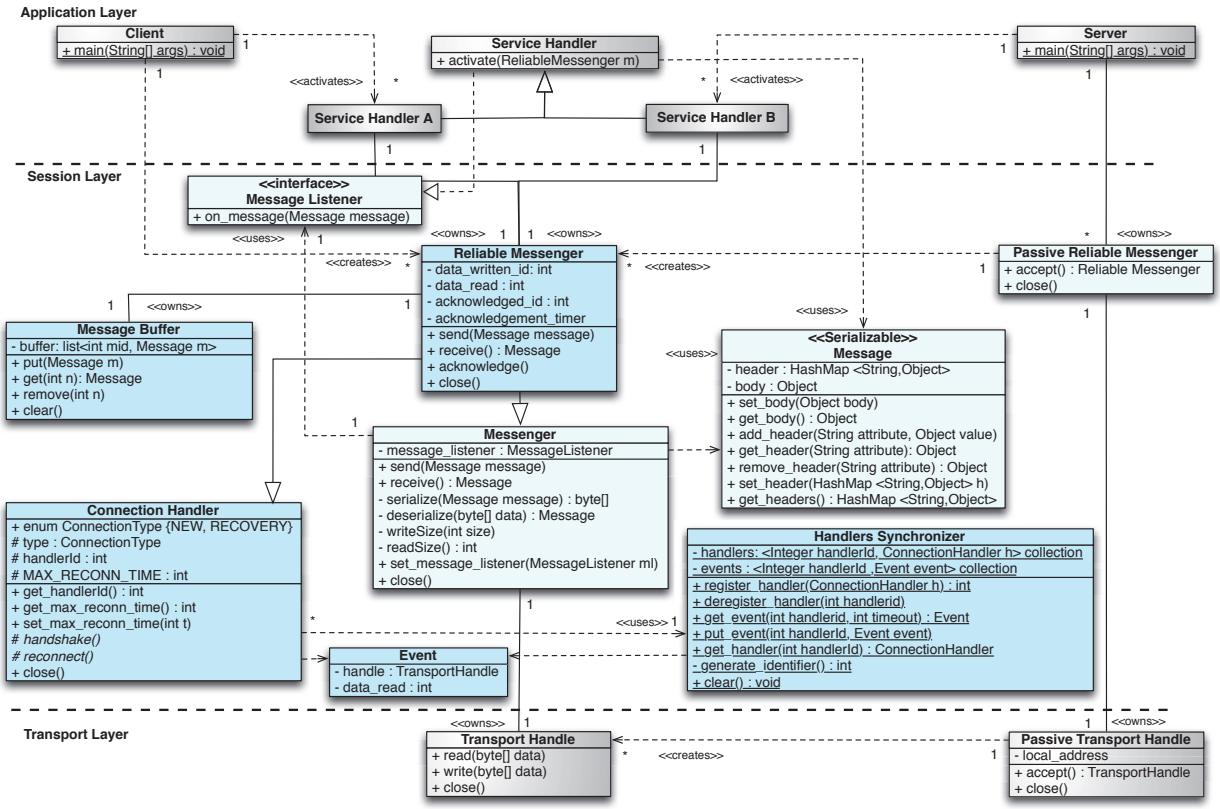


Fig. 2: Reliable Messenger Design Pattern

by allowing `Connection Handlers` to: 1) register and deregister themselves into/from the list of the handlers; 2) put an event for a new handler owning a failed connection; and 3) wait and get the event coming from another connection handler with a recovered connection.

We also need a reliable endpoint, called `Reliable Messenger`, which inherits the properties of the `Connection Handler`, and implements its handshake and reconnection processes. The `Reliable Messenger` also extends the functionalities of the `Messenger`, to enable exchange of `Messages`. Each `Reliable Messenger` owns one `Message Buffer` that implements the `Buffer` interface of the `Connection Handler` design pattern. The `Reliable Messenger` must modify the *send()* and *receive()* operations, to interact with this buffer. Upon creation of a connection and after initialization of both client and server, the `Service Handlers` start exchanging messages, using the `Reliable Messenger`.

Figure 3 presents the recovery process in detail. When a connection fails, both client and server `Reliable Messengers`, try to reconnect by invoking the `reconnect()` method of the `Connection Handler` class.

The initiative to reestablish a connection always belongs to the client's messenger, while the server's messenger (rm_1) waits for a recovery connection to arrive. In order to distinguish new connections from the recovery connections (created for recovery purposes), each connection requires

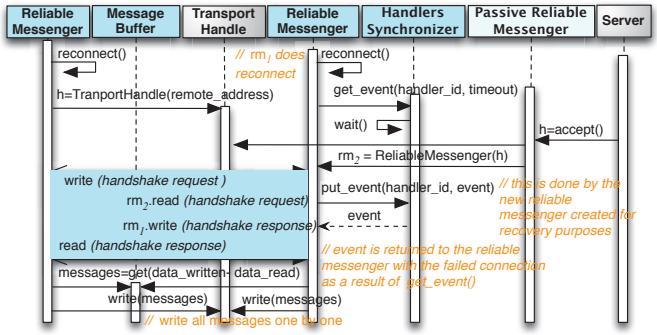


Fig. 3: Recovery from Crashes with the Reliable Messenger

a unique identifier (*handler_id*), which is defined in the `Connection Handler`. To generate this identifier, the `Reliable Messengers` need to perform a handshake before starting the exchange of data messages, when the connection is created for the very first time. When a crash occurs, and after a successful reconnection, the client sends a handshake request, including this identifier, to the server, to let the server know that the connection is created for recovery purposes. In addition to the connection's identifier (*handler_id*), the identifier of the last message received (*data_read*) is also exchanged between the peers through the handshake procedure. This allows the `Reliable Messengers` to re-transmit the messages that were lost in transit due to the connection crash.

The replacement of the failed connection with the new one occurs using the `Handlers Synchronizer`. The `Connection Handler`, created for recovery purposes

(rm_2), asks the `Handlers Synchronizer` to deliver an `Event`, which includes the new transport handle and the id of the last message received, to the appropriate `Connection Handler` (rm_1). Once the failed handle is replaced with the new one, a handshake response is sent back to the client. At this point, both `Reliable Messengers` get the lost messages from the `Message Buffer` and retransmit them.

IV. Related Work

Reliable messaging often works in two ways: either in direct request-response protocols, or using an intermediate message broker. On top of the request-response group, we find the immensely popular HTTP [10]. HTTPR [11] leverages on HTTP, to provide reliable transport of messages between peers, even in the presence of network or endpoint crashes. It uses logging and retransmission, to ensure delivery of each message to the application. CoRAL [12] is another HTTP-based solution that handles server crashes, in web-based services. CoRAL uses connection replication and application-level logging mechanisms. Still in the request-response paradigm, WS-Reliability [13] and WS-ReliableMessaging [14] aim to guarantee delivery of SOAP messages.

The second group of solutions includes the presence of a broker that decouples communication between peers. Java Message Service (JMS) [15] is a messaging standard that provides an API for Java. JMS uses a broker between producers and consumers of messages, to enable loosely-coupled communication. Microsoft MSMQ [16] is another message queuing technology for the Windows operating systems. IBM WebSphere MQ [17] is also a queue-based message oriented middleware that enables reliable messaging, by using acknowledgments, negative acknowledgments, and sequence numbers. Oracle Advanced Queuing [18] is maintained by the Oracle Corporation and integrated into its Oracle database, as a repository to provide message queuing for asynchronous communications.

The Advanced Message Queueing Protocol (AMQP) [19] is also designed to support loosely-coupled and asynchronous communication patterns. While the above queue-based solutions (JMS, MSMQ, WebSphere MQ, and OracleAQ) provide a standard API for a specific platform, AMQP provides a standard messaging protocol across all platforms. ZeroMQ [20] is a high-performance messaging library aimed for scalable distributed applications. It provides a message queue, but unlike message-oriented middleware, a ZeroMQ system can run without a standalone message broker. The library is designed to have a familiar socket-style API.

V. Conclusion

This paper presented design solutions for message-based communication. These solutions target two limitations of TCP for reliable messaging. The Messenger facilitates sending and receiving enveloped messages and is independent of the application-layer protocol. The Reliable Messenger provides transparent recovery, should a connection failure occur. By being reliable, light-weight and independent from the platform, our designs close the gap between heavyweight reliable message-oriented solutions, and many simple applications requiring one-way communication.

Acknowledgment

This work was carried out under the project PTDC/EEI-ESS/1189/2014 — Data Science for Non-Programmers, supported by COMPETE 2020, Portugal 2020-POCI, UE-FEDER and FCT.

References

[1] S. Herring, D. Stein, and T. Virtanen, *Pragmatics of computer-mediated communication*. Walter de Gruyter, 2013, vol. 9.

[2] N. Veljkovic, M. Punt, M. Z. Bjelica, and N. Crvenkovic, "Tv-centric multiplayer gaming over the cloud for consumer electronic devices," in *Third IEEE International Conference on Consumer Electronics (ICCE-Berlin)*, 2013, pp. 1–3.

[3] R. Canning, *Real-Time Web Technologies in the Networked Performance Environment*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library, 2012.

[4] K. P. Birman, "Group communication systems," in *Guide to Reliable Distributed Systems*. Springer, 2012, pp. 369–405.

[5] G. Gharbi, M. B. Alaya, C. Diop, and E. Exposito, "Aoda: an autonomic and ontology-driven architecture for service-oriented and event-driven systems," *International Journal of Collaborative Enterprise*, vol. 3, no. 2/3, pp. 167–188, 2013.

[6] J. Postel, "Rfc793: Transmission control protocol." *Information Sciences Institute*, vol. 27, pp. 123–150, 1981.

[7] N. Ivaki, F. Araujo, and F. Barros, "Session-based fault-tolerant design patterns," in *The 20th International Conference on Parallel and Distributed Systems (ICPADS)*, 2014.

[8] N. Ivaki and F. Araujo, "Connection handler: A design pattern for recovery from connection crashes." [Online]. Available: https://eden.dei.uc.pt/~naghmeh/papers/chdp.pdf

[9] G. Hohpe and B. Woolf, *Enterprise Integration Patterns — Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.

[10] B. Krishnamurthy and J. Rexford, *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*, 1st ed. Addison-Wesley Professional, May 2001.

[11] A. Banks, J. Challenger, P. Clarke, D. Davis, R. King, K. Witting, A. Donoho, T. Holloway, J. Ibbotson, and S. Todd, "Httpr specification," *IBM Software Group.*, vol. 10, 2002.

[12] N. Aghdaie and Y. Tamir, "Coral: A transparent fault-tolerant web service," *Journal of Systems and Software*, vol. 82, no. 1, 2009.

[13] C. Evans, D. Chappell, D. Bunting, G. Tharakan, H. Shimamura, J. Durand, J. Mischkinsky, K. Nihei, K. Iwasa, M. Chapman *et al.*, "Web services reliability (ws-reliability), ver. 1.0," *joint specification by Fujitsu, NEC, Oracle, Sonic Software, and Sun Microsystems*, 2003.

[14] R. Bilorusets, D. Box, L. F. Cabrera, D. Davis, D. Ferguson, C. Ferris, T. Freund, M. A. Hondo, J. Ibbotson, L. Jin *et al.*, "Web services reliable messaging protocol (ws-reliablemessaging)," *Specification, BEA, IBM, Microsoft and TIBCO*, 2005.

[15] M. Richards, R. Monson-Haefel, and D. A. Chappell, *Java message service*. " O'Reilly Media, Inc.", 2009.

[16] S. Horrell, "Microsoft message queue," *Enterprise Middleware*, 1999.

[17] J. Hart, "Web sphere mq: connecting your applications without complex programming," *IBM WebSphere Software White Papers*, 2003.

[18] D. Gawlick, "Messaging/queuing in oracle8," in *IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 1998, pp. 66–66.

[19] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, no. 6, pp. 87–89, 2006.

[20] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. " O'Reilly Media, Inc.", 2013.

Traffic Engineering based on Shortest Path Routing Algorithms for FTV (Free-Viewpoint Television) Applications

Priscila Solis and Henrique Domingues Garcia
Computer Science Department, Universidade de Brasília
Prédio CIC/EST, Campus Darcy Ribeiro, Asa Norte, Caixa Postal 4466 CEP 70910-900, Brasília, Brazil
Email: pris@unb.br, henriquedgarcia@gmail.com

***Abstract*—This paper proposes a traffic engineering methodology based on the self-similarity parameter of video traffic as a restriction on an Integer Linear Programming problem (ILP) for the definition of link costs for routing optimization in the OSPF (Open Shortest Path First) routing protocol. The goal is to optimize video routing considering QoS constraints and to identify the maximum delay and link load parameters for transmission between video sources and middleboxes for FTV (Free Viewpoint Television) applications. The proposal was evaluated in a simulation environment and the results show that the routing optimization procedure based on the self-similar parameter of video traffic for defining link weights, can significantly reduce the delay in the network and allow the identification of more accurate threshold values for link loads and delays for video transmission on the Internet.**

I. Introduction

With the increasing demands of bandwidth and data rates for multimedia applications, it is estimated that until 2018 more than 80% of the traffic on the Internet will be video. Also, the research of the last 20 years has the consensus on the statistic properties of multimedia traffic. which are defined as self-similar or multifractal [1] [2] [3], increases the difficulty to guarantee QoS (Quality of Service) metrics for applications on a best effort based architecture, such as TCP/IP. Several research works have addressed traffic engineering and routing optimization with traditional protocols [4] [5] [6]. Altin [7] presents a survey of the most relevant research related to optimization shortest path protocols. In this scope, Fortz et al.[6] proposed a model of Integer Linear Programming (ILP), which is used as a comparison reference in our research. Later, in [5] implements the heuristics proposed by Fortz in the TOTEM simulator (ToolBox for Traffic Engineering Methods) and Balon in [5] compared several objective functions for engineering traffic, in which the model has the best results. On the other hand, some authors as [4] search for new techniques for solving optimization problems related to the discovery of the best costs, since most of these problems are NP-complete.

When middleboxes are virtualized, they generate an NFV (Network Functions Virtualization) that can be initialized or deactivated as needed, increasing the efficiency and dynamism of the network. This paper proposes a method to optimize routing in a traditional IP network with an architecture based on the use of middleboxes for FTV applications, still in development and standardization by MPEG (Moving Picture Experts Group) [8] [9].

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

Variable	Description
i, j, s, t	Network nodes
N	Set of network nodes
(i, j)	A link with source in node i and destination in node j
A	Set of all network links
$\phi_{(i,j)}$	Link cost (i,j)
(s, t)	source,destination
$\delta_{(s,t)}$	Traffic from s to t
$f^{(s,t)}_{(i,j)}$	Flow from s to t, going though (i,j)
$l_{(i,j)}$	Total traffic load in (i,j)
$c_{(i,j)}$	Link capacity (i,j)

TABLE I
Variables and Notation for the MCNFP.

II. Related Work

A. FTV, Traffic Engineering and Routing Optimization

A FTV (Free viewpoint TeleVision) is a live video application that allows the viewer to choose from a video scene one or more viewing angles of interest, in order to produce a viewing experience closer to reality [9]. A possible improvement, already discussed by Rexford et al.[10] is to select the routing parameters on a wide variety of performance and reliability constraints. But nevertheless, the IP routing protocols have various tunable parameters that may be adjusted to change or optimize the paths the routers use to forward data packets, identifying a a good setting of these parameters in a large network is a challenging and non trivial procedure.The discovery of optimal weights can be treated as an Integer Linear Programming problem (ILP). Sub-optimal solutions can be computed in polynomial time using heuristics such as Branch and Cut, Local Search or Tabu Search [11]. The routing optimization problem, known as MCNFP (Multi-Commodity Network Flow Problem) [12] is described as an ILP problem in Eqs.1 to 5. The notation used in the equations is described in Table I.

$$\text{Min } \Phi = \sum_{(i,j)\in A} \phi_{(i,j)} \tag{1}$$

$$\text{subject to: } \sum_{j \in N} f_{(i,j)}^{(s,t)} - \sum_{j \in N} f_{(j,i)}^{(s,t)} = \begin{cases} \delta(s,t) \text{ if } i = s \\ -\delta(s,t) \text{ if } i = t \\ 0 \text{ otherwise} \\ \\ s,t \in N,\ (i,j) \in A \end{cases} \tag{2}$$

$$l_{(i,j)} = \sum_{(s,t) \in N} f_{(i,j)}^{(s,t)} \tag{3}$$

$$l_{(i,j)} \leq c_{(i,j)} \qquad (i,j) \in A \tag{4}$$

$$\phi \geq \alpha_z l_{(i,j)} - \beta_z c_{(i,j)} \quad (i,j) \in A,\ z \in Z \tag{5}$$

Eq. 1 is the objective function used to minimize the link costs considering the cost function ϕ. Eq. 2 is the continuity restriction that allows that each node that has traffic inputs and outputs (s,t) are in matrix δ. Eq. 3 defines the load as the addition of all flows traversing links (i,j). Eq. 4 limits the link load to its capacity and Eq. 5 is the cost function defined by a set of linear equations. The coefficients α and β are associated to the behavior of the cost function. In our proposal, these coefficients are calculated by a linear regression procedure in specific intervals to optimize the objective function.

B. Traffic Characterization for FTV Appplications

Norros in [13] describes a autossimilar stochastic process called Fractal Envelope Process (FEP) characterized by the Hurst parameter, the standard deviation and the network load when considering the traffic fraction encapsulation of $1 - epsilon$ generated by a fBm (fractional Brownian Motion) process, where $epsilon$ is the packet loss due to buffer size and equals the probability that the fBm exceeds the FEP, which is a practical way to define the upper limit of the queue. Thus, the maximum number of elements in a FEP queue is given by Eq.6, where $k = -2ln\epsilon$ and q_{max} is the maximum queue size. Considering the above concepts, it is possible to estimate the effective data rate of self-similar traffic trace using the maximum queue delay, defined by Eqs. 7 and 8 respectively, as proposed by Fonseca et al.[14]. When the traffic results from the aggregation of several self-similar flows in a link, it is necessary to calculate the resulting H parameter H_g [1].

$$q_{max} = (c - \bar{a})^{\frac{H}{H-1}} (\kappa\sigma)^{\frac{1}{1-H}} (H)^{\frac{H}{1-H}} (1 - H) \tag{6}$$

$$\hat{c} = \bar{a} + K^{\frac{H-1}{H}} (\kappa\sigma)^{\frac{1}{H}} H(1-H)^{\frac{1-H}{H}} \tag{7}$$

$$t_{max} = (\kappa\sigma H c - \bar{a})^{\frac{1}{1-H}} \tag{8}$$

The FTV applications aggregate hundreds or thousands of video flows, being transmitted simultaneously with their depths maps. Since the standard has not been totally defined, we consider in this research that the video flows are coded using H.265. The transmission of FTV applications has not yet a well defined standard and some proposals have been done for this purpose [9]. In Figure 1, based on the proposal of Scandarolli et al.[9], we describe an architecture, in which the

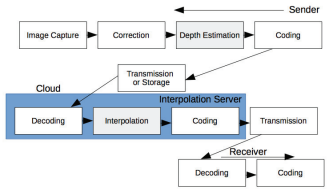


Fig. 1. Adopted FTV Architecture [9]

video transmissor can send all the flows to a middlebox, physically closer to the final user, that will execute the interpolation. This server will transmit to final users that may want to watch different video angles that must be interpolated, coded and delivered using a distribution system such as P2P or DASH. To verify the statistical properties of a FTV application, we used the video trace from *Tears of Steel* in 4K (4096 x 1744) coded with H.265 [15]. We consider that the video flow will be transmitted from the sender to the cloud, in which is the interpolation server and can be implemented in a real network by a middlebox. The mean data rate for this flow is 5,67 Mbps. To simulate an hypothetical FTV application, the resulting flow was configured considering the aggregation of 10 video flows, for a total data rate of 56,7 Mbps. The resulting flow is self-similar, with an *H* parameter of 0.87, which confirms the need to consider this property in the routing optimization procedures.

III. Optimization Model

The results of this work raise the discussion that despite the advantages of the more advanced routing protocols, that simple approach of adjusting the static link weights may remain viable for many operating IP networks. Based on that work, our research developed an enhanced proposal to optimize the cost function, described in Eqs.1 to 5. We propose an ILP system, described by Eq.9, called by LPM-FEP (Linear Program Model - Fractal Envelope Process).

$$\varphi' = \begin{array}{ll} 1 & 0 \leq l/c < 2,5/100 \\ 5 & 2,5/100 \leq l/c < 5/100 \\ 20 & 5/100 \leq l/c < 8,5/100 \\ 150 & 8,5/100 \leq l/c < 15/100 \\ 1300 & 15/100 \leq l/c < 20/100 \\ 17000 & 20/100 \leq l/c < \infty \end{array} \tag{9}$$

$$\begin{array}{lll} \phi_{(i,j)} & \geq & l_{(i,j)} \\ \phi_{(i,j)} & \geq & 5l_{(i,j)}H_{(i,j)} - \frac{1}{10}c_{(i,j)} \\ \phi_{(i,j)} & \geq & 20l_{(i,j)}H_{(i,j)} - \frac{17}{20}c_{(i,j)} \\ \phi_{(i,j)} & \geq & 150l_{(i,j)}H_{(i,j)} - \frac{119}{10}c_{(i,j)} \\ \phi_{(i,j)} & \geq & 1300(i,j)H_{(i,j)} - \frac{922}{5}c_{(i,j)} \\ \phi_{(i,j)} & \geq & 17000l_{(i,j)}H_{(i,j)} - 3325c_{(i,j)} \end{array} \tag{10}$$

To perform the optimization procedures based on the above mentioned models, we propose the framework which is described in the next steps:

- Step 1: Define the maximum queue size using Eq. 6;

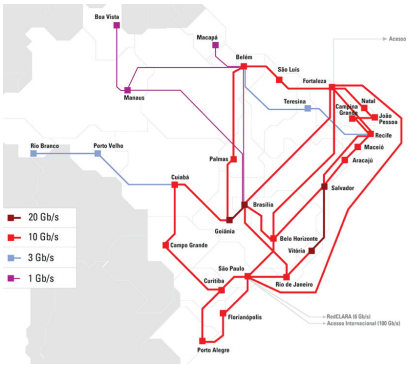


Fig. 2. Network Topology for simulation: Ipe Network

- Step 2: Extract network information : topology and total traffic (background traffic and FTV traffic);
- Step 3: Calculate H_g, σ_g^2 and the effective band for all the network links using 7
- Step 4: Create a traffic matrix with the data collected in step 3;
- Step 5: Solve the PLI model and calculte links weights;
- Step 6: Insert the new weights in the network links and calculate new routes;
- Step 7: Monitor the QoS metrics.

We will validete our proposal through the comparison with the Fortz Model[6] hereby called as LPM-Fortz and the standard Cisco cost model which is a static and the inverse of the link capacity [16], i.e. $\phi_{(i,j)} = 1/c_{i,j}$, hereby called as Invcap.

IV. Experimental Results

To perform the simulation we used the TOTEM (TOolbox for Trafic Engineering Methods),which implements several algorithms for traffic engineering with shortest path protocols [5], which was also used in related previous works [6].The network topology chosen to perform the evaluation was the RNP (Rede Nacional de Pesquisa), available in [17] and described in Figure 2. The network connects 27 cities in Brazil with links varying from 100 Mbps to 20 Gbps. We used in our evaluations the OSPFv2 protocol, as defined in the RFC 2328 [18], which uses the ECMP (Equal-Cost Multi-Path) technique. In our proposal,the evaluated metric is defined as the mean maximum network delay is the ratio between the addition of the maximum delays of all the links and the number of links. This value shows the mean upper limit that a packet waits in the buffer before being forwarded to the next hop or discarded until arriving to the middlebox. Accordingly to Szigeti [19] the mean delay has to be lower than 5 seconds for video transmission. Considering the architecture described in Figure 1, we simulated a FTV transmission between Porto Alegre and Belem (see Fig2), varying from 100 Mbps to 2500 Mbps. The resulting traffic showed an Hg = 0,87, consistent with the results obtained on a real video traffic trace, as shown in Section 3. Also, we defined for all the links a set of background traffic, varying from 5% until 20% of the link capacity. with a random H parameter between 0,5 e 0,85, uniformly distributed. Figure 3 shows the maximum mean network delay versus the traffic load, with a background traffic of 5%. The traffic rate varies between 100 Mbps and 2500 Mbps. As can be seen, with the LPM-FEP was used to optimize the link weights, the delay is lower than the LPM-Fortz, even with higher traffic loads. The limit of 5 seconds was reached with 2300 Mbps of traffic load for our model, 75% more than the load reached with the LPM-Fortz and the Invcap for the same delay, which indicates a better route selection for the different flows. Altogether with the background traffic, this load equals to 28% of the mean network capacity.

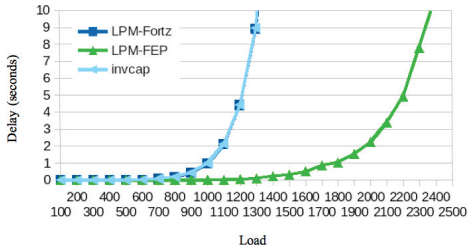


Fig. 3. Maximum Mean Delay, with background traffic equal to 5%

Figure 4 shows the results for background traffic equal to 10%. As can be seen in 4 there is a burst for the maximum mean delay with the LPM-FEP at 200 Mbps, however, suddenly the delay stabilizes and remains lower than the other models for higher loads, with the 5 seconds upper limit between 1100 Mbps and 1200 Mbps, while the LPM-Fortz and Invcap reach that limit at 900 Mbps and 1000 Mbps. The total network load with the background traffic achieves 22% of the average network capacity.

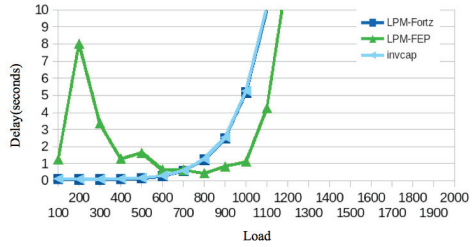


Fig. 4. Maximum Mean Delay with background traffic = 10%

With a load of 15% for background traffic, the results are shown in Figure 5. As can be seen, the LPM-FEP has very good performance for this scenario, provides a more uniform distribution of traffic in the network and the delay grows more softly. The value remains under 5 seconds for all the simulation interval. The FTV application plus the background traffic corresponds to 30% of the average network capacity. In this scenario the Invcap model presents the greatest delays and hits the 5 seconds limit (at 700 Mbps, 53% less than the LPM-FEP and very close to the LPM-Fortz model.

Figure 6 shows the results for background traffic equal to 20 %. The first metric for the FEP-LPM LPM was superior to

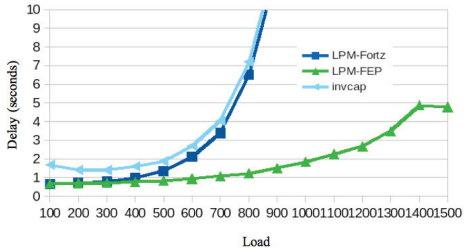


Fig. 5. Maximum Mean Delay with background traffic equal to 15%

the one calculated with the LPM-Fortz model at 400 Mbs, and from this point the delays were inferior for our model when comparing to the other models. However, for this scenario, the delay is higher that the upper limit, reaching more than 10 seconds throughout the whole simulation.

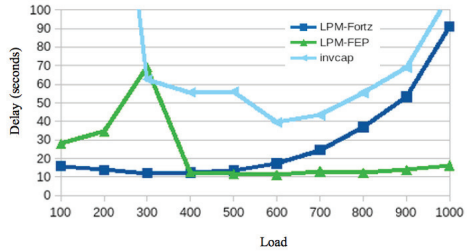


Fig. 6. Maximum Mean Delay with background traffic equal to 20%

The results of the previous scenarios indicate that to keep the average delay below 5 seconds for FTV applications, it is necessary to maintain the average network load below 25%. This result is based on the assumption that video traffic is autossimilar and results in an aggregated self-similar traffic with background traffic from other applications.

V. Conclusions and Future Work

This paper presented a traffic engineering methodology for routing optimization for FTV applications using traditional protocols. The idea considers that, according to other related works, for emerging applications a good setting using traditional IP routing protocols appears to be faster and less costly than deploying or enabling new protocols that may adapt automatically to the prevailing traffic. Using a real operational network topology, the experimental results show that the routing optimization methodology proposed in our work has the potential to reduce the delay compared to other methods.

One important conclusion of our work is that the average network load should not exceed 25% to meet the QoS requirements of live digital video streams on the Internet. This value, although low, is now found in several operational networks. However, the strong increase of applications that consume large bandwidth may change this in the near future and this value may be used as a interesting reference for network managers.The future work of this research will focus on improving the model and extend its evaluation to other application scenarios that may extend the ideas for new routing protocols in the Future Internet.

References

[1] P. Solis Barreto, “Uma metodologia de engenharia de trafego baseada na abordagem auto-similar para a caracterizacao de parametros e a otimizacao de redes multimidia,” Tese, Universidade de Brasilia, 2007.

[2] Z. Sahinoglu and S. Tekinay, “On multimedia networks: self-similar traffic and network performance,” *Communications Magazine, IEEE*, vol. 37, no. 1, pp. 48–52, Jan 1999.

[3] I. Reljin, A. Samčović, and B. Reljin, “H.264/avc video compressed traces: Multifractal and fractal analysis,” *EURASIP J. Appl. Signal Process.*, vol. 2006, pp. 123–123, jan 2006. [Online]. Available: http://dx.doi.org/10.1155/ASP/2006/75217

[4] D. Papadimitriou, B. Fortz, and E. Gorgone, “Lagrangian relaxation for the time-dependent combined network design and routing problem,” in *Communications (ICC), 2015 IEEE International Conference on*, June 2015, pp. 6030–6036.

[5] G. Leduc, H. Abrahamsson, S. Balon *et al.*, “An open source traffic engineering toolbox,” *Comput. Commun.*, vol. 29, no. 5, pp. 593–610, Mar. 2006, Available at `http://totem.run.montefiore.ulg.ac.be/`. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2005.06.010

[6] B. Fortz, J. Rexford, and M. Thorup, “Traffic engineering with traditional ip routing protocols,” *Communications Magazine, IEEE*, vol. 40, no. 10, pp. 118–124, Oct 2002.

[7] A. Altin, B. Fortz, M. Thorup, and H. Umit, “Intra-domain traffic engineering with shortest path routing protocols,” *Annals of Operations Research*, vol. 204, no. 1, pp. 65–95, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10479-012-1270-7

[8] M. Tanimoto, M. Tehrani, T. Fujii, and T. Yendo, “Free-viewpoint TV - a review of the ultimate 3DTV and its related technologies,” *Signal Processing Magazine, IEEE*, vol. 28, no. 1, pp. 67–76, Jan. 2011.

[9] T. Scandarolli, R. de Queiroz, and D. Florencio, “Attention-weighted rate allocation in free-viewpoint television,” *Signal Processing Letters, IEEE*, vol. 20, no. 4, pp. 359–362, Apr. 2013.

[10] J. He, J. Rexford, and M. Chiang, “Design for optimizability: Traffic management of a future internet,” in *Algorithms for Next Generation Networks*, ser. Computer Communications and Networks, G. Cormode and M. Thottan, Eds. Springer London, 2010, pp. 3–18. [Online]. Available: http://dx.doi.org/10.1007/978-1-84882-765-3_1

[11] B. Fortz and D. Papadimitriou, “Branch-and-cut strategies for a multi-period network design and routing problem,” in *Control, Decision and Information Technologies (CoDIT), 2014 International Conference on*, Nov 2014, pp. 128–133.

[12] D. P. Bertsekas and R. G. Gallager, *Data networks*. London: Prentice-Hall International, 1992, 1Ã¨re Ã©dition publiÃ©e en 1987. International student edition en 1992. [Online]. Available: http://opac.inria.fr/record=b1082477

[13] I. Norros, “A storage model with self-similar input,” *Queueing Systems*, vol. 16, no. 3-4, pp. 387–396, 1994. [Online]. Available: http://dx.doi.org/10.1007/BF01158964

[14] N. L. S. Fonseca, G. S. Mayor, and C. A. V. Neto, “On the equivalent bandwidth of self-similar sources,” *ACM Trans. Model. Comput. Simul.*, vol. 10, no. 2, pp. 104–124, Apr. 2000. [Online]. Available: http://doi.acm.org/10.1145/364996.365003

[15] P. Seeling and M. Reisslein, “Video traffic characteristics of modern encoding standards: H.264/avc with svc and mvc extensions and h.265/hevc,” *The Scientific World Journal*, vol. 2014, p. 16, 2014.

[16] Cisco, “Ios quality of service solutions configuration guide, release 12.2,” Tech. Rep., 2015, www.cisco.com/c/en/us/td/docs/ios/12/2/qos/configuration/guide/fqos/c.

[17] RNP, “Topologia,” na Internet, 07 2015, available at http://www.rnp.br/servicos/conectividade/rede-ipe.

[18] J. Moy, “Ospf version 2,” RFC 2328 (INTERNET STANDARD), Internet Engineering Task Force, apr 1998, updated by RFCs 5709, 6549, 6845, 6860. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt

[19] T. Szigeti and C. Hattingh, *End-to-End QoS Network Design: Quality of Service in LANs, WANs, and VPNs (Networking Technology)*. Cisco Press, 2004.

vtTLS: A Vulnerability-Tolerant Communication Protocol

André Joaquim Miguel L. Pardal Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Lisbon, Portugal
{andre.joaquim, miguel.pardal, miguel.p.correia}@tecnico.ulisboa.pt

***Abstract*—We present VTTLS, a vulnerability-tolerant communication protocol. There are often concerns about the strength of some of the encryption mechanisms used in SSL/TLS channels, with some regarded as insecure at some point in time. VTTLS is our solution to mitigate the problem of secure communication channels being vulnerable to attacks due to unexpected vulnerabilities in encryption mechanisms. It is based on diversity and redundancy of cryptographic mechanisms and certificates to provide a secure communication channel even when one or more mechanisms are vulnerable. VTTLS relies on a combination of k cipher suites. Even if $k-1$ cipher suites are insecure or vulnerable, VTTLS relies on the remaining cipher suites to maintain the channel secure. We evaluated the performance of VTTLS by comparing it to an OpenSSL channel.**

Keywords*—network protocol, secure communication channels, diversity, redundancy, vulnerability-tolerance.

I. Introduction

Secure communication protocols are fundamental building blocks of the current digital economy. *Transport Layer Security* (TLS) alone is responsible for protecting most economic transactions done using the web, with a value so high that it is hard to estimate. These protocols allow entities to exchange messages or data over a secure channel on the Internet, that provides *confidentiality* and *integrity* of communications. However, vulnerabilities may exist in the specification, the cryptographic mechanisms used, or in the implementation.

VTTLS is a new protocol proposed in this paper that provides *vulnerability-tolerant communication channels*. These channels do not rely on individual cryptographic mechanisms, so if one of them is found vulnerable (or possibly a few of them), the channels remain secure. The idea is to leverage *diversity* and *redundancy* of cryptographic mechanisms and keys, i.e., the use of different and multiple sets of mechanisms/keys, respectively. More specifically, diversity is employed in pair certificate/private key, key exchange mechanism, authentication mechanism, encryption mechanism and message authentication code (MAC). This use of diversity and redundancy is inspired by previous works on computer immunology [1], diversity in security [2], [3], and moving-target defenses [4].

VTTLS is configured with a parameter k, the *diversity factor* ($k > 1$). This parameter indicates the number of different cipher suites and different mechanisms for key exchange, authentication, encryption, and signing. This parameter means also that VTTLS remains secure as long as only $k - 1$ vulnerabilities exist. We expect k to be usually small, e.g., $k = 2$ or $k = 3$, because vulnerabilities, even if unknown (zero-day), do not appear in large numbers in the same components [5].

978-1-5090-3216-7/16/$31.00

The main contribution of this paper is the VTTLS protocol and an experimental evaluation that shows that it has an acceptable overhead when compared with the TLS implementation in which our prototype is based: OpenSSL [6].

II. Background and Related Work

This section presents related work on diversity (and redundancy) in security, provides background on TLS, and discusses vulnerabilities in cryptographic mechanisms.

A. Diversity in Security

The term *diversity* is used to describe multi-version software in which redundant versions are deliberately created and made different between themselves [2]. Without diversity, all instances are the same, with the same implementation vulnerabilities. Using diversity it is possible totw present different versions to the attacker, hopefully with different vulnerabilities. Software diversity targets mostly software implementation and the ability of the attacker to replicate the user's environment. Diversity does not change the program's logic, so it is not helpful if a program is badly designed. According to Littlewood and Strigini [2], multi-version systems on average are more reliable. They also state that the key to achieving effective diversity is that the dependence between the different programs needs to be as low as possible.

B. SSL/TLS Protocol Vulnerabilities

TLS is composed by the Handshake Protocol and the Record Protocol. The Handshake Protocol is used to establish or resume a secure session between two communicating parties – client and server. The Record protocol is responsible for processing all the messages to sent and received.

TLS vulnerabilities discovered in the past can be classified in two types: *specification vulnerabilities* that concern the protocol itself and can only be fixed by a new protocol version or an extension; and *implementation vulnerabilities* that exist in the code of some of the implementations of SSL/TLS.

One of the most recent attacks against a *specification vulnerability* is Logjam, a man-in-the-middle attack exploiting several Diffie-Hellman key exchange weaknesses [7]. *Heartbleed* is one of the most recent *implementation vulnerabilities*. It was a bug in OpenSSL 1.0.1 through 1.0.1f, when the heartbeat extension was introduced and enabled by default which allowed an attacker to perform a buffer over-read [8].

C. Vulnerabilities in Cryptographic Schemes

The Advanced Encryption Standard (AES) is the current American standard for symmetric encryption [9]. AES can be employed with different key sizes – 128, 192 or 256 bits. The number of rounds corresponding to each key size is, respectively, 10, 12 and 14. The most successful cryptanalysis of AES was published by Bogdanov *et al.* in 2011, using a biclique attack. Ferguson et al. [10] presented the first known attacks on the first seven and eight rounds of AES.

Regarding public-key cryptography, Kleinjung et al. [11] performed the factorization of RSA-768, a number with 232 digits. The researchers spent almost two years in the whole process, which is clearly a non-feasible time for most attacks.

Some generic attacks to hash function include brute force attacks, birthday attacks, and side-channel attacks. SHA-1 is a cryptographic hash function which produces a 160-bit message digest. Although there is no public knowledge of collisions for SHA-1, it is no longer recommended for use [12]. Stevens et al. [13] presented a freestart collision attack for SHA-1's internal compression function, where the attacker can choose the initial chaining value, known as initialization vector (IV). Regarding SHA-2 and its security, Khovratovich et al. [14] presented a biclique attack against SHA-2's preimage resistance. For several years, other researchers have also tried differential attacks for finding collisions and pseudo-collisions [15], [16].

III. Vulnerability-Tolerant TLS

VTTLS is a protocol for diverse and redundant vulnerability-tolerant secure communication channels. Unlike TLS, it negotiates more than one cipher suite between client and server. Diversity and redundancy appear firstly in VTTLS in the Handshake protocol, in which client and server negotiate the k cipher suites to be used in the communication. The server chooses the best combination of k cipher suites according to the cipher suites server and client have, and the available certificates. VTTLS uses a subset of the k cipher suites to encrypt the messages.

A. Protocol Specification

The VTTLS Handshake Protocol is similar to the TLS Handshake Protocol. The first message to be sent is ClientHello containing a list of the client's available cipher suites. The server responds with a ServerHello message containing the k cipher suites to be used in the communication. The server proceeds to send a (Server) Certificate message containing its k certificates. The ServerKeyExchange message is then sent to the client. For every k cipher suites using ECDHE or DHE, the server sends a ServerKeyExchange messages containing the server's DH ephemeral parameters for that cipher suite. Instead of computing all the ephemeral parameters and sending them all on a single larger message, the server sends each one immediately.

After sending its certificates, the client sends k ClientKeyExchange messages to the server. The content of these messages is based on the k cipher suites chosen. Client and server now exchange ChangeCipherSpec messages. Just like in the Cipher Spec Protocol of TLS 1.2, from that moment on, they use the previously negotiated cipher suites for encrypting messages. In order to finish the Handshake, the client and server send each other a Finished message. These are the first encrypted messages sent using the k cipher suites.

B. Combining Diverse Cipher Suites

Regarding integrity, all of VTTLS' cipher suites use either AEAD (Authenticated Encryption with Associated Data) (MAC-then-Encrypt mode), SHA-2 (SHA-256 or SHA-384), SHA-1 or MD5. The choice starts from the current security status of each hash function. While AEAD (which is not an hash function) and SHA-2 are considered secure, SHA-1 is being deprecated and MD5 is considered insecure. Therefore, we excluded SHA-1 and MD5 from the possible combinations of hash functions. Our choice for creating maximum diversity relies on AEAD plus a variant of SHA-2. As AEAD is a different approach to MACs, it is expected to be vulnerable to different attacks than the ones targeting hash functions, such as SHA-2. Using a combination of two SHA-2 variants would not create maximum diversity, because even though they have different digest sizes and rounds, their structure is identical. If SHA-3 was available in TLS, it could also be used.

As we want to increase diversity in order to increase security, we prioritize mechanisms which grant perfect forward secrecy (PFS) instead of mechanisms with disjoint mathematical hard problems. After comparing several public-key encryption mechanisms, we concluded that the best three combinations are: `RSA + ECDH(E)`, `ECDSA + ECDH(E)` and `ECDSA + RSA`. VTTLS uses public-key encryption mechanisms for key exchange and authentication. Regarding authentication, the preferred combination is `ECDSA + RSA`. Regarding key exchange, the preferred is `RSA + ECDH(E)`.

The symmetric mechanisms chosen for comparison were `AES`, `Camellia`, `SEED`, and `3DES EDE`. We also included `ARIA` in our comparison, even though it is not available in VTTLS. Symmetric-key encryption mechanisms' three most important metrics are structure, mode of operation and common known attacks. In a certain sense, these metrics are related: attacks target a specific structure or mode of operation. Therefore, the combinations of symmetric encryption mechanisms we consider to be the most diverse are: `AES + CAMELLIA` (using a different mode of operation and key size), and `AES256-GCM + SEED128-CBC`. We consider the most diverse combination to be `AES256-GCM + CAMELLIA128-CBC` due to the fact that their structure is different, as its mode of operation and the set of known attacks is disjoint. Although, using a cipher suite that contains `Camellia` or `SEED`, in order to maximize diversity in symmetric encryption would force reduced diversity and security essentially in MAC. In the end, the best combination is `AES256-GCM + AES128-CBC` which can be considered diverse, although it is not the most diverse of all the ones considered.

Concluding, the best combination of cipher suites is arguably: `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` and `TLS_RSA_WITH_AES_128_CBC_SHA256`. The least diverse mechanisms are MAC and symmetric encryption, due to the fact that TLS 1.2 does not support SHA-3 and OpenSSL does not support `Camellia` keyed-hash message authentication code (HMAC) based cipher suites.

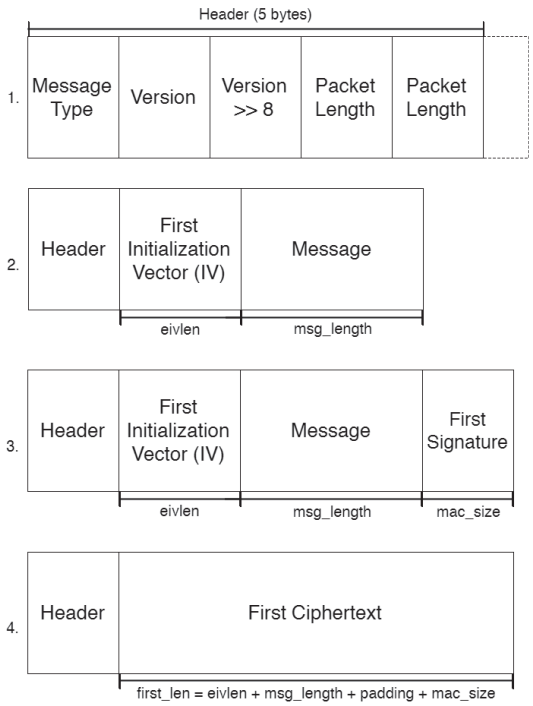


Fig. 1: First four steps regarding the ordering of the encryption and signing of VTTLS using a diversity factor $k = 2$.

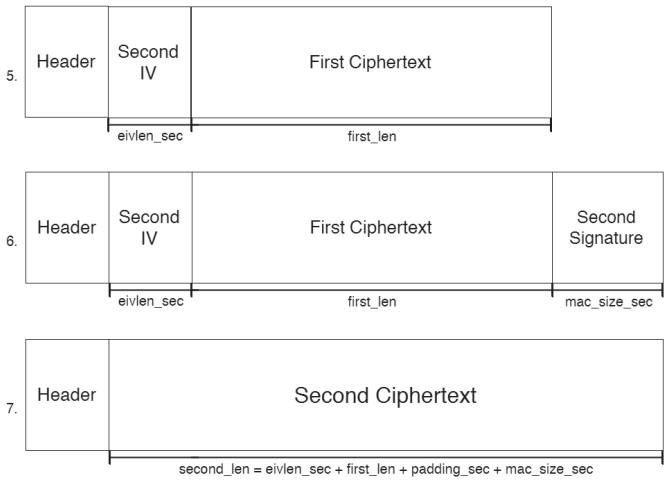


Fig. 2: Final three steps regarding the ordering of the encryption and signing of VTTLS using a diversity factor $k = 2$.

IV. Implementation

VTTLS' implementation is a modified version of OpenSSL v1.0.2g. Existing software such as OpenSSL has the advantage of being extensively tested. Furthermore, creating a new secure communication channel, and consequently a new API, would create adoption barriers to programmers otherwise willing to use our protocol. Therefore, we chose to implement VTTLS based on OpenSSL and keeping the same API. Nevertheless, OpenSSL is a huge code base (currently 438,841 lines of code) and modifying it so support diversity was quite a challenge. VTTLS adds a few functions to the OpenSSL API. They allow defining additional certificates, keys, cipher suites, etc. The signing and encryption ordering is very important for VTTLS. Figure 1 shows the ordering for one cipher and one MAC in the OpenSSL implementation.

The approach taken was the following, ordered from first to last: apply the first MAC to the plaintext; encrypt the first message and its MAC with the first encryption mechanism; apply the second MAC to the ciphertext; encrypt the ciphertext and its MAC with the second encryption mechanism. Figure 2 shows the final ordering of VTTLS communication data.

In relation to the *Record Protocol*, signing and encrypting k times has a cost in terms of message size. Figures 1 and 2 show also the expected increase of the message size due to the use of a second MAC and a second encryption function (for $k = 2$). For OpenSSL, the expected size of a message is $first\_len = eivlen + msg\_length + padding + mac\_size$, where $eivlen$ is the size of the initialization vector (IV), msg_length the original message size, $padding$ the size of the padding in case a block cipher is used, and mac_size the size of the MAC. For VTTLS, the additional size of the message is $eivlen\_sec + first\_len + padding\_sec + mac\_size\_sec$, where $eivlen\_sec$ is the size of the IV associated with the second cipher and mac_size_sec the size of the second MAC.

In the best case, the number of packets is the same for OpenSSL and VTTLS. In the worst case, one additional packet may be sent if the encryption function requires a fixed block size and the maximum size of the packet is exceeded by, at least, one byte after the second MAC and the second encryption. In this case, an additional full packet is needed due to the constraint of having fixed block size.

V. Experimental Evaluation

Implementing diversity has performance costs as it creates overhead in the communication. Every message sent needs to be ciphered and signed $k - 1$ times more than using a TLS implementation and every message received needs to be deciphered and verified also $k - 1$ times more. In the worst case, users should experience a connection k times slower than using OpenSSL. We considered $k = 2$ in all experiments, as this is the value we expect to be used in practice (we expect vulnerabilities to appear rarely, so the ability to tolerate one vulnerability per mechanism is sufficient). All the tests were done in the same controlled environment and same geographic locations in order to maintain the evaluation valid, exact and precise. We evaluated VTTLS's *performance* and *costs* and considered the OpenSSL implementation of TLS as the baseline.

A. Performance

In order to evaluate VTTLS performance, we executed several tests in order to understand if the overhead of VTTLS is lower, equal, or bigger than k comparing to OpenSSL. We configured VTTLS to use the following cipher suites: `TLS_RSA_WITH_AES_256_GCM_SHA384` and `TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384`. The suite used with OpenSSL was the latter.

To evaluate the performance of the handshake, we executed 100 times the Handshake Protocol of both VTTLS and

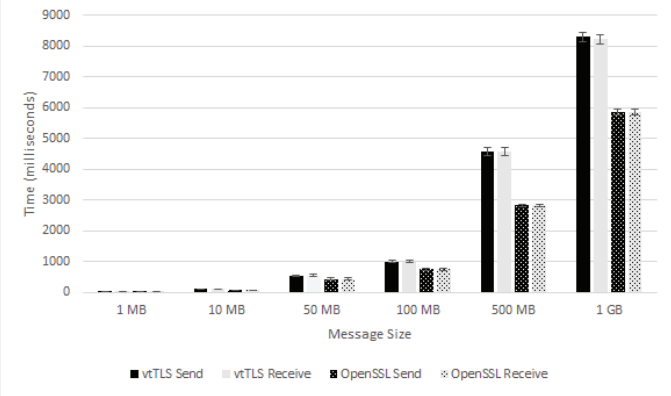


Fig. 3: Comparison between the time it takes to send and receive a message using VTTLS and OpenSSL.

OpenSSL. In average, the VTTLS handshake took 3.909 milliseconds to conclude whereas the OpenSSL handshake took 2.345 milliseconds. Therefore, the VTTLS handshake takes 66.7% longer than the OpenSSL handshake which is better than the worst case (that would be 100% longer).

B. Data Communication

We also performed data communication tests to assess the overhead generated by the diversity and redundancy of mechanisms. The communication is expected to be, at most, $k = 2$ times slower than using TLS. For this test, we considered a sample of 100 messages sent and received using VTTLS, and other 100 messages sent and received using OpenSSL. Figure 3 shows the comparison between the time it takes to send and receive a message using VTTLS and OpenSSL.

In average, a message sent through a VTTLS channel takes 22.88% longer than a message sent with OpenSSL. For example, a 50 MB message takes, in average 534.55 ms to be sent with VTTLS. Using OpenSSL, the same message takes 435.01 ms to be sent. The overhead generated is much smaller than the worst case.

We evaluated the message size increase of the ciphertext of several plaintexts with different sizes. A 1 MB plaintext message corresponds to a ciphertext of $1,029,054$ bytes using VTTLS, while using OpenSSL the same message converts into a message of $1,025,856$ bytes i.e. sending 1 MB through a VTTLS channel costs an additional $3,198$ bytes over using an OpenSSL channel.

VI. Conclusions

VTTLS is a diverse and redundant vulnerability-tolerant secure communication protocol designed for communication on the Internet. It aims at increasing security using diverse cipher suites to tolerate vulnerabilities in the encryption mechanisms used in the communication channel. In order to evaluate our solution, we compared it to an OpenSSL communication channel. While expected to be $k = 2$ times slower than an OpenSSL channel, the evaluation showed that using diversity and redundancy of cryptographic mechanisms in VTTLS does not generate such a high overhead. VTTLS takes, in average, 22.88% longer to send a message than TLS/OpenSSL, but considering the increase in security, this overhead is acceptable. Overall, considering the additional costs of having an extra certificate, the time increase, and potential management costs, VTTLS provides an interesting trade-off for a set of critical security applications.

Acknowledgements This work was supported by the European Commission through project H2020-653884 (SafeCloud) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 (INESC-ID).

References

[1] S. Forrest, S. A. Hofmeyr, and A. Somayaji, "Computer immunology," *Communications of the ACM*, vol. 40, no. 10, pp. 88–96, Oct. 1997.

[2] B. Littlewood and L. Strigini, "Redundancy and diversity in security," in *Computer Security – ESORICS 2004, 9th European Symposium on Research Computer Security*, 2004, pp. 227–246.

[3] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "OS diversity for intrusion tolerance: Myth or reality?" in *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks*, 27–30 June 2011, pp. 383 –394.

[4] M. Carvalho and R. Ford, "Moving-target defenses for computer networks," *IEEE Security and Privacy*, vol. 12, no. 2, pp. 73–76, 2014.

[5] L. Bilge and T. Dumitras, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2012, pp. 833–844.

[6] J. Viega, M. Messier, and P. Chandra, *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly, 2002.

[7] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. Vandersloot, E. Wustrow, and S. Paul, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, October 2015. [Online]. Available: https://weakdh.org/imperfect-forward-secrecy.pdf

[8] M. Carvalho, J. DeMott, R. Ford, and D. A. Wheeler, "Heartbleed 101," *IEEE Security Privacy*, vol. 12, no. 4, pp. 63–67, July 2014.

[9] V. Rijmen and J. Daemen, "Advanced Encryption Standard," *U.S. National Institute of Standards and Technology (NIST)*, vol. 2009, pp. 8–12, 2001.

[10] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved cryptanalysis of Rijndael," in *Proceedings of Fast Software Encryption*, G. Goos, J. Hartmanis, J. van Leeuwen, and B. Schneier, Eds. Springer, 2001, vol. LNCS 1978, pp. 213–230.

[11] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thomé, J. Bos, P. Gaudry, A. Kruppa, P. Montgomery, D. Osvik, H. Te Riele, A. Timofeev, and P. Zimmermann, "Factorization of a 768-bit RSA modulus," in *Proceedings of the 30th Annual Conference on Advances in Cryptology*, vol. LNCS 6223, 2010, pp. 333–350.

[12] ENISA, "Algorithms, key size and parameters report – 2014," nov 2014.

[13] M. Stevens, P. Karpman, and T. Peyrin, "Freestart collision on full SHA-1," Cryptology ePrint Archive, Report 2015/967, 2015.

[14] D. Khovratovich, C. Rechberger, and A. Savelieva, "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family," Cryptology ePrint Archive, Report 2011/286, 2011, http://eprint.iacr.org/.

[15] C. Dobraunig, M. Eichlseder, and F. Mendel, "Analysis of SHA-512/224 and SHA-512/256," Cryptology ePrint Archive, Report 2016/374, 2016, http://eprint.iacr.org/.

[16] M. Eichlseder, F. Mendel, and M. Schlffer, "Branching Heuristics in Differential Collision Search with Applications to SHA-512," Cryptology ePrint Archive, Report 2014/302, 2014, http://eprint.iacr.org/.

Optimal Proportion Computation with Population Protocols

Yves Mocquard
yves.mocquard@irisa.fr
IRISA / Université de Rennes 1, France

Emmanuelle Anceaume
emmanuelle.anceaume@irisa.fr
IRISA / CNRS, France

Bruno Sericola
bruno.sericola@inria.fr
Inria Rennes - Bretagne Atlantique, France

***Abstract*—The computational model of population protocols is a formalism that allows the analysis of properties emerging from simple and pairwise interactions among a very large number of anonymous finite-state agents. Significant work has been done so far to determine which problems are solvable in this model and at which cost in terms of states used by the agents and time needed to converge. The problem tackled in this paper is the *population proportion* problem: each agent starts independently from each other in one of two states, say A or B, and the objective is for each agent to determine the proportion of agents that initially started in state A, assuming that each agent only uses a finite set of states, and does not know the number n of agents. We propose a solution which guarantees that in presence of a uniform probabilistic scheduler every agent outputs the population proportion with any precision $\varepsilon \in (0, 1)$ with any high probability after having interacted $O(\log n)$ times. The number of states maintained by every agent is optimal and is equal to $2\lceil 3/(4\varepsilon)\rceil + 1$. Finally, we show that our solution is optimal in time and space to solve the counting problem, a generalization of the proportion problem. Finally, simulation results illustrate our theoretical analysis.**

Keywords Population protocols; Proportion; Majority; Counting; Performance evaluation.

I. Introduction

In 2004, Angluin et al. [4] have proposed a model that allows the analysis of emergent global properties based on pairwise interactions. This model, named the population protocol, provides minimalist assumptions on the computational power of the agents: agents are finite-state automata, identically programmed, with no identity, unaware of the population size n, and they progress in their computation through random pairwise interactions. The objective of this model is to ultimately converge to a state from which the sought property can be derived from any agent [8]. Examples of systems whose behavior can be modeled by population protocols range from molecule interactions of a chemical process to sensor networks in which agents, which are small devices embedded on animals, interact each time two animals are in the same radio range. A considerable amount of work has been done so far to determine which properties can emerge from pairwise interactions between finite-state agents, together with the derivation of lower bounds on the time and space needed to reach such properties (e.g., [2], [6], [12], [14], [18]). Among them, is majority. Briefly, each agent starts independently from each other in one of two input states, say A and B, and the objective for each agent is to eventually output *yes* if a majority of agents started their execution in input state A and *no* otherwise. Section IV provides an overview of the results recently obtained for the majority task.

In this paper, we focus on a related but more general question. Namely, instead of having each agent answer *yes* if a majority of agents initially started their execution in input state A, one may ask the following question:

"Is it feasible for each agent to compute *quickly* and with any *high precision* the *proportion of agents* that started in the input state A?".

Answering such a question is very important in the context of, for example, infectious-disease surveillance of large-scale animal populations. In this context, different kinds of alerts could be triggered according to the infected population proportion (*e.g.*, Alert 1 is triggered if less than 0.05% of the population is infected, Alert 2 if this proportion lies in $[0.05\%, 3.0\%)$, Alert 3 if it lies in $[3.0\%, 10.0\%)$, and so on ...). Input state A would manifest an excessive temperature of an animal while input state B would indicate a safe temperature. By relying on the properties exhibited by our population protocol (convergence time logarithmic in the population size and memory space proportional to the sought precision), one can easily implement a regular and self-autonomous monitoring of large-scale populations.

We answer affirmatively to this question, and we propose a population protocol that allows each agent to converge to a state which, when queried, provides the proportion of agents that started in a given input state. Specifically, each agent is a $(2m + 1)$-finite state machine, $m \geq 1$, where m is the value associated to input state A and $-m$ is the one associated to input state B. Each agent starts its execution with m or $-m$, and each pair of agents that meet, adopt the average of their values (or as close as they can get when values are restricted to integers, as will be clarified in Section V). The rationale of this method [3], [16], [1] is to preserve the sum of the initial values, and after a small number of pairwise interactions, to ensure that every agent converges with high probability to a state from which it derives the proportion of agents that started in a given state. Technically, our protocol guarantees that each agent is capable of computing with any precision $\varepsilon \in (0, 1)$ the proportion of agents that initially started in a specific input state by using $2\lceil 3/(4\varepsilon)\rceil + 1$ states. This is achieved in no more than $(-2\ln\varepsilon + 8.47\ln n - 13.29\ln\delta - 2.88)$ interactions with probability at least $1 - \delta$, for any $\delta \in (0, 1)$.

This work was partially funded by the French ANR project SocioPlug (ANR-13-INFR-0003), and by the DeSceNt project granted by the Labex CominLabs excellence laboratory (ANR-10-LABX-07-01).

Our second contribution relates to the counting problem. The counting problem generalizes the majority problem by requiring, for each agent, to converge to a state in which each agent is capable of, assuming the knowledge of n, computing n_A or n_B, where n_A and n_B represent respectively the number of agents that started in state A and B. In the present paper, we prove that the counting problem can be solved using $O(n)$ states per agent. This significantly improves upon a previous analysis [16] that shows that $O(n^{3/2}/\delta^{1/2})$ states allow each agent to converge to the exact solution in no more than a logarithmic number in n of interactions, with $\delta \in (0,1)$. What is very important to notice is that this drastic improvement is due to an original convergence analysis that allows us to refine previous results. Indeed, both [16] and the present paper rely on the same interaction rules, however by precisely characterizing the evolution of the interacting agents, our present analysis is highly tighter. We also demonstrate that any protocol that solves the counting problem requires $\Omega(\log n)$ parallel interactions to converge and $\Omega(n)$ local states. As will be detailed, this shows that our algorithm is an optimal solution both in space and time to solve the counting problem and optimal in space to solve the proportion one.

The remainder of this paper is organized as follows. Section II presents the population protocol model. Section III specifies the problem addressed in this work. Section IV provides an overview of the most recent population protocols. The protocol to compute the population proportion is presented in Section V. Analysis of the protocol is detailed in Section VI. We show in Section VII, that our protocol is optimal both in space and time. We have simulated our protocol to illustrate our theoretical analysis. Section VIII presents a summary of these simulation results. Finally, Section IX concludes.

II. Population protocols model

The population protocol model has been introduced by Angluin et al. [4]. This model describes the behavior of a collection of agents that interact pairwise. The following definition is from Angluin et al [7]. A population protocol is characterized by a 6-tuple $(Q, \Sigma, Y, \iota, \omega, f)$, over a complete interaction graph linking the set of n agents, where Q is a finite set of states, Σ is a finite set of input symbols, Y is a finite set of output symbols, $\iota : \Sigma \to Q$ is the input function that determines the initial state of an agent, $\omega : Q \to Y$ is the output function that determines the output symbol of an agent, and $f : Q \times Q \to Q \times Q$ is the transition function that describes how any two distinct agents interact and locally update their states. Initially all the agents start with a initial symbol from Σ, and upon interactions update their state according to the transition function f. Interactions between agents are orchestrated by a random scheduler: at each discrete time, any two agents are randomly chosen to interact with a given distribution. Note that the random scheduler is fair, meaning that any possible interaction cannot be avoided forever. The notion of time in population protocols refers to as the successive steps at which interactions occur, while the parallel time is equal to the total number of interactions averaged by n [8]. Agents do not maintain nor use identifiers (agents are anonymous and cannot determine whether any two interactions have occurred with the same agents or not). However, for ease of presentation, the agents are numbered $1, 2, \ldots, n$. We denote by $C_t^{(i)}$ the state of agent i at time t. The stochastic process $C = \{C_t,\ t \geq 0\}$, where $C_t = (C_t^{(1)}, \ldots, C_t^{(n)})$, represents the evolution of the population protocol. The state space of C is thus Q^n and a state of this process is also called a protocol configuration.

III. The Proportion Problem

We consider a set of n agents, interconnected by a complete graph, that start their execution in one of two input states of $\Sigma = \{A, B\}$. Let n_A be the number of agents whose input state is A and n_B be the number of agents that start in input state B. The quantity $\gamma_A = n_A/n$ (resp. $\gamma_B = n_B/n$) is the proportion of the agents that initially started in state A (resp. in state B). The output set Y is the set of all possible values of γ_A, that is a subset of $[0, 1]$. In the following we introduce the notation $\gamma = \gamma_A - \gamma_B$. Let $\omega_A(C_t^{(i)})$ be the approximation of γ_A by agent i at time t.

A population protocol solves the proportion problem within τ steps (with preferably τ in $O(\log n)$) if for all $\delta \in (0,1)$, for all $\varepsilon \in (0,1)$ and for all $t \geq \tau$, we have

$$\mathbb{P}\{|\omega_A(C_t^{(i)}) - \gamma_A| < \varepsilon \text{ for all } i = 1, \ldots, n\} \geq 1 - \delta.$$

IV. Related Work

In 2004, Angluin et al. [4] have formalized the population protocol model, and have shown how to express and compute predicates in this model. Then in [5] the authors have completely characterized the computational power of the model by establishing the equivalence between predicates computable in the population model and those that can be defined in the Presburger arithmetic. Since then, there has been a lot of work on population protocols including the majority problem [12], [14], [6], [18], [2], the leader election problem [9], [15], in presence of faults [11], and on variants of the model [13], [10].

The closest problem to the one we address is the computation of the majority. In this problem, all the agents start in one of two distinguished states and they eventually converge to 1 if $\gamma > 0$ (*i.e.* $n_A > n_B$), and to 0 if $\gamma < 0$ (*i.e.* $n_A < n_B$). In [12], [14] the authors propose a four-state protocol that solves the majority problem with a convergence parallel time logarithmic in n but only in expectation. Moreover, the expected convergence time is infinite when n_A and n_B are close to each other (that is γ approaches 0). The authors in [6], [18] propose a three-state protocol that converges with high probability after a convergence parallel time logarithmic in n but only if γ is large enough, i.e when $|n_A - n_B| \geq \sqrt{n} \log n$. Alistarh et al. [2] propose a population protocol based on an average-and-conquer method to exactly solve the majority problem. Their algorithm uses two types of interactions, namely, averaging interactions and conquer ones. The first type of interaction is close to the one used in our protocol while the second one is used to diffuse the result of the computation to the zero state agents. Actually, to show their convergence time, they need to assume a rather large number of intermediate states (*i.e.* $2d$ states, with $d = 1,000$). This is essentially due to the fact that they need to prove that all the agents with maximum positive values and minimal negative values will have sufficiently enough time to halve their values. Note that in practice, their algorithm does not require more

than n state to converge to the majority, however their proof necessitates $m+1,000\log m\log n$ with $\log n\log m \leq m \leq n$ states, and at least $432\log m\log n$ interactions per agent to converge to the majority, where m is the initial value associated to state A.

In [16], the authors have presented a solution to the counting problem. As previously said, the counting problem generalizes the majority problem by requiring, for each agent, to converge to a state in which each agent is capable of, assuming the knowledge of n, computing n_A or n_B, where n_A and n_B represent respectively the number of agents that started in state A and B. Both [16] and the present paper use the same interaction rules, but of course the output functions in both papers are different. The originality of [16], beyond tackling a new problem, was a proof of convergence based on tracking the euclidean distance between the random vector of all agents' values and the limiting distribution. In the present paper, we provide a highly tighter analysis which shows that the interaction rules together with the "counting" and "proportion" output functions are optimal solutions to solve both problems.

V. Computing the Proportion

Our protocol uses the average technique to compute the proportion of agents that started their execution in a given state A. The set of input of the protocol is $\Sigma = \{A, B\}$, and the input function ι is defined by $\iota(A) = m$ and $\iota(B) = -m$, with m a positive integer. This means that, for every $i = 1,\ldots,n$, we have $C_0^{(i)} \in \{-m, m\}$. At each discrete instant t, two distinct indices i and j are chosen among $1,\ldots,n$ with probability $p_{i,j}(t)$. Once chosen, the couple (i,j) interacts, and both agents update their respective local state $C_t^{(i)}$ and $C_t^{(j)}$ by applying the transition function f, leading to state C_{t+1}, given by $f(C_t^{(i)}, C_t^{(j)}) = (C_{t+1}^{(i)}, C_{t+1}^{(j)})$, with

$$\left(C_{t+1}^{(i)}, C_{t+1}^{(j)}\right) = \left(\left\lfloor \frac{C_t^{(i)} + C_t^{(j)}}{2} \right\rfloor, \left\lceil \frac{C_t^{(i)} + C_t^{(j)}}{2} \right\rceil\right) \text{ and}$$
$$C_{t+1}^{(m)} = C_t^{(m)} \text{ for } m \neq i, j. \quad (1)$$

The set Q of states is $\{-m, -m+1, \ldots, m-1, m\}$. The output function is given, for all $x \in Q$ by,

$$\omega_A(x) = (m+x)/2m.$$

Finally, the set of output Y is the set of all possible values of ω_A, i.e.

$$Y = \left\{0, \frac{1}{2m}, \frac{2}{2m}, \ldots, \frac{2m-2}{2m}, \frac{2m-1}{2m}, 1\right\}.$$

VI. Analysis of the Proportion Protocol

We denote by X_t the random variable representing the choice at time t of two distinct indices i and j among $1,\ldots,n$ with probability $p_{i,j}(t)$, that is $\mathbb{P}\{X_t = (i,j)\} = p_{i,j}(t)$. We suppose that the sequence $\{X_t,\ t \geq 0\}$ is a sequence of independent and identically distributed random variables. Since C_t is entirely determined by the values of $C_0, X_0, X_1, \ldots, X_{t-1}$, this means in particular that the random variables X_t and C_t are independent and that the stochastic process C is a discrete-time homogeneous Markov chain. As usual in population protocols, we suppose that X_t is uniformly distributed, i.e. that is

$$p_{i,j}(t) = \frac{1}{n(n-1)}.$$

We will use in the sequel the Euclidean norm denoted simply by $\|.\|$ and the infinite norm denoted by $\|.\|_\infty$ defined for all $x = (x_1,\ldots,x_n) \in \mathbb{R}^n$ by

$$\|x\| = \left(\sum_{i=1}^n x_i^2\right)^{1/2} \text{ and } \|x\|_\infty = \max_{i=1,\ldots,n} |x_i|.$$

It is well-known that these norms satisfy $\|x\|_\infty \leq \|x\| \leq \sqrt{n}\|x\|_\infty$.

Lemma *1:* For every $t \geq 0$, we have

$$\sum_{i=1}^n C_t^{(i)} = \sum_{i=1}^n C_0^{(i)}.$$

Proof: The proof is immediate since the transformation from C_t to C_{t+1} described in Relation (1) does not change the sum of the entries of C_{t+1}. Indeed, from Relation (1), we have $C_{t+1}^{(i)} + C_{t+1}^{(j)} = C_t^{(i)} + C_t^{(j)}$ and the other entries do not change their values. ■

We denote by ℓ the mean value of the sum of the entries of C_t and by L the row vector of $\mathbb{R}^n$ with all its entries equal to ℓ, that is

$$\ell = \frac{1}{n}\sum_{i=1}^n C_t^{(i)} \text{ and } L = (\ell,\ldots,\ell).$$

Our analysis is orchestrated as follows. By relying on the mathematical tool derived in Theorem 2, we show in Theorem 5 that the stochastic process C_t belongs to the ball of radius $\sqrt{n/2}$ and center L in the 2-norm, with any high probability, after no more than $O(\log n)$ parallel time. Then, assuming that the stochastic process C_t belongs to the ball of radius $\sqrt{n/2}$ and that $\ell - \lfloor \ell \rfloor \neq 1/2$, we demonstrate that the stochastic process C_t belongs to the open ball of radius $3/2$ and center L in the infinite norm, with any high probability after no more than $O(\log n)$ parallel time (Theorem 6). In practice this means that all the entries of the subsequent configurations will be among the three closest integer values of ℓ. Then by applying Theorem 5 and Theorem 6 (if $\ell - \lfloor \ell \rfloor \neq 1/2$) or Theorem 4 (otherwise), we derive our main theorem (see Theorem 7) which shows that in both cases the stochastic process C_t belongs to an open ball of radius $3/2$ and center L in the infinite norm, with any high probability in $O(\log n)$ parallel time. Finally, we have all the necessary tools to construct an output function which solves the proportion problem in $O\left(\log n - \log\varepsilon - \log\delta\right)$ parallel time, and with $O(1/\varepsilon)$ states, for any $\varepsilon, \delta \in (0,1)$ (see Theorem 8). The detailed proofs of all the results are given in [17].

In order to simplify the writing we will use the notation $Y_t = \|C_t - L\|^2$ when needed and we denote by $1_{\{A\}}$ the indicator function which is equal to 1 if condition A is satisfied and 0 otherwise.

The following Theorem is a conditional version of Theorem 6 of [16].

Theorem *2:* For every $0 \leq s \leq t$ and $y \geq 0$, we have

$$\mathbb{E}\left(Y_t \mid Y_s \geq y\right) \leq \left(1 - \frac{1}{n-1}\right)^{t-s} \mathbb{E}\left(Y_s \mid Y_s \geq y\right) + \frac{n}{4}. \tag{2}$$

Proof: See [17]. ■

Lemma *3:* The sequence $Y_t = \|C_t - L\|^2$ is decreasing with t.

Proof: See [16]. ■

Theorem *4:* For all $\delta \in (0,1)$, if $\ell - \lfloor \ell \rfloor = 1/2$ and if there exists a constant K such that $\|C_0 - L\|_\infty \leq K$, then, for every $t \geq (n-1)\left(2\ln K + \ln n - \ln \delta\right)$, we have

$$\mathbb{P}\{\|C_t - L\|_\infty \neq 1/2\} \leq \delta.$$

Sketch of the Proof: If $\ell - \lfloor \ell \rfloor = 1/2$ then, since all the $C_t^{(i)}$ are integers, we have $\|C_t - L\|^2 \geq n/4$. From Theorem 2 in which we set $s = 0$ and $y = 0$, we obtain

$$\mathbb{E}(\|C_t - L\|^2 - n/4) \leq \left(1 - \frac{1}{n-1}\right)^t \mathbb{E}(\|C_0 - L\|^2).$$

Let $\tau = (n-1)\left(2\ln K + \ln n - \ln \delta\right)$. For $t \geq \tau$, the Markov inequality leads to

$$\mathbb{P}\{\|C_t - L\|^2 - n/4 \geq 1\} \leq \delta.$$

We then conclude by observing that

$$\mathbb{P}\{\|C_t - L\|_\infty \neq \frac{1}{2}\} = \mathbb{P}\{\|C_t - L\|^2 - n/4 \geq 1\} \leq \delta.$$

The reader is invited to read the detailed proof in [17]. ■

Theorem *5:* For all $\delta \in (0, 4/5)$, if there exists a constant K such that $K \geq \sqrt{n/2}$ and $\|C_0 - L\| \leq K$ then, for all $t \geq n\theta$, we have

$$\mathbb{P}\{\|C_t - L\|^2 \geq n/2\} \leq \delta$$

where

$$\theta = 2\ln K - \ln n + 3\ln 2 - \frac{2\ln 2}{2\ln 2 - \ln 3}\ln \delta.$$

Sketch of the Proof: Let $(T_k)_{k \geq 0}$ be the sequence of instants defined by $T_0 = 0$ and

$$T_{k+1} = T_k + \left\lceil (n-1)\ln\left(\frac{8\mathbb{E}(Y_{T_k} \mid Y_{T_k} \geq n/2)}{n}\right)\right\rceil. \tag{3}$$

From Theorem 2 and Formula 3 we get

$$\mathbb{E}\left(Y_{T_{k+1}} \mid Y_{T_k} \geq n/2\right) \leq 3n/8. \tag{4}$$

Using the conditional Markov inequality, we deduce that

$$\mathbb{P}\{Y_{T_{k+1}} \geq n/2 \mid Y_{T_k} \geq n/2\} \leq 3/4.$$

We introduce the sequence α_k defined by $\alpha_0 = 3n/(8K^2)$ and, for $k \geq 1$,

$$\alpha_k = \max\left\{\mathbb{P}\{Y_{T_k} \geq n/2 \mid Y_{T_{k-1}} \geq n/2\}, 3n/(8K^2)\right\}.$$

We then obtain for every $k \geq 0$,

$$\mathbb{E}(Y_{T_k} \mid Y_{T_k} \geq n/2) \leq 3n/(8\alpha_k).$$

Summing the differences $T_{i+1} - T_i$ for i from 0 to $k-1$, we obtain, for $k \geq 1$,

$$T_k \leq (n-1)\left(k\ln(3) - \ln\left(\prod_{i=0}^{k-1}\alpha_i\right)\right) + k, \tag{5}$$

and we have

$$\mathbb{P}\{Y_{T_k} \geq n/2\} \leq \prod_{i=1}^{k}\alpha_i. \tag{6}$$

Now, for all $\delta \in (0, 4/5)$, there exists $k \geq 1$ such that

$$\prod_{i=1}^{k}\alpha_i < \delta \leq \prod_{i=1}^{k-1}\alpha_i \leq (3/4)^{k-1}.$$

From Relation (6) and using the fact that Y_t is decreasing (see Lemma 3) we get, for $t \geq n\theta$,

$$\begin{aligned}\mathbb{P}\{Y_t \geq n/2\} &\leq \mathbb{P}\{Y_{n\theta} \geq n/2\} \\ &\leq \mathbb{P}\{Y_{T_k} \geq n/2\} \\ &\leq \prod_{i=1}^{k}\alpha_i \leq \delta.\end{aligned}$$

The reader is invited to read the detailed proof in [17]. ■

Theorem *6:* For all $\delta \in (0,1)$, if $\|C_0 - L\| \leq \sqrt{n/2}$ and $\ell - \lfloor \ell \rfloor \neq 1/2$ we have, for every $t \geq 1600(n-1)\left(\ln n - \ln \delta - 4\ln 2 + \ln 3\right)/189$,

$$\mathbb{P}\{\|C_t - L\|_\infty \geq 3/2\} \leq \delta.$$

Sketch of the Proof: Let λ be defined by

$$\lambda = \begin{cases} \ell - \lfloor \ell \rfloor & \text{if} \quad \ell - \lfloor \ell \rfloor < 1/2 \\ \ell - \lceil \ell \rceil & \text{if} \quad \ell - \lfloor \ell \rfloor > 1/2. \end{cases}$$

Note that λ is positive in the first case and negative in the second one. In both cases we have $|\lambda| < 1/2$ and $\ell - \lambda$ is the closest integer to ℓ.

We introduce the notation $B = \lceil 1/2 + \sqrt{n/2} \rceil$. B is the upper bound of $C_t^{(i)}$ for $i = 1, \ldots, n$, since $\|C_0 - L\| \leq \sqrt{n/2}$.

For $k \in \{-B, -B+1, \ldots, B\}$, we denote by $\alpha_{k,t}$ the number of agents with the value $\ell - \lambda + k$ at time t, that is

$$\alpha_{k,t} = \left|\left\{i \in \{1, \ldots, n\} \mid C_t^{(i)} = \ell - \lambda + k\right\}\right|.$$

It is easily checked that

$$\sum_{k=-B}^{B}\alpha_{k,t} = n. \tag{7}$$

Moreover we have

$$\sum_{k=-B}^{B} k\alpha_{k,t} = n\lambda. \tag{8}$$

Observing that $\|C_t - L\|^2 = \|C_t\|^2 - n\ell^2$ and using (7) and (8), we obtain

$$\sum_{k=-B}^{B} k^2\alpha_{k,t} = \|C_t - L\|^2 + n\lambda^2. \tag{9}$$

Using the hypothesis $\|C_0 - L\|^2 \leq n/2$, we obtain

$$\sum_{k=-B}^{B} k^2 \alpha_{k,t} \leq n\lambda^2 + n/2. \tag{10}$$

Using these results, it can be shown that

$$\sum_{k=0}^{B} \alpha_{k,t} > 3n/8 \qquad \text{and} \qquad \sum_{k=-B}^{0} \alpha_{k,t} > 3n/8. \tag{11}$$

Let us now introduce the sequences $(N_t)_{t\geq 0}$ and $(\Phi_t)_{t\geq 0}$ defined by

$$N_t = \sum_{k=2}^{B} \alpha_{k,t} + \sum_{k=-B}^{-2} \alpha_{k,t}$$

and

$$\Phi_t = \sum_{k=2}^{B} k^2 \alpha_{k,t} + \sum_{k=-B}^{-2} k^2 \alpha_{k,t}.$$

Since $\alpha_{k,t}$ are non negative integers, we have, for every $t \geq 0$,

$$N_t = 0 \Longleftrightarrow \Phi_t = 0.$$

Note that our objective is to obtain $\Phi_t = 0$ because

$$\|C_t - L\|_\infty < 3/2 \Longleftrightarrow \Phi_t = 0.$$

We also introduce the sets H_t^+ and H_t^- defined by

$$H_t^+ = \{i \in \{1, \ldots, n\} \mid C_t^{(i)} - \ell + \lambda \geq 2\}$$

and

$$H_t^- = \{i \in \{1, \ldots, n\} \mid C_t^{(i)} - \ell + \lambda \leq -2\}$$

and we define $H_t = H_t^+ \cup H_t^-$. It can be shown that

$$N_t \leq 3n/16.$$

Let I_t^+ and I_t^- be the sets defined by

$$I_t^+ = \{i \in \{1, \ldots, n\} \mid C_t^{(i)} - \ell + \lambda \geq 0\}$$

and

$$I_t^- = \{i \in \{1, \ldots, n\} \mid C_t^{(i)} - \ell + \lambda \leq 0\}.$$

Relations (11) can be rewritten as

$$|I_t^+| \geq 3n/8 \text{ and } |I_t^-| \geq 3n/8. \tag{12}$$

Consider the probability that an agent of H_t^+ interacts with an agent of I_t^- or that an agent of H_t^- interacts with an agent of I_t^+, at time t. Let E denote the set of these interactions. It can be shown that

$$\mathbb{P}\{X_t \in E\} \geq \frac{21 N_t}{32(n-1)}. \tag{13}$$

We consider now the difference $\Phi_t - \Phi_{t+1}$ in function of the interactions occurring at time t. It can be shown that

$$\mathbb{E}(\Phi_t - \Phi_{t+1} \mid X_t \in E) \geq \frac{9\mathbb{E}(\Phi_t)}{50 N_t}.$$

Now, using (13), we have

$$\begin{aligned}
\mathbb{E}(\Phi_{t+1}) &= \mathbb{E}(\Phi_t) - \mathbb{E}(\Phi_t - \Phi_{t+1}) \\
&\leq \mathbb{E}(\Phi_t) - \mathbb{E}((\Phi_t - \Phi_{t+1}) \mid X_t \in E)\mathbb{P}\{X_t \in E\} \\
&\leq \mathbb{E}(\Phi_t) - \left(\frac{9\mathbb{E}(\Phi_t)}{50 N_t}\right)\left(\frac{21 N_t}{32(n-1)}\right) \\
&= \left(1 - \frac{189}{1600(n-1)}\right)\mathbb{E}(\Phi_t).
\end{aligned}$$

We easily get

$$\mathbb{E}(\Phi_t) \leq \left(1 - \frac{189}{1600(n-1)}\right)^t \mathbb{E}(\Phi_0).$$

Let τ be defined by

$$\tau = \frac{1600(n-1)}{189}\left(\ln n - \ln \delta - 4\ln 2 + \ln 3\right).$$

We then have for $t \geq \tau$

$$\mathbb{P}\{\|C_t^{(i)} - \ell\|_\infty \geq 3/2\} \leq \mathbb{P}\{\Phi_t \neq 0\} = \mathbb{P}\{\Phi_t \geq 4\} \leq \delta.$$

The reader is invited to read the detailed proof in [17]. ■

Theorem *7:* For all $\delta \in (0,1)$, if there exists a constant K such that $\|C_0 - L\| \leq K$ then, for every $t \geq n\left(2\ln K + 7.47 \ln n - 13.29 \ln \delta - 2.88\right)$, we have

$$\mathbb{P}\{\|C_t - L\|_\infty \geq 3/2\} \leq \delta.$$

Proof: We consider first the case where $\ell - \lfloor \ell \rfloor = 1/2$. Since $\|C_0 - L\|_\infty \leq \|C_0 - L\| \leq K$ and since

$$\begin{aligned}
&(n-1)(2\ln K + \ln n - \ln \delta) \\
&\qquad \leq n\left(2\ln K + 7.47 \ln n - 13.29 \ln \delta - 2.88\right),
\end{aligned}$$

Theorem 4 gives

$$\mathbb{P}\{\|C_t - L\|_\infty \neq 1/2\} \leq \delta,$$

for $t \geq n\left(2\ln K + 7.47 \ln n - 13.29 \ln \delta - 2.88\right)$.

Now since the $C_t^{(i)}$ are integers and since $\ell - \lfloor \ell \rfloor = 1/2$, we have

$$\mathbb{P}\{\|C_t - L\|_\infty \geq 3/2\} = \mathbb{P}\{\|C_t - L\|_\infty \neq 1/2\} \leq \delta.$$

Consider now the case where $\ell - \lfloor \ell \rfloor \neq 1/2$. We apply successively Theorem 5 and Theorem 6 replacing δ by $\delta/2$. We introduce the notation

$$\theta_1 = 2\ln K - \ln n + 3\ln 2 - \frac{2\ln 2}{2\ln 2 - \ln 3}\ln(\delta/2).$$

If $\|C_0 - L\| < \sqrt{n/2}$ then we have $\|C_0 - L\|^2 < n/2$ and since $\|C_t - L\|^2$ is decreasing (see Lemma 3), we get, for all $t \geq 0$,

$$\begin{aligned}
\mathbb{P}\{\|C_t - L\|^2 < n/2\} &\geq \mathbb{P}\{\|C_0 - L\|^2 < n/2\} \\
&= 1 \geq 1 - \delta/2.
\end{aligned}$$

If $\|C_0 - L\| \geq \sqrt{n/2}$ then from Theorem 5 we get, for all $t \geq n\theta_1$, $\mathbb{P}\{\|C_t - L\|^2 \geq n/2\} \leq \delta/2$, or equivalently

$$\mathbb{P}\{\|C_t - L\|^2 < n/2\} \geq 1 - \delta/2.$$

Let us introduce the instant τ defined by

$$\tau = n\theta_1 + \frac{1600(n-1)}{189}\left(\ln n - \ln(\delta/2) - 4\ln 2 + \ln 3\right).$$

We have, for all $t \geq \tau$,

$$\begin{aligned}\mathbb{P}\{\|C_t - L\|_\infty < 3/2\} &\geq \mathbb{P}\{\|C_t - L\|_\infty < 3/2, \|C_{n\theta_1} - L\|^2 < n/2\} \\ &= \mathbb{P}\{\|C_t - L\|_\infty < 3/2 \mid \|C_{n\theta_1} - L\|^2 < n/2\} \\ &\quad \times \mathbb{P}\{\|C_{n\theta_1} - L\|^2 < n/2\}.\end{aligned}$$

We have seen that $\mathbb{P}\{\|C_{n\theta_1} - L\|^2 < n/2\} \geq 1 - \delta/2$. Using the fact that the Markov chain $\{C_t\}$ is homogeneous and applying Theorem 6, we obtain

$$\begin{aligned}\mathbb{P}\{\|C_t - L\|_\infty < 3/2 \mid \|C_{n\theta_1} - L\|^2 < n/2\} &= \mathbb{P}\{\|C_{t-n\theta_1} - L\|_\infty < 3/2 \mid \|C_0 - L\|^2 < n/2\} \\ &= \mathbb{P}\{\|C_{t-n\theta_1} - L\|_\infty < 3/2 \mid \|C_0 - L\| < \sqrt{n/2}\} \\ &\geq 1 - \delta/2.\end{aligned}$$

Putting together these two results gives, for all $t \geq \tau$,

$$\mathbb{P}\{\|C_t - L\|_\infty < 3/2\} \geq (1 - \delta/2)^2 \geq 1 - \delta$$

or equivalently

$$\mathbb{P}\{\|C_t - L\|_\infty \geq 3/2\} \leq \delta.$$

The rest of the proof consists in simplifying the expression of τ. We have

$$\begin{aligned}\theta_1 &= 2\ln K - \ln n + 3\ln 2 - \frac{2\ln 2}{2\ln 2 - \ln 3}\ln(\delta/2) \\ &= 2\ln K - \ln n + \left(4 + \frac{\ln 3}{2\ln 2 - \ln 3}\right)\ln 2 \\ &\qquad - \frac{2\ln 2}{2\ln 2 - \ln 3}\ln\delta\end{aligned}$$

and

$$\begin{aligned}\tau &= n\theta_1 + \frac{1600(n-1)}{189}(\ln n - \ln(\delta/2) - 4\ln 2 + \ln 3) \\ &= n\theta_1 + \frac{1600(n-1)}{189}(\ln n - \ln\delta - 3\ln 2 + \ln 3) \\ &\leq n\left[2\ln K + \frac{1411}{189}\ln n - \left(\frac{1789}{189} + \frac{\ln 3}{2\ln 2 - \ln 3}\right)\ln\delta \right. \\ &\qquad \left. - \left(\frac{1348}{63} - \frac{\ln 3}{2\ln 2 - \ln 3}\right)\ln 2 + \frac{1600}{189}\ln 3\right] \\ &\leq n\left(2\ln K + 7.47\ln n - 13.29\ln\delta - 2.88\right),\end{aligned}$$

which completes the proof. ∎

We now apply these results to compute the proportion γ_A of agents whose initial input was A, with $\gamma_A = n_A/(n_A + n_B) = n_A/n$. Recall that the output function ω_A is given, for all $x \in Q$, by

$$\omega_A(x) = (m + x)/(2m).$$

Theorem *8:* For all $\delta \in (0,1)$ and for all $\varepsilon \in (0,1)$, by setting $m = \lceil 3/(4\varepsilon)\rceil$, we have, for all $t \geq n\left(8.47\ln n - 2\ln\varepsilon - 13.29\ln\delta - 2.88\right)$,

$$\mathbb{P}\{|\omega_A(C_t^{(i)}) - \gamma_A| < \varepsilon \text{ for all } i = 1, \ldots, n\} \geq 1 - \delta.$$

Proof: We have $\|C_0 - L\| \leq m\sqrt{n}$. Applying Theorem 7, with $K = \sqrt{n}/\varepsilon \geq \lceil 3/(4\varepsilon)\rceil\sqrt{n} = m\sqrt{n}$, we obtain for all $\delta \in (0,1)$ and $t \geq n\left(8.47\ln n - 2\ln\varepsilon - 13.29\ln\delta - 2.88\right)$,

$$\mathbb{P}\{\|C_t - L\|_\infty \geq 3/2\} \leq \delta$$

or equivalently

$$\mathbb{P}\{|C_t^{(i)} - (\gamma_A - \gamma_B)m| < 3/2, \text{ for all } i = 1, \ldots, n\} \geq 1 - \delta.$$

Since $\gamma_A + \gamma_B = 1$ we have

$$\begin{aligned}|C_t^{(i)} - (\gamma_A - \gamma_B)m| &= |C_t^{(i)} - (2\gamma_A - 1)m| \\ &= |m + C_t^{(i)} - 2m\gamma_A| \\ &= 2m|\omega_A(C_t^{(i)}) - \gamma_A|.\end{aligned}$$

Then

$$\mathbb{P}\{|\omega_A(C_t^{(i)}) - \gamma_A| < 3/(4m), \text{ for all } i = 1, \ldots, n\} \geq 1 - \delta.$$

So

$$\mathbb{P}\{|\omega_A(C_t^{(i)}) - \gamma_A| < \varepsilon, \text{ for all } i = 1, \ldots, n\} \geq 1 - \delta,$$

which completes the proof. ∎

From Theorem 8, the convergence time to get the proportion γ_A of agents that were in the initial state A, with any precision ε and with any high probability $1 - \delta$ is $O\left(n(\log n - \log\varepsilon - \log\delta)\right)$ and thus the corresponding parallel convergence time is $O\left(\log n - \log\varepsilon - \log\delta\right)$. Still from Theorem 8, the size of the set of states to compute γ_A is equal to $2\lceil 3/(4\varepsilon)\rceil + 1$. It is important to note that the number of states does not depend, even logarithmically, in n.

VII. Lower Bounds

The second contribution of our paper is the derivation of lower bounds on a more general problem, namely the counting problem, introduced in [16]. This problem aims, for each agent, at computing the exact number of agents that started in the initial state A. Using the interaction rules given in Relation (1) and the output function

$$\omega'_A(x) = \lfloor n(m + x)/(2m) + 1/2\rfloor,$$

we can exploit the results derived in the present paper to show that the counting problem can be solved with $O(n)$ states, improving upon [16] in which the number of states is in $O(n^{3/2})$. We show that $O(n)$ states and $O(\log n)$ parallel time are lower bounds to solve the counting problem.

Finally, we prove that any algorithm solving the proportion problem with a precision $\varepsilon \in (0,1)$, requires $\Omega(1/\varepsilon)$ states. This demonstrates that our proportion protocol is optimal in the number of states.

Theorem *9:* By setting $m = \lceil 3n/2\rceil$, for all $\delta \in (0,1)$ and for all $t \geq n\left(10.47\ln n - 13.29\ln\delta - 1.49\right)$, we have

$$\mathbb{P}\{\omega'_A(C_t^{(i)}) = n_A, \text{ for all } i = 1, \ldots, n\} \geq 1 - \delta.$$

Proof: Observe that we have

$$\omega'_A(x) = \lfloor n\omega_A(x) + 1/2\rfloor.$$

Applying Theorem 8 with $\varepsilon = 1/(2n)$ and for $t \geq n\left(10.47\ln n - 13.29\ln\delta - 1.49\right)$, we obtain

$$\mathbb{P}\{|n\omega_A(C_t^{(i)}) - n\gamma_A| < 1/2 \text{ for all } i = 1, \ldots, n\} \geq 1 - \delta.$$

Since $n\gamma_A = n_A$ is an integer, we get

$$\mathbb{P}\{\omega'_A(C_t^{(i)}) = n_A, \text{ for all } i = 1, \ldots, n\} \geq 1 - \delta,$$

which completes the proof. ■

Thus each agent can solve the counting problem in $O(\log n)$ parallel time and with $O(n)$ states.

Theorem *10:* Any algorithm solving the counting problem takes an expected $\Omega(\log n)$ parallel time to convergence.

Proof: Solving the counting problem bounds to solving the exact majority problem. By applying Theorem C.1 of [2], this algorithm takes an expected $\Omega(\log n)$ parallel time to convergence under a worst-case input. ■

Theorem *11:* Any algorithm solving the counting problem requires $\Omega(n)$ states.

Proof: To solve the counting problem, the size of the output set Y must be $n+1$. So, the number of states (*i.e.* $|Q|$) is at least $n+1$. The lower bound of the number of states is thus $\Omega(n)$. ■

Theorem *12:* Any algorithm solving the proportion problem with a precision $\varepsilon \in (0,1)$, requires $\Omega(1/\varepsilon)$ states.

Proof: The value of γ_A could be any rational value between 0 and 1, the difference between two output values cannot exceed 2ε, thus the lower bound for the size of the output Y is $\lceil 1/(2\varepsilon) \rceil + 1$. Hence, the number of states (*i.e.*, $|Q|$) is at least $\lceil 1/(2\varepsilon) \rceil + 1$. Thus the lower bound of the number of states is $\Omega(1/\varepsilon)$. ■

VIII. Simulation results

We have conducted simulations to illustrate our theoretical analysis. Figure 1 provides a summary of these simulations. In this figure, each point of the curves represents the mean of 100 simulations (with the maximum and the minimum of the 100 simulations), a simulation consisting in computing the total number of interactions, divided by n, needed for all the agents to converge to γ_A with precision ε. The number n of agents varies from 2^5 to 2^{22}, and the precision ε of the result is set to 10^{-1}, 10^{-3}, and 10^{-5}. Note that as shown theoretically, Figure 1(a) and Figure 1(b) illustrate the fact that the number of interactions per agent to converge is independent of the value of γ, that is independent from the difference between both proportions. From the generated data, for instance when $\delta = 1/2$, one can deduce for each curve an empirical approximation of the convergence parallel time given by $-2\ln\varepsilon + 0.62\ln n - 0.6$.

IX. Conclusion

This paper has shown that in a large-scale system, any agent can compute quickly and with a high precision specified in advance the proportion of agents that initially started in some given input state. This problem is a generalization of the majority problem. Specifically, our protocol guarantees that by using $2\lceil 3/(4\varepsilon) \rceil + 1$ states, any agent is capable of computing the population proportion with precision $\varepsilon \in (0,1)$, in no more than $(-2\ln\varepsilon + 8.47\ln n - 13.29\ln\delta - 2.88)$ interactions with probability at least $1-\delta$, for any $\delta \in (0,1)$. We have also shown that our solution is optimal both in time and space. As future work, we aim at using the same detailed analysis to obtain new results for the majority problem.

References

[1] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald Rivest. Time-space trade-offs for population protocols. In *arXiv:1602.08032v1*, 2016.

[2] Dan Alistarh, Rati Gelashvili, and Milan Vojnovíc. Fast and exact majority in population protocols. In *Proceedings of the 34th annual ACM symposium on Principles of Distributed Computing (PODC)*, pages 47–56, 2015.

[3] Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In *Proceedings of the 34th annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2015.

[4] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

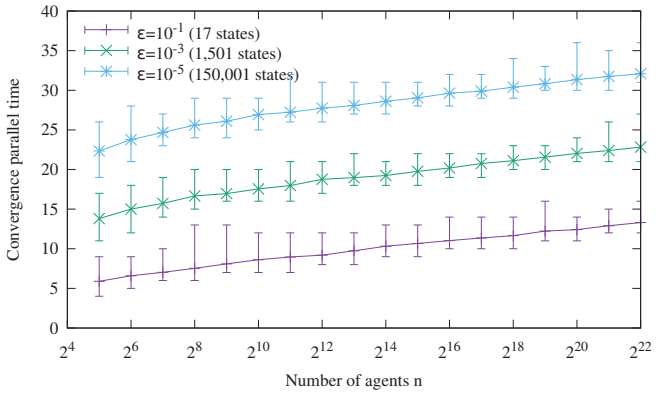
[5] Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the 25th annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 292–299, 2006.

[6] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 20(4):279–304, 2008.

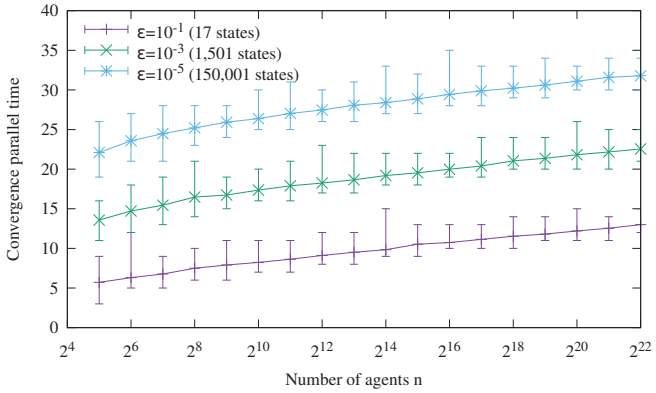
[7] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[8] James Aspnes and Eric Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science, Distributed Computing Column*, 93:98–117, 2007.

[9] Joffroy Beauquier, Peva Blanchard, and Janna Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *Proceedings of the 17th International Conference on Principles of Distributed Systems (OPODIS)*, 2013.



(a) $\gamma = 0$ (*i.e* $\gamma_A = \gamma_B$=1/2)



(b) $\gamma = 1/2$ (*i.e* $\gamma_A = 3/4$ and $\gamma_B = 1/4$)

Figure 1. Number of interactions per agent as a function of the size of the system.

[10] Olivier Bournez, Cohen Johanne, and Mikaël Rabie. Homonym population protocols. In *Proceedings of the 3rd International Conference on Networked Systems (NETYS)*, 2015.

[11] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Eric Ruppert. When birds die: Making population protocols fault-tolerant. In *Proceedings of the 2nd IEEE Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 51–66, 2006.

[12] Moez Draief and Milan Vojnovic. Convergence speed of binary interval consensus. *SIAM Journal on Control and Optimization*, 50(3):1087–11097, 2012.

[13] Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part II (ICALP)*, pages 484–495, 2009.

[14] George B. Mertzios, Sotiris E. Nikoletseas, Christoforos Raptopoulos, and Paul G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *Proceedings of the 41st International Colloquium (ICALP)*, pages 871–882, 2014.

[15] Ryu Mizoguchi, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. On space complexity of self-stabilizing leader election in mediated population protocol. *Distributed Computing*, 25(6):451–460, 2012.

[16] Yves Mocquard, Emmanuelle Anceaume, James Aspnes, Yann Busnel, and Bruno Sericola. Counting with population protocols. In *Proceedings of the 14th IEEE International Symposium on Network Computing and Applications*, pages 35–42, 2015.

[17] Yves Mocquard, Emmanuelle Anceaume, and Bruno Sericola. Optimal Proportion Computation with Population Protocols. Technical report, August 2016. `https://hal.archives-ouvertes.fr/hal-01354352`

[18] Etienne Perron, Dinkar Vasudevan, and Milan Vojnovic. Using three states for binary consensus on complete graphs. In *Proceedings of the INFOCOM Conference*, pages 2527–2435, 2009.

CoVer-ability: **Co**nsistent **Ver**sioning in Asynchronous, Fail-Prone, Message-Passing Environments

Nicolas Nicolaou*, Antonio Fernández Anta*, Chryssis Georgiou†
* IMDEA Networks Institute, Madrid, Spain, nicolas.nicolaou@imdea.org,antonio.fernandez@imdea.org
† Dept. of Computer Science, University of Cyprus, Nicosia, Cyprus, chryssis@cs.ucy.ac.cy

***Abstract*—An *object type* characterizes the domain space and the operations that can be invoked on an object of that type. In this paper we introduce a new property for concurrent objects, we call *coverability*, that aims to provide precise guarantees on the consistent evolution of the *version* (and thus value) of an object. This new property is suitable for a variety of distributed objects, including *concurrent file objects*, that demand operations to manipulate the latest version of the object. To preserve the order of versions, traditional approaches use locking, compare-and-swap (CAS), or linked-load/conditional-store (LL/SC) primitives to allow a single modification at a time on such objects. Such primitives however can be used to solve consensus, and thus are impossible to be implemented in an asynchronous, message-passing environment with failures.**

***Coverability*, relaxes the strong requirements imposed by stronger primitives, and allows us to define and implement consistent versioning in the aforementioned adversarial environment. In particular, coverability allows *multiple operations* to modify the same version of an object concurrently, leading to a set of different versions. Given an order of operations, coverability properties specify a single version in that set that any subsequent operation may modify, preserving this way the consistent evolution of the object. We first define *versioned* objects and then provide the specification of *coverability*. We then combine coverability with atomic guarantees to yield *coverable atomic read/write registers*; we show that coverable registers cannot be implemented by similar types of registers, such as ranked-registers. Next, we show how coverable registers may be implemented by modifying an existing MWMR atomic register implementation, and we continue by showing that coverable registers may be used to implement basic (weak) read-modify-write and file objects.**

I. Introduction

Motivation and Prior Work. A concurrent system allows multiple processes to interact with a single object at the same time. A long string of research work [2], [6], [15]–[17] has been dedicated to explain the behavior of concurrent objects, defining the order and the outcomes of operations when those are invoked concurrently on the object. Lamport in [16], [17] presented three different incremental semantics, *safety*, *regularity*, and *atomicity* that characterize the behavior of read/write objects (registers) when those are modified or read concurrently by multiple processes. The strongest, and most difficult to provide in a distributed system, is *atomicity* which provides the illusion that the register is accessed sequentially. Herlihy and Wing presented *linearizability* in [15], an extension of atomicity to general concurrent objects. More recent developments have proposed abortable operations in the event of concurrency [2], and ranked registers [6] that allow operations to abort in case a higher "ranked" operation was previously or concurrently executed in the system.

Although consistency semantics strictly specify the "placement" of events on an execution trace based on their timing characteristics, in many cases they are oblivious of the state of the object at the point when an event takes effect. For example, a write operation ω on a read/write register is ordered after all the writes that completed before ω, irrespective to the value that ω writes on the register. With the advent of cloud computing, emerging families of more complex concurrent objects, like files, distributed databases, and bulleting boards, demand precise guarantees on the consistent evolution of the object. For example, in *concurrent file objects* one would expect that if a write operation ω_2 is invoked after a write operation ω_1 is completed, then ω_2 modifies either the version of the file written by ω_1 or a version of the file newer than the one written by ω_1. Such guarantees are easy to achieve in systems that readily provide atomic compare-and-swap (CAS), or linked-load/conditional-store (LL/SC) operations. Such primitives allow modify operations to atomically obtain the current version and value of an object, modify both, and store the new version along with the new value of the object. As shown by Herlihy in [14], CAS can be used to solve consensus as it has a consensus number infinite. However, as shown by Fischer, Lynch and Paterson [11], solving consensus in an asynchronous, message-passing, fail-prone environment is impossible in the existence of a single crash failure. So the main question we will try to address is: *Can we provide versioning guarantees in an asynchronous, message-passing, fail-prone environment using weaker primitives, like read/write registers?*

A seminal work by Attiya, Bar Noy and Dolev [3], demonstrated that it is possible to introduce atomic read/write

This work is supported in part by FP7-PEOPLE-2013-IEF grant ATOMICDFS No:629088, Ministerio de Economia y Competitividad grant TEC2014- 55713-R, Regional Government of Madrid (CM) grant Cloud4BigData (S2013/ICE-2894, co- funded by FSE & FEDER), NSF of China grant 61520106005.

978-1-5090-3216-7/16/$31.00

registers in an asynchronous, message-passing environment where processes may fail. As noted before, in existing atomic read/write register implementations, write operations are allowed to modify the value of the register, even when they are unaware of the value written by the latest preceding write operation. In systems that assume a single writer [3], [8], [12], [13], the problem may be diminished by having the sole writer compute the next value to be written in relation to the previous values it wrote. The problem becomes more apparent when multiple writers may alter the value of a single register concurrently [9], [20]. In such cases, atomic read/write register implementations appear unsuitable to directly implement objects that demand evolution guarantees. Closer candidates to build such objects are the bounded [4] and ranked [6] registers. These objects take into account the "rank" or sequence number of previous operations to decide whether to allow a read/write operation to commit or abort. These approaches do not prevent, however, the use of an arbitrarily higher rank, and thus an arbitrarily higher version, than the previous operations. This affects the consistent evolution of the object, as intermediate versions of the object maybe ignored.

Contributions. In this paper we propose a formalism to extend a concurrent object in such a way that the evolution of its state satisfies certain guarantees. To this end, we extend an object state with a *version,* and introduce the concept of *coverability,* that defines how the versions of an object can evolve (Section III).

In particular, we first introduce a new class of a concurrent read/write register type, which we call *versioned register*. A concurrent register is of a *versioned* type if the state of the register, and any operation (read or write) that attempts to modify the state of the register, are associated with a *version*. An operation may modify the state and the version of the register, or it may just retrieve its state-version pair.

Coverability defines the exact guarantees that a versioned register provides when it is accessed concurrently by multiple processes with respect to the evolution of its versions. Coverability allows *multiple operations* to change a version, generating in this way a *tree* with possibly multiple version branches that can grow in parallel. This shares similarities with *fork linearizability* presented in [21]. However, in contrast to [21], coverability allows processes that change the same version of the object to see the changes of each other in subsequent operations. In particular, by coverability, when all the operations that extend a particular version of the object terminate, there is one version ver that was generated by one of those operations, which is the ancestor of any version extended by any subsequent operation. Thus, only a single branch in the tree is extended and that branch denotes the evolution of the register. The rest of the branches are discarded. This resembles the way that the forks in a bitcoin blockchain converge [1]. In particular, forks in a blockchain are created when two miners generate a new block concurrently. Both blocks are legitimate and each miner results in a different branch, rooted from the same blockchain. Miners tend to quickly converge on one chain and discard the other because of profit-related motives. These discarded chains are usually only one block long and are considered a statistical loss. In contrast to the "profit-related" motives of the bitcoin, coverability specifies which of the branches need to be discarded based on a provable ordering of the events. Notice that the stronger form of coverability where modify operations are totally ordered, avoids branching of the versions. However such primitive is equivalent with strong primitives like CAS and LL/SC, and thus it is as powerful as consensus (details can be found in [22]). Hence, it is challenging to implement strong coverability in some distributed systems, and impossible in an asynchronous system prone to failures (from the FLP result [11]).

An interesting property of coverability is that it is defined over a given order of events. Therefore coverability can be defined over the ordering yielded by any consistency scheme. In this paper we combine coverability with atomic guarantees and we obtain *coverable atomic read/write registers*. Coverable atomic registers have very interesting features. At first, they provide strong atomic guarantees, i.e they surpass weaker consistency guarantees like regularity [16], or eventual consistency [23], and in addition provide guarantees on the evolution of the value of the register. This allows coverable atomic registers to be used for the implementation of more complex objects like: (i) interesting *weak read-modify-write (RMW) objects* which in turn can be used to implement (ii) *file objects* (Section VI). Furthermore, we show they cannot be implemented using similar register types such as ranked registers (Section IV). And last but not least, they can be implemented in message passing asynchronous distributed systems where processes can fail, with a simple modification of existing atomic read/writer register implementations (Section V).

II. Model

We consider a distributed system composed of n *asynchronous* processes, with identifiers from a set $\mathcal{I} = \{p_1, \ldots, p_n\}$, that communicate by exchanging messages. A subset of processes in $\mathcal{I}$ may fail by *crashing*.

Processes can be modeled in terms of I/O Automata [19]. An automaton A (which combines the automata A_i for each process $p_i \in \mathcal{I}$) is defined over a set of *states* and a set of *actions*. An *execution* ξ of A is an alternating sequence of *states* and *actions* of A. An *execution fragment* is a finite prefix of an execution. We say that an execution fragment ξ' *extends* an execution fragment ξ, if ξ is a prefix of ξ'. A *history* of an automaton A, denoted by H_ξ, is the subsequence of actions occurring in some execution fragment ξ of A. An automaton A *invokes* an operation when an *invocation action* occurs in an execution ξ, and receives a *response* to an action when a *response action* occurs. An operation π is *complete* in an execution ξ, if H_ξ contains both the invocation and the matching response actions for π; otherwise π is *incomplete*. A history H_ξ of the automaton A_i of a process p_i is *well formed* if it begins with an invocation event and alternates between matching invocation and response events. (This demonstrates

the assumption that each process is a single thread of control.) Each history H_ξ includes a precedence relation $\rightarrow_{H_\xi}$ on its operations. An operation π_1 *precedes* an operation π_2 (or π_2 *succeeds* π_1) in H_ξ if the response of π_1 appears before the invocation of π_2 in H_ξ. This is denoted by $\pi_1 \rightarrow_{H_\xi} \pi_2$. If $\pi_1 \not\rightarrow_{H_\xi} \pi_2$ and $\pi_2 \not\rightarrow_{H_\xi} \pi_1$ in H_ξ, then π_1 and π_2 are *concurrent*. A process p_i *crashes* in an execution ξ if the event fail_{p_i} appears and is the last action of p_i in H_ξ; otherwise p_i is *correct*.

III. Coverable Atomic Read/Write Registers

In this section we define a new type of read/write (R/W) register, the *versioned register*. Next we provide a new consistency property for concurrent versioned registers called *coverability*. We show how coverability can be combined with atomic guarantees to yield a coverable atomic register.

Versioned register. Let $Versions$ be a *totally ordered* set of *versions*. A *versioned register* is a type of R/W register where each value written is assigned with a version from the set $Versions$. Moreover, each write operation π that attempts to change the value of the register is also associated with a version, say ver_π, denoting that it intends to overwrite the value of the register associated with the version ver_π. More precisely, an implementation of a R/W register offers two operations: *read* and *write*. A process $p_i \in \mathcal{I}$ *invokes* a *write* (resp. *read*) operation when it issues a $\mathsf{write}(val)_{p_i}$ (resp. read_{p_i}) request. The *versioned* variant of a R/W register also offers two operations: (i) $\mathsf{cvr\text{-}write}(val, ver)_{p_i}$, and (ii) $\mathsf{cvr\text{-}read}()_{p_i}$. A process p_i invokes a $\mathsf{cvr\text{-}write}(val, ver)_{p_i}$ operation when it performs a write operation that attempts to change the value of the object. The operation returns the value of the object and its associated version, along with a flag informing whether the operation has successfully changed the value of the object or failed. We say that a write is *successful* if it changes the value of the register; otherwise the write is *unsuccessful*. The read operation $\mathsf{cvr\text{-}read}()_{p_i}$ involves a request to retrieve the value of the object. The response of this operation is the value of the register together with the version of the object that this value is associated with.

Read operations do not incur any change on the value of the register, whereas write operations attempt to modify the value of the register. More formally, let Δ_T be the set of transitions for the versioned register. Then, each $\delta \in \Delta_T$ is a tuple $\langle \sigma, \pi, p_i, \sigma', res \rangle$, denoting that the register moves from state σ to state σ', and responds with res, as a result of operation π invoked by process $p_i \in \mathcal{I}$. The state of a versioned register is essentially its *value*, drawn from a set V, and its *version*, drawn from the set $Versions$. We assume that Δ_T is *total*, that is, for every $\pi \in \{\mathsf{cvr\text{-}write}(val, ver)_{p_i}, \mathsf{cvr\text{-}read}()_{p_i}\}$, $p_i \in \mathcal{I}$, and $\sigma = (val, ver) \in V \times Versions$, there exists $\sigma' = (val', ver') \in V \times Versions$ and res such that $\langle \sigma, \pi, p_i, \sigma', res \rangle \in \Delta_T$. As such, the transitions of the versioned register type can be written as follows:

1) $\langle (val, ver), \mathsf{cvr\text{-}write}(val', ver_\omega), p_i, (val', ver'), (val', ver', chg) \rangle$, for $ver_\omega = ver$,
2) $\langle (val, ver), \mathsf{cvr\text{-}write}(val', ver_\omega), p_i, (val, ver), (val, ver, unchg) \rangle$, for $ver_\omega \neq ver$
3) $\langle (val, ver), \mathsf{cvr\text{-}read}(), p_i, (val, ver), (val, ver) \rangle$.

Notice that write operations may or may not modify the value/version of the register. In the transitions above, ver_ω denotes the version of the register which the write operation tries to modify. The relationship of ver with ver' may vary depending on the application that uses this register (but seems natural to assume that $ver' > ver$). A read operation does not make any changes on the value or the version of the object. To simplify notation, in the rest of the paper we avoid any reference to the value of the register. Additionally we only use the flag when its value is $unchg$. Thus, $\mathsf{cvr\text{-}write}(v, ver)(v, ver', chg)_{p_i}$ is denoted as $cvr\text{-}\omega(ver)[ver']_{p_i}$, and $\mathsf{cvr\text{-}write}(v, ver)(v', ver', unchg)_{p_i}$ is denoted as $cvr\text{-}\omega(ver)[ver', unchg]_{p_i}$.

We say that, a write operation *revises* a version ver of the versioned register to a version ver' (or *produces* ver') in an execution ξ, if $cvr\text{-}\omega(ver)[ver']_{p_i}$ completes in H_ξ. Let the set of *successful write* operations on a history H_ξ be defined as:

$$\mathcal{W}_{\xi,succ} = \{\pi : \pi = cvr\text{-}\omega(ver)[ver']_{p_i} \text{ completes in } H_\xi\}.$$

The set now of produced versions in the history H_ξ is defined by:

$$Versions_\xi = \{ver_i : cvr\text{-}\omega(ver)[ver_i]_{p_i} \in \mathcal{W}_{\xi,succ}\} \cup \{ver_0\}$$

where ver_0 is the initial version of the object. Observe that the elements of $Versions_\xi$ are totally ordered. In the rest of the text we use '$*$' in the place of some parameter to denote that any legal value for that parameter can be used. Now we present the *validity* property which defines explicitly the set of executions that are considered to be valid executions.

Definition 1 (Validity): An execution ξ (resp. its history H_ξ) is a *valid execution* (resp. history) on a versioned object, for any $p_i, p_j \in \mathcal{I}$:

- $\forall cvr\text{-}\omega(ver)[ver']_{p_i} \in \mathcal{W}_{\xi,succ}, ver < ver'$,
- for any operations $cvr\text{-}\omega(*)[ver']_{p_i}$ and $cvr\text{-}\omega(*)[ver'']_{p_j}$ in $\mathcal{W}_{\xi,succ}$, $ver' \neq ver''$, and
- for each $ver_k \in Versions_\xi$ there is a sequence of versions $ver_0, ver_1, \ldots, ver_k$, such that $cvr\text{-}\omega(ver_i)[ver_{i+1}] \in \mathcal{W}_{\xi,succ}$, for $0 \leq i < k$.

Validity makes it clear that an operation changes the version of the object to a larger version, according to the total ordering of the versions. Also validity specifies that versions are *unique*, i.e. no two operations associate two states with the same version. This can be easily achieved by, for example, recording a counter and the id of the invoking process in the version of the object. Finally, validity requires that each version we reach in an execution is *derived* (through a chain of operations) from the initial version of the register ver_0. From this point onward we fix ξ to be a valid execution and H_ξ to be its valid history.

Coverability. We can now define the *coverability* properties over a valid execution ξ of versioned registers with respect to some total order $>_\xi$ on the operations of ξ.

Definition 2 (Coverability): A valid execution ξ is **coverable** with respect to a total order $<_\xi$ on operations in $\mathcal{W}_{\xi,succ}$ if:

- (**Consolidation**) If $\pi_1 = cvr\text{-}\omega(*)[ver_i], \pi_2 = cvr\text{-}\omega(ver_j)[*] \in \mathcal{W}_{\xi,succ}$, and $\pi_1 \rightarrow_{H_\xi} \pi_2$ in H_ξ, then $ver_i \leq ver_j$ and $\pi_1 <_\xi \pi_2$.
- (**Continuity**) if $\pi_2 = cvr\text{-}\omega(ver)[ver_i] \in \mathcal{W}_{\xi,succ}$, then there exists $\pi_1 \in \mathcal{W}_{\xi,succ}$ s.t. $\pi_1 = cvr\text{-}\omega(*)[ver]$ and $\pi_1 <_\xi \pi_2$, or $ver = ver_0$.
- (**Evolution**) let $ver, ver', ver'' \in Versions_\xi$. If there are sequences of versions $ver'_1, ver'_2, \ldots, ver'_k$ and $ver''_1, ver''_2, \ldots, ver''_\ell$, where $ver = ver'_1 = ver''_1$, $ver'_k = ver'$, and $ver''_\ell = ver''$ such that $cvr\text{-}\omega(ver'_i)[ver'_{i+1}] \in \mathcal{W}_{\xi,succ}$, for $1 \leq i < k$, and $cvr\text{-}\omega(ver''_i)[ver''_{i+1}] \in \mathcal{W}_{\xi,succ}$, for $1 \leq i < \ell$, and $k < \ell$, then $ver' < ver''$.

Intuitively, *Consolidation* specifies that write operations may revise the register with a version larger than any version modified by a preceding write operation, and may lead to a version newer than any version introduced by a preceding write operation. *Continuity* defines that a write operation may revise a version that was introduced by a preceding write operation according to the given total order. Finally, *Evolution* limits the relative increment on the version of a register that can be introduced by any operation.

By Definition 2, coverability allows multiple write operations to revise the same version ver_i of the register, each to a *unique* version ver_j. Given the set of successful operations $\mathcal{W}_{\xi,succ}$ and the set of versions $Versions_\xi$, Definitions 1 and 2 define a connected rooted tree $\mathcal{T}$ s.t.:

- The set of nodes of $\mathcal{T}$ is $Versions_\xi$,
- The initial version ver_0 of the object is the root of $\mathcal{T}$,
- A node ver_i is the parent of a node ver_j in $\mathcal{T}$ iff $\exists \pi(ver_i)[ver_j] \in \mathcal{W}_{\xi,succ}$,
- If $\pi_1 = cvr\text{-}\omega(*)[ver_i] \in \mathcal{W}_{\xi,succ}$, s.t. π_1 is not concurrent with any other operation, then $\forall \pi_2 \in \mathcal{W}_{\xi,succ}$, s.t. $\pi_1 \rightarrow_\xi \pi_2$ and $\pi_2 = \pi(ver_z)[*]$, then ver_i is an ancestor of ver_z in $\mathcal{T}$, or $ver_i = ver_z$ (by Consolidation, Continuity, and Validity)
- if ver_i is an ancestor of ver_j in $\mathcal{T}$, then $cvr\text{-}\omega(*)[ver_i] <_\xi cvr\text{-}\omega(*)[ver_j]$ (by Continuity).
- if ver_i is at level k of $\mathcal{T}$ and ver_j is at level ℓ of $\mathcal{T}$ s.t. $k < \ell$, then $ver_i < ver_j$ (by Evolution).

Observe that without the properties imposed by coverability, validity allows the creation of a tree of versions and does not prevent operations from being applied on an old version of the register. *Continuity*, *Consolidation*, and *Evolution* explicitly specify the conditions that reduce the branching of the generated tree, and in the case of not concurrency lead the operations to a single path on this tree. Figure 1 provides an illustration of a tree created from a coverable execution ξ. We box sample instances of the execution and we indicate the coverability properties they satisfy.

Atomic coverability. We now combine coverability with atomic guarantees to obtain coverable atomic read/write registers. A register is linearizable [15], or equivalently *atomic* (as defined specifically for registers by [17], [18]) if the following conditions are satisfied by any execution ξ of an implementation of the object.

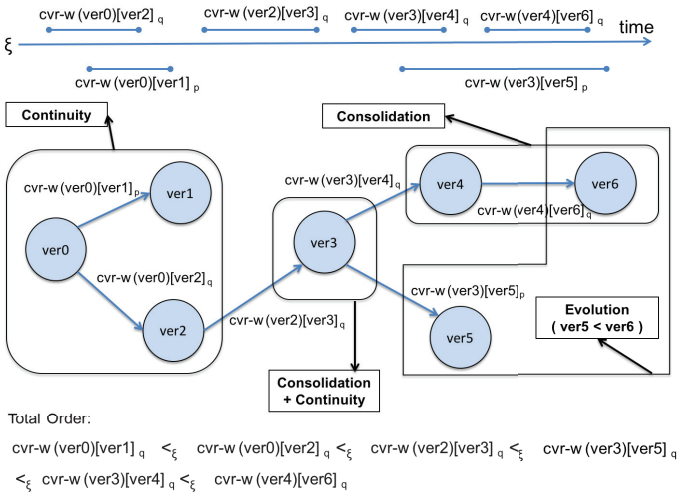


Fig. 1: Tree Illustration from Coverable Execution

Definition 3 (Atomicity): [18, Section 13.4] An execution ξ of an automaton A is *atomic* if every *read* and *write* operation in ξ is *complete* and there is a partial ordering $\prec_{H_\xi}$ on all operations Π in H_ξ such that: **A1.** For any pair of operations $\pi_1, \pi_2 \in \Pi$, if $\pi_1 \rightarrow_{H_\xi} \pi_2$ then it cannot hold that $\pi_2 \prec_{H_\xi} \pi_1$, **A2.** If $\pi \in \Pi$ is a *write* operation and π' any operation in Π, then either $\pi \prec_{H_\xi} \pi'$ or $\pi' \prec_{H_\xi} \pi$, and **A3.** If v is the value returned by a *read* ρ then v is the value written by the last preceding *write* according to $\prec_{H_\xi}$ (or the initial value v_0 if there is no such a write).

In the context of versioned registers, in Definition 3, a *write* refers to a successful write $(cvr\text{-}\omega(*)[*, chg])$ operation on the versioned register. Therefore, all the write operations in an execution ξ are the ones that appear in $\mathcal{W}_{\xi,succ}$. A *read* refers to a versioned read $(cvr\text{-}\rho()[*])$ or an unsuccessful write $(cvr\text{-}\omega(*)[*, unchg])$ operation that does not modify the value (nor the version) of the register.

Definition 4 (Coverable atomic register): A versioned register is **coverable** and **atomic**, referred as *coverable atomic register*, if any execution ξ on the register satisfies: (i) atomicity (Definition 3), and (ii) coverability (Definition 2) with respect to the total order imposed by **A2** on $\mathcal{W}_{\xi,succ}$.

Note that in a coverable atomic register, the ordering of read operations follows the ordering from atomicity. From this point onward, when clear from context, we refer to a coverable atomic register, as simply *coverable register*.

IV. Coverable Atomic Registers vs Ranked Registers.

A type of registers that at first might resemble coverable registers are *ranked-registers* [6]. As we show here, ranked-registers are weaker than coverable registers. In particular, we

show that it is impossible to implement coverable registers using ranked-registers; we begin by providing a formal definition of ranked-registers.

Definition 5 (Ranked-Registers [6]): Let $Ranks$ be a totally ordered set of ranks with r_0 the initial rank. A ranked register is a MWMR shared object that offers the following operations: (i) rr-read(r), with $r \in Ranks$ and returns $(r, v) \in Ranks \times Values$, and (ii) rr-write$(\langle r, v\rangle)$, with $(r, v) \in Ranks \times Values$ and returns $commit$ or $abort$. A ranked register satisfies the following properties: (i) **Safety.** Every rr-read operation returns a value and a rank that was written in some rr-write invocation or (r_0, v_0). Additionally, if $W =$ rr-write$(\langle r_1, v\rangle)$ a write operation which commits and $R =$ rr-read(r_2) such that $r_2 > r_1$, then R returns (r, v) where $r \geq r_1$. (ii) **Non-Triviality.** If a rr-write operation W invoked with a rank r_1 aborts, then there exists an operation with rank $r_2 > r_1$ which returns before W is invoked, or is concurrent with W (iii) **Liveness.** if an operation is invoked by a correct process then eventually it returns.

We want to use rank-registers to implement the operations of a coverable register. As in Section II, we denote by $cvr\text{-}\omega(ver)[ver', flag]$ the coverable write operation that tries to revise version ver, and returns version ver' with a $flag \in \{chg, unchg\}$. Similarly we denote by $rr\text{-}\omega(r)[r_h, res]$ a write operation on a ranked-register that uses rank r and tries to modify the value of the register. The rank r_h is the highest rank observed by an operation and $res \in \{abort, commit\}$. In the following results we assume that a coverable register is implemented using a set of ranked-registers. We begin with a lemma that shows that a coverable write operation revises the coverable register only if it invokes a write operation on some rank register and that write operation commits. *Omitted proofs can be found in [22].*

Lemma 6: Suppose there exists an algorithm A that implements a coverable register using ranked-registers. In any execution ξ of A, if a process p_i invokes a coverable write operation $cvr\text{-}\omega(ver)[ver', chg]_{p_i}$, then p_i performs a write $rr\text{-}\omega(r)[r_h, commit]_{p_i,j}$ on some shared ranked-register j.

Next we show that if π_1, π_2 are two non-concurrent write operations on the coverable register, then π_2 performs a ranked write (that commits or aborts) on at least a single ranked register on which π_1 performed a committed ranked write operation. For the sake of the lemma R_i is the set of ranked registers on which π_i writes, and cR_i a subset of them on which the write commits.

Lemma 7: Let $\pi_1 = cvr\text{-}\omega(ver)[ver_1, chg]_{p_i}$ and $\pi_2 = cvr\text{-}\omega(ver_1)[ver_2, *]_{p_z}, i \neq z$, be two write operations that appear in an execution ξ s.t. $\pi_1 \rightarrow_\xi \pi_2$. There exists some shared register $j \in R_2 \cap cR_1$ with a highest rank r_j before the invocation of π_1, such that p_i performs an $rr\text{-}\omega(r)[*, commit]_{p_i,j}$ during π_1, and p_z performs an $rr\text{-}\omega(r')[*, *]_{p_z,j}$ during π_2.

Thus far we showed that a successful coverable write operation needs to commit on at least a single ranked register (Lemma 6), and two non-concurrent coverable write operations need to invoke a ranked write operation on a common rank register (Lemma 7). Using now Lemma 7 we can show that a coverable write operation that changes the version of the coverable register must use a rank higher than any previously successful coverable write operation.

Lemma 8: In any execution ξ if $\pi_1 = cvr\text{-}\omega(ver)[ver_1, chg]_{p_i}$ and $\pi_2 = cvr\text{-}\omega(ver_1)[ver_2, chg]_{p_z}$, $z \neq i$, s.t. $\pi_1 \rightarrow_\xi \pi_2$, then there exists some shared register j such that p_i performs an $rr\text{-}\omega(r)[*, commit]_{p_i,j}$ during π_1, and p_z performs an $rr\text{-}\omega(r')[*, commit]_{p_z,j}$ during π_2, and $r' > r$.

Now we prove our main result stating that a coverable register cannot be implemented with ranked registers as those were defined in [6].

Theorem 9: There is no algorithm that implements a coverable register using a set of ranked registers.

Proof: The theorem follows from Lemmas 6, 7, and 8, and the fact that a ranked register allows a write operation to commit even if it uses a rank smaller than the highest rank of the register. As by Lemma 6 a successful write must commit, then by ranked registers it can commit with a rank smaller than the highest rank of the accessed register. This, however, by Lemma 8 may lead to violation of the consolidation and continuity properties, and thus violation of coverability. ■

Observe that the key fact that makes ranked registers weaker than coverable registers is that the former allow write operations to commit even if their ranks are out of order. In particular, note that the Non-Triviality property *does not force* a write operation invoked with a rank r_1 to abort, even if there exists a completed prior operation with rank $r_2 > r_1$. As shown in [6] *non-fault-tolerant* ranked registers may preserve the total order of the ranks, and thus be used to implement consensus. As we show in [22] such ranked registers (i.e., that implement consensus) could be used to implement strongly coverable registers.

V. Implementing Coverable Atomic Read/Write Registers

We now show how we can implement coverable atomic registers. We do so by enhancing the Multi-Writer version of algorithm ABD [3], [20] (referred as MWABD) to preserve the properties of coverability. The presented technique can be applied to implementations of atomic R/W objects that utilize a $\langle tag, value\rangle$ pair to order the write operations and where each write performs two phases before completing: a *query phase* to obtain the latest value of the atomic object and a *propagation phase* to write the new value on the object. We could also adopt implementations of stronger objects like the ones presented in [4]–[7] but we preferred to show the simplest modification in a fundamental algorithm. To capture the semantics of a coverable atomic register we modify the operations of algorithm MWABD to comply with the versioned variant of the R/W register. We use cvr-write(ver, v) and cvr-read$()$ as the write and read operations respectively. A cvr-write(ver, v) operation may impact differently the state of the object, depending on the version of the shared object: it may appear as a *read*, not modifying the value nor the version

of the register, or as a *write*, changing both the value and the version of the register.

In brief, the original MWABD replicates an object to a set of hosts (replicas) $\mathcal{S} \subset \mathcal{I}$ and it uses $\langle tag, value \rangle$ pairs to order the *read* and *write* operations. A tag consists of a *non-negative integer* and a *writer identifier* which is used to break the ties among concurrent write operations. Both the read and write protocols have two phases: a *query* and a *propagation* phase. During the *query* phase the invoking process broadcasts a query message to all the replicas and waits for a majority of them to reply with their tag-value pairs. Once those replies are received the process discovers the largest tag-value pair among the replies. In the second phase, a read operation propagates the discovered tag-value pair to the majority of the replicas. A write operation increments the largest tag, associates the new tag with the value to be written, and propagates the new tag-value pair to the majority of the replicas.

In the *versioned* MWABD, vMWABD for short, we use the tags associated with each value to denote the version of the register. The pseudocode of each operation of vMWABD is described in Figure 2. The cvr-read operation is similar to the read operation of MWABD with the difference that it returns both the value and the version of the register. A cvr-write operation differs from the original write by utilizing a condition before its *propagation* phase and depending whether the condition holds it changes the state of the register (value and version) or not, as detailed in Figure 2. Note that the version parameter of the write operation is equal to the maximum tag that the invoking process witnessed.

Theorem 10: Algorithm vMWABD implements coverable atomic registers.

Proof: It is clear that vMWABD still satisfies properties **A1-A3**. Any write operation that is not successful can be mapped to a read operation that performs two phases and propagates the latest value/version to a majority of replicas before completing. It remains to show that vMWABD also satisfies validity and coverability.

Validity is satisfied since each tag is unique, as it is composed by an integer ts and the id of a process wid. The tag is monotonically incrementing at each replica, as according to the algorithm a replica updates its local copy only if a higher tag is received. A writer process w_i discovers the maximum tag $\langle ts, w_j \rangle$ among the replicas and in the second phase it generates a tag $\langle ts + 1, w_i \rangle$. As the tag at each replica is monotonically incrementing then each writer never generates the same tag twice. Also, for every write $cvr\text{-}\omega(tag)[tag', chg], tag' = \langle tag.ts{+}1, wid \rangle \Rightarrow tag' > tag$. Finally, since every tag is generated by extending the initial tag and each write operation extends a tag that obtains during its query phase then there is a sequence of tags leading from the initial tag to the tag used by the write operation.

For *consolidation* we need to show that for two write operations $\omega_1 = cvr\text{-}\omega(*)[tag_1, chg]$ and $\omega_2 = cvr\text{-}\omega(tag_2)[*, chg]$, if $\omega_1 \rightarrow_\xi \omega_2$ then $tag_1 \leq tag_2$. According to the algorithm ω_1 propagates tag_1 to the majority of replicas before completing. In the query phase, ω_2 receives

```
1: at each writer w_i
2: Components:
3: maxP ∈ ℕ⁺ × 𝒲 × V, tg ∈ ℕ⁺ × 𝒲, v ∈ V, flag ∈ {chg, unchg}
4: Initialization:
5: tg ← ⟨0, w_i⟩, v ← ⊥, maxP ← ⟨tg, v⟩
6: function CVR-WRITE(val, ver)
7:     send (Query) to all servers                                  ▷ Query Phase
8:     wait until (|𝒮|+1)/2 servers reply
9:     maxP ← max({m.⟨tg', v'⟩ | m received from some server})
10:    if ver = maxP.tg' then
11:        tg ← ⟨maxP.tg'.ts + 1, w_i⟩; v ← val; flag ← chg
12:        send (Write, ⟨tg, v⟩) to all servers                     ▷ Write Phase
13:        wait until (|𝒮|+1)/2 servers reply
14:    else
15:        tg ← maxP.tg'; v ← maxP.v'; flag ← unchg
16:        send (Propagate, maxP) to all servers                    ▷ Propagate Phase
17:        wait until (|𝒮|+1)/2 servers reply
18:    end if
19:    return(⟨tg, v⟩, flag)
20: end function

21: at each reader r_i
22: Components:
23: maxP ∈ ℕ⁺ × 𝒲 × V
24: function CVR-READ( )
25:    send (Query) to all servers                                  ▷ Query Phase
26:    wait until (|𝒮|+1)/2 servers reply
27:    maxP ← max({m.⟨tg', v'⟩ | m received from some server})
28:    send (Propagate, maxP) to all servers                        ▷ Propagate Phase
29:    wait until (|𝒮|+1)/2 servers reply
30:    return(maxP)
31: end function

32: at each server s_i
33: Components:
34: tg ∈ ℕ⁺ × 𝒲, v ∈ V
35: Initialization:
36: tg ← ⟨0, ⊥⟩, v ∈ V
37: function RCV(M)_q                                               ▷ Reception of a message from q
38:    if M.type ≠ Query and M.tg > tg then
39:        ⟨tg, v⟩ ← ⟨M.tg, M.v⟩
40:    end if
41:    send(⟨tg, v⟩) to q
42: end function
```

Fig. 2: The operations of algorithm vMWABD.

messages from the majority of replicas. So there is one replica s that received tag_1 from ω_1 before replying to ω_2. Since the tag in s is monotonically incrementing, then s replies to ω_2 with a tag $tag_s \geq tag_1$. So ω_2 receives a maximum tag $tag_{max} \geq tag_1$. Since ω_2 also changes the value and version of the register it means that its local tag tag_2 is equal to tag_{max}. This shows immediately that $tag_2 \geq tag_1$.

Continuity is preserved as a write operation first queries the replicas for the latest tag before proceeding to the propagation phase to write a new value. Since the tags are generated and propagated only by write operations then if a write changes the value of the system then it appends a tag already written, or the initial tag of the register.

To show that *evolution* is preserved, we observe that the version of a register is given by its tag, where tags are compared lexicographically (first the number $tag.ts$ and then the writer identifier to break ties). A successful write $\pi_1 = cvr\text{-}\omega(tag)[tag']$ generates a new tag tag' from tag such that $tag'.ts = tag.ts + 1$. Consider sequences of tags $tag_1, tag_2, \ldots, tag_k$ and $tag'_1, tag'_2, \ldots, tag'_\ell$ such that $tag_1 = tag'_1$. Assume that $cvr\text{-}\omega(tag_i)[tag_{i+1}]$, for $1 \leq i < k$, and $cvr\text{-}\omega(tag'_i)[tag'_{i+1}]$, for $1 \leq i < \ell$, are successful writes. If $tag_1.ts = tag'_1.ts = z$, then $tag_k.ts = z + k$ and $tag'_\ell.ts = z + \ell$, and if $k < \ell$ then $tag_k < tag'_\ell$. ■

Supporting Large Versioned Objects. Fan and Lynch [10], using algorithm MWABD as a building block, showed how large atomic R/W objects can be efficiently replicated. The main idea of their algorithm, called LDR, is to have two distinguished sets of servers: Replicas and Directories. Replica servers are the ones that actually store the object's data (value), while Directories keep track of the tags of the object and the associated Replicas that store the data of the object. A reader or writer first runs algorithm MWABD on the Directories to obtain the highest tag of the object, and the identity of the Replicas that have the associated value (aka, the most recent value of the object). A read operation, then contacts a subset of the Replicas to obtain the value of the object. A write sends the new value to a majority of the Replicas, while ensuring that Directories are updated (see [10] for details). By replacing algorithm MWABD with algorithm VMWABD and performing a few modifications to the Replicas, we can turn algorithm LDR into an algorithm that can handle *large versioned R/W objects*, such as large files. See [22] for the modified LDR.

VI. Applications of Coverable Atomic Read/Write Registers

Weak RMW registers. A shared object satisfies atomic *read-modify-write* (RMW) semantics if a process can atomically *read* and *modify* the value of the object using some function $\mathcal{F}$, and then *write* the new value on the object. Coverable atomic R/W registers can be used to implement a weak version of RMW semantics. In a weak RMW object not all operations may successfully modify the value of the object. In case that a RMW operation is not concurrent with any other operation then this operation satisfies the RMW semantics. In case where two or more operations invoke RMW concurrently, at least one of them will satisfy the RMW semantics. Finally, weak RMW allow multiple RMW operations to modify successfully the same value.

Figure 3 presents an implementation of a weak RMW object using coverable atomic R/W registers. We assume that the object offers a rmw($\mathcal{F}$) action that accepts a function and tries to apply that function on the value of the object. The object returns the initial value of the object and a flag indicating whether the value of the object was modified successfully.

```
1: At each process i ∈ 𝓘
2: State Variables:
3: lcver ∈ Versions; oldval, lcval, newv ∈ Values;
4: flag ∈ {chg, unchg}
5: function RMW(𝓕)
6:     ⟨oldval, lcver⟩ ← cvr-read()
7:     newv ← 𝓕(oldval)
8:     ⟨lcval, lcver, flag⟩ ← cvr-write(lcver, newv)
9:     if flag = chg then
10:        return ⟨lcval, success⟩
11:    else
12:        return ⟨lcval, fail⟩
13:    end if
14: end function
```

Fig. 3: Weak RMW using Coverable Atomic Registers

Theorem 11: The construction in Figure 3 implements a weak RMW object.

Proof: Consider an execution ξ of the algorithm. We begin the proof by studying the case where an operation rmw($\mathcal{F}$) is not concurrent with any other operation in ξ. The atomic nature of the register ensures that cvr-read returns the latest value and version, say $\langle ver, val\rangle$, written on the register. When the cvr-write operation is invoked, the write operation tries to modify the value associated with version ver. As there is no concurrent operation, the version of the register remains ver and thus according to *consolidation and continuity*, the write operation successfully writes the new value completing the RMW operation.

Consider now the case of two operations, π_1 and π_2, invoking rmw concurrently. Each of these operations involve a cvr-read followed by a cvr-write operation. Let ρ_{π_i} (resp. ω_{π_i}) denote the read (resp. write) operation invoked during π_i, for $i \in [1, 2]$. We have the following cases wrt the order of these operations: (i) $\omega_{\pi_1} \rightarrow \rho_{\pi_2}$, (ii) $\omega_{\pi_2} \rightarrow \rho_{\pi_1}$, (iii) $\rho_{\pi_2} \rightarrow \omega_{\pi_1} \rightarrow \omega_{\pi_2}$, (iv) $\rho_{\pi_1} \rightarrow \omega_{\pi_2} \rightarrow \omega_{\pi_1}$, or (v) ω_{π_1} is concurrent with ω_{π_2}. In case (i), both read and write operations of π_1 complete before the read and write operations of π_2 are invoked. In this case notice that the version of the object remains the same from the read to the write operation of both operations. Thus, according to *consolidation and continuity*, both write operations will successfully change the value of the register. The same holds for case (ii), where π_2's ops complete before the invocation of π_1's ops. In case (iii) the write operation of π_1 completes before the write operation of π_2. Let ρ_{π_2} in this case complete before ω_{π_1}. Both read operations ρ_{π_1} and ρ_{π_2} discover by *atomicity* the same version, say ver. So both write operations will be invoked as cvr-write(ver, v). Since no operation changes the version of the register before ω_{π_1} is invoked, then by *consolidation and continuity*, ω_{π_1} changes the version of the object to, say, ver_{π_1}. Notice that by *validity*, $ver_{\pi_1} > ver$. When ω_{π_2} is invoked it fails by *consolidation* to change the value of the object as $\omega_{\pi_1} \rightarrow \omega_{\pi_2}$ and it tries to change the version $ver < ver_{\pi_1}$ (the version of ω_{π_1}). Hence, only π_1 will manage to preserve RMW semantics. Similarly, we can show that only π_2 will preserve RMW semantics in case (iv). Finally, in case (v) if both writes try to change the version ver, both may succeed and preserve RMW semantics. Since, however, their versions are unique and comparable, then by *consolidation* any subsequent operation will RMW the highest of the two versions. So in all cases at least a single operation satisfies the RMW semantics, as desired. ■

From the proof we can extract that coverable registers may allow multiple writes to change the same version of the register, but *consolidation* ensures that at least one write satisfies RMW semantics for each version. Finally, *consolidation and continuity* ensure that eventually RMW operations diverge in a single path in the constructed tree.

Concurrent File Objects. A file object can be implemented directly using RMW semantics since one can retrieve, revise, and write back the new version of the file. As RMW semantics can be used to solve consensus [14], they are impossible to be implemented in an asynchronous system with a single crash

```
1: At each process i ∈ 𝓘
2: State Variables:
3: lcver ∈ Versions
4: lcval ∈ Values
5: flag ∈ {chg, unchg}
6: Initialization:
7: lcver ← ver_0; lcval ← ⊥
8: function GET( )
9:     ⟨lcval, lcver⟩ ← cvr-read()
10:    return ⟨lcval, lcver⟩
11: end function
12: function REVISE(v, ver)
13:    ⟨lcval, lcver, flag⟩ ←
14:          cvr-write(ver, v)
15:    if flag = chg then
16:       return OK
17:    end if
18:    return ⟨lcval, lcver⟩
19: end function
```

Fig. 4: File Object using Coverable Atomic Registers

failure. Therefore, we consider file objects that comply to the weak RMW semantics as those were given in the paragraph above. In particular, we consider *concurrent file objects* that allow two fundamental operations, *revise* and *get* to be invoked concurrently by multiple processes. The revise operation is used to change the contents of the file object, whereas the get action is analogous to a read operation and facilitates the retrieval of the contents of the file. Semantically, a file object requires that a revise operation is applied on the latest version of the file and a get operation returns the file associated with the latest written version. Depending on the implementation, the values written and returned by these operations can be the complete file object, a fragment of the file object, or just the journal containing the operations to be applied on a file (similar to a journaled file system).

Figure 4 presents the algorithm that implements the two operations. The *revise* operation specifies the version of the file to be revised along with the new value of the shared object. The cvr-write operation attempts to perform the write with the given version and returns the value and version of the register, and whether the write succeeded or not. If the write succeeded then the operation informs the application for the proper completion of the revise operation; otherwise the latest discovered value-version pair is returned. From Theorem 11 and Figure 4 we may conclude the following theorem.

Theorem 12: The construction in Figure 4 implements a file object.

VII. Conclusion

In this paper we have introduced *versioned registers* and a new property for concurrent versioned registers, we call *coverability*. A versioned register associates a version with its value, and with each operation that wants to modify its value. An operation may modify the value and the version of the register, or it may just retrieve its value-version pair. Coverability defines the exact guarantees that a versioned register provides when it is accessed concurrently by multiple processes with respect to the evolution of its versions, over a total order of its operations.

We combine coverability with atomicity to obtain coverable atomic registers. The successful writes on the register follow the total order of atomicity, while preserving the properties required by coverability. We note that a different total ordering could be used with coverability to obtain other types of "coverable objects". In fact, we believe it would be interesting to investigate further the use of coverable objects for the introduction of distributed algorithms for various applications. The fact that each operation is enhanced by the version of the object provides the flexibility to manipulate the effect of an operation under some conditions on the version of the object with respect to the version of the operation.

References

[1] What is bitcoin fork? http://blog.cex.io/bitcoin-dictionary/what-is-bitcoin-fork-14622. Accessed: 2016-05-05.

[2] Aguilera, M. K., and Horn, S. L. Abortable and query-abortable objects and their efficient implementation. In *Proc. of PODC 2007,* pp. 23–32.

[3] Attiya, H., Bar-Noy, A., and Dolev, D. Sharing memory robustly in message passing systems. *Journal of the ACM 42(1)* (1996), 124–142.

[4] Boichat, R., Dutta, P., Frølund, S., Guerraoui, R. Deconstructing paxos. *SIGACT News 34*, 1 (2003), 47–67.

[5] Chockler, G., Dobre, D., and Shraer, A. Brief announcement: Consistency and complexity tradeoffs for highly-available multi-cloud store. In *Proc. of DISC 2013*.

[6] Chockler, G., and Malkhi, D. Active disk paxos with infinitely many processes. *Distributed Computing 18*, 1 (2005), 73–84.

[7] Dobre, D., Viotti, P., and Vukolić, M. Hybris: Robust hybrid cloud storage. In *Proc. of SOCC 2014*.

[8] Dutta, P., Guerraoui, R., Levy, R. R., and Chakraborty, A. How fast can a distributed atomic read be? In *Proc. of PODC 2004,*

[9] Englert, B., Georgiou, C., Musial, P. M., Nicolaou, N., and Shvartsman, A. A. On the efficiency of atomic multi-reader, multi-writer distributed memory. In *Proc. of OPODIS 2009,* pp. 240–254.

[10] Fan, R., and Lynch, N. Efficient replication of large data objects. In *Proc. of DISC 2003,* pp. 75–91.

[11] Fischer, M. J., Lynch, N. A., and Paterson, M. S. Impossibility of distributed consensus with one faulty process. *Journal of ACM 32*, 2 (1985), 374–382.

[12] Georgiou, C., Nicolaou, N. C., and Shvartsman, A. A. On the robustness of (semi) fast quorum-based implementations of atomic shared memory. In *Proc. of DISC 2008,* pp. 289–304.

[13] Georgiou, C., Nicolaou, N. C., and Shvartsman, A. A. Fault-tolerant semifast implementations of atomic read/write registers. *Journal of Parallel and Distributed Computing 69*, 1 (2009), 62–79.

[14] Herlihy, M. Wait-free synchronization. *ACM Trans. Program. Lang. Syst. 13*, 1 (1991), 124–149.

[15] Herlihy, M. P., and Wing, J. M. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst. 12,* 3 (1990), 463–492.

[16] Lamport, L. How to make a multiprocessor computer that correctly executes multiprocess programm. *IEEE Trans. Comput. 28*, 9 (1979), 690–691.

[17] Lamport, L. On interprocess communication, part I: Basic formalism. *Distributed Computing 1*, 2 (1986), 77–85.

[18] Lynch, N. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[19] Lynch, N., and Tuttle, M. An introduction to input/output automata. *CWI-Quarterly* (1989), 219–246.

[20] Lynch, N. A., and Shvartsman, A. A. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Proc. of FTC 1997,* pp. 272–281.

[21] Mazières, D., and Shasha, D. Building secure file systems out of byzantine storage. In *Proc. of PODC 2002,* pp. 108–117.

[22] Nicolaou, N., Fernández Anta, A., and Georgiou, C. Cover-ability: Consistent versioning for concurrent objects. *CoRR abs/1601.07352* (2016).

[23] Vogels, W. Eventually consistent. *Commun. ACM 52*, 1 (Jan. 2009), 40–44.

CMTS: Consensus-based Multi-hop Time Synchronization Protocol in Wireless Sensor Networks

Amin Saïah*, Chafika Benzaïd* and Nadjib Badache*
*Univ. of Sciences and Tech. Houari Boumediene, Algiers, Algeria
Email: a.saiah@univ-chlef.dz, cbenzaid@usthb.dz, badache@cerist.dz

***Abstract*—The Consensus Time Synchronization (CTS) overcomes the shortcoming of centralized time synchronization in terms of scalability and robustness to node failure. However, CTS leads to slow convergence rate, high communication traffic and the inability to provide synchronization to an external time source. This paper proposes a novel distributed time synchronization protocol for WSNs, the Consensus-based Multi-hop Time Synchronization (CMTS) protocol. CMTS combines the benefits of consensus-based scheme, multi-level topology, synchronization by overhearing, master node synchronization, and MAC-layer timestamping. Simulations are performed to validate the effectiveness of CMTS. The results show that CMTS achieves high accuracy and improves the convergence time compared to competing schemes in the literature.**

I. Introduction

Time synchronization plays an important role in the performance of wireless sensor networks (WSNs). It can enhance the throughput and the lifetime of the network by improving the energy-efficiency, the freshness of collected data and reducing the network traffic and message conflicts.

The existing time synchronization protocols rely on one of the following pairwise synchronization methodologies: Sender-Receiver (SR), Receiver-Receiver (RR) or Receiver-Only (RO). In SR (e.g., FTSP [5], GTSP [8]), a receiver adjusts its clock according to the timestamp received from a reference node. In RR (e.g., RBS [2]), receivers within one hop use a number of synchronization pulses initiated by a sender to synchronize among themselves. While in RO (e.g., SPiRT [1], PBS [6]), a group of nodes can be synchronized by only overhearing the timing messages exchanged between a pair of nodes. The RO approach shows a promising approach to achieve time synchronization with a significantly reduced number of timing messages. It offers better performance (communication cost and energy efficiency) compared to SR and RR methodologies [6].

A variety of centralized and distributed synchronization protocols have been proposed. The centralized time synchronization protocols, such as [2], [3], [5], [6], [1], provides high synchronization accuracy and fast convergence rate due to the use of reference nodes and hierarchical structure of network topology. However, centralized time synchronization protocols rely on a single timing source which makes the synchronization protocol vulnerable to its failure. Therefore, consensus time synchronization, such as [8], [9], [7], [4], [10], are developed to overcome the shortcomings of centralized time synchronization approaches in terms of scalability and robustness to node failure. Unlike the centralized time synchronization protocols, consensus time synchronization protocols are completely distributed and do not rely on any specific node or hierarchical structure to perform time synchronization. While this confers to consensus-based schemes their scalability and robustness to node failure, it leads to slow convergence rate and high communication traffic.

This paper proposes a novel distributed time synchronization approach which overcomes the shortcomings of centralized and consensus time synchronization schemes by combining their advantages. The timing messages are exchanged following the RO methodology. This reduces the communication overhead and speeds up the convergence time. The sensor nodes compensate their clock offset and skew by synchronizing to multiple synchronized nodes which increases synchronization accuracy and robustness to node failures. In addition, nodes are organized into hierarchical structure, which allows to establish a particular communication pattern between nodes leading to reduce the number of exchanged messages and the convergence time. The main contributions of this paper are three-fold: (1) We propose CMTS, a novel consensus-based multi-hop time synchronization protocol that suits failure-prone resource-constrained WSNs. (2) We derive the Maximum Likelihood Estimators (MLE) and the corresponding Cramer-Rao lower bounds (CRLB) for joint offset/skew model under Gaussian delay model are derived. (3) We provide simulation results that show the low time synchronization error and the fast convergence rate achieved by CMTS in comparison to existing solutions.

The rest of this paper is organized as follows. Section II presents the problem statement, followed by a detailed description and analysis of CMTS in Section III. The validation results are in Section IV. Finally, Section V concludes this paper.

II. Problem Statement

The synchronization accuracy is a key factor in designing a time synchronization protocol. Indeed, the synchronization accuracy depends on various other factors, such as the network structure and setup, channel status, and the estimation scheme

978-1-5090-3216-7/16/$31.00

employed. As reported by [9], a consensus-based estimation scheme, where a node adjusts its clock by averaging the clock values of all its neighbors, is the best approximation to the ideal time. However, it should be pointed out that consensus-based schemes do not rely on any reference node or hierarchical structure. Although this confers to consensus-based schemes their scalability and robustness to node failure, it leads to slow convergence rate, high communication traffic and the inability to provide synchronization to an external time source. The need of external synchronization arises in applications that interact with users and/or measure physical quantities. In such applications, an absolute time as measured by an external reference such as UTC is essential.

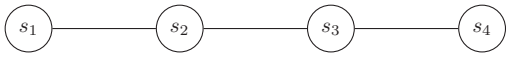


Fig. 1: A linear network topology with four sensor nodes

Since consensus-based schemes do not synchronize to an external time source, the convergence rate of all clocks to a common global time is affected by the initial clock difference among neighbor nodes. Thus, the more the clock difference among neighbors is, the higher the synchronization error and consequently the slower the convergence rate will be. To demonstrate the impact of the initial clock difference between neighbor nodes on the synchronization accuracy and convergence rate, let's consider the example in Fig. 1. In this example, four sensor nodes s_1, s_2, s_3 and s_4 aligned in a linear topology, correct their clocks using a consensus value computed by averaging the local timing information as follow: node s_1 adjusts its clock by $\frac{C(s_1)+C(s_2)}{2}$, node s_2 adjusts its clock by $\frac{C(s_1)+C(s_2)+C(s_3)}{3}$, node s_3 adjusts its clock by $\frac{C(s_2)+C(s_3)+C(s_4)}{3}$, and node s_4 adjusts its clock by $\frac{C(s_3)+C(s_4)}{2}$. $C(s_i)$ is the local clock value of node s_i.

The average synchronization error and the convergence rate have been numerically evaluated by varying the initial clock difference between nodes, δ. Three values of δ are considered, namely: $1\mu s$, $10\mu s$ and $100\mu s$. Each node adds a randomly generated and uniformly distributed offset in the range $[0, \delta]$ to its clock value. The consensus value is calculated for each node using the formula above and the average synchronization error is measured for 200 independent runs. It is observed from Fig. 2 that the average synchronization error increases as the initial clock difference between nodes increases. Therefore, using synchronized values for computing the consensus value increases synchronization accuracy.

We define the convergence rate as the number of iterations needed to achieve a steady-state synchronization in the network, that is, all clocks reach a common global time. Hence, a smaller number of iterations ensures a faster convergence. To assess the impact of the initial clock difference on the convergence rate, the calculation of the consensus value using the formula above is repeated until reaching a steady-state with an average synchronization error below $10^{-3}\mu s$. Fig. 3 presents the results, averaged over 100 independent runs. The results demonstrate a slow convergence rate as the initial clock difference increases. Indeed, the inset in Fig. 3 shows that the error becomes smaller than $10^{-3}\mu s$ after 16 iterations when the initial clock difference δ is up to $1\mu s$. Meanwhile, more than 22 and 29 iterations are required when δ is up to $10\mu s$ and $100\mu s$, respectively.

In order to fulfill the need of external synchronization and fast convergence rate, we assume the existence of an external time reference to which nodes will be synchronized. However, relying on a single timing source makes the synchronization protocol vulnerable to its failure. The proposed protocol overcomes this limitation by using a set of *master nodes* which are already synchronized. The master nodes act as reference nodes allowing other nodes to synchronize their clocks by computing a consensus value based on timestamps received from different master nodes. Accordingly, fast convergence rate with high synchronization precision can be achieved.

Moreover, the iterative process when combined to the decentralized nature of distributed consensus algorithms may cause frequent message collisions in dense WSNs. Thus, both communication overhead and convergence time will increase considerably, resulting in higher energy consumption and lower synchronization precision. To address this issue, we incorporate a multi-level hierarchy into distributed consensus algorithms. The level-L nodes synchronize their clocks to nodes' clocks of the two higher levels $(L-1)$ and $(L-2)$ which are already synchronized. By synchronizing sensor nodes to multiple source nodes, we can increase the robustness of our scheme to node failures. Meanwhile, using a level-based scheme allows to establish a particular communication pattern between nodes leading to reduce the convergence time.

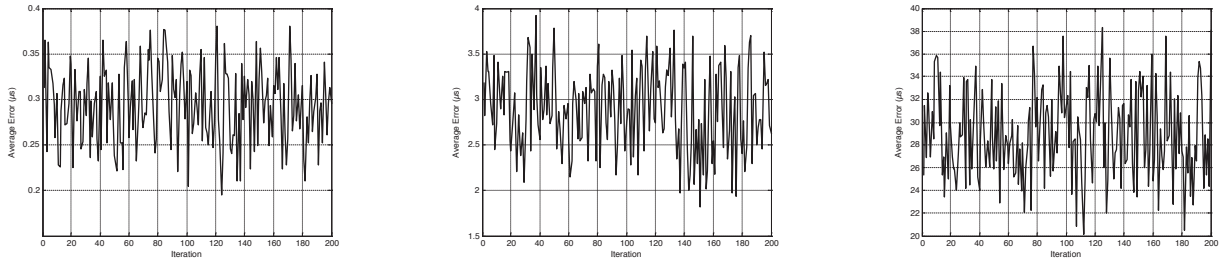
III. CMTS: Consensus-based Multi-hop Time Synchronization

A. System Model

We consider a network where static sensor nodes are organized into levels: 0, 1, $\cdots$, L, $\cdots$, etc. Every node in the network has a unique ID and is assigned a level that reflects the number of hops on the shortest path from a node to the base station (i.e., the sink). Nodes are neighbor-aware. A level-L node without neighbors at level-$(L+1)$ is a *leaf* node. Fig. 4 illustrates an example of a sensor network organized into four levels, where nodes 2, 5 ,8, 9, 10 and 11 are leaves.

The sink at level 0 and nodes at level 1 represent the set of master nodes. We assume that the sink's clock is synchronized with a conventional external clock source (e.g., UTC) and serves as an accurate time server for master nodes at level 1. A level-L node $(L \geq 2)$ synchronizes its clock by computing a consensus value based on timestamps received from the two higher levels $(L-1)$ and $(L-2)$. In this work, we focus on achieving network-wide external time synchronization for applications requiring an absolute, human timescale such as security applications, target tracking, etc. Hence, the presence of an *external clock source* is essential. Nevertheless, CMTS can operate even in case the external time source is not required, or not accessible.

The timing messages are exchanged following the RO methodology, due to the significantly reduced communication



(a) Offset=1μs (b) Offset=10μs (c) Offset=100μs

Fig. 2: Average synchronization error vs. the initial clock difference between nodes in the target linear topology

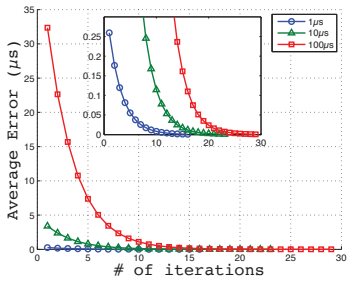


Fig. 3: Convergence rate

cost entailed by this methodology. In addition, timestamping is performed at MAC layer to decrease synchronization error.

B. Main Idea of CMTS

The key idea behind CMTS consists in achieving network-wide external synchronization by combining the aforementioned benefits of consensus-based schemes, multi-level topology, and RO synchronization methodology. A level-L node ($L \geq 2$) synchronizes its clock by computing a consensus value based on timestamps received from the two higher levels $(L-1)$ and $(L-2)$. For instance, in Fig. 4, nodes 6, 7 and 8 can synchronize their clocks using timestamps received from nodes 1, 2, 3, 4 and 5.

The synchronization of a level-L node's clock is as follows: Every non-leaf node of level $L-1$ broadcasts a beacon. On receiving the beacon message, its one-hop neighbors that are in the same level $(L-1)$ or one level upper $(L-2)$) send back the beacon's arrival time. After collecting all reply messages, the beacon sender broadcasts the list of collected timestamps to level-L nodes. Thus, a level-L node will receive a list of timestamps from each of its neighbors at level $L-1$. Using the received timestamps, a level-L node synchronizes its clock by computing a consensus value. This process is repeated as subsequent levels until synchronizing all nodes. It is important to note that consensus value is calculated based on timestamps issued from synchronized clocks. By doing so, fast convergence with high synchronization precision over hop counts can be achieved. It is also noteworthy that level-L nodes can get synchronized without exchanging any packets with other nodes. In the example of Fig. 4, to synchronize nodes of level 2, the non-leaf nodes at level 1 (i.e., nodes 3 and 4) broadcast a beacon message. Upon receiving the beacon message, their neighbors at level 1 and 0 reply by sending the beacon's arrival time. Hence, nodes 1, 2 and 4 reply to node 3, meanwhile, nodes 1, 3 and 5 reply to node 4. Afterwards, nodes 3 and 4 broadcast the collected timestamps to nodes of level 2 and the latter can be synchronized by computing a consensus value from received timestamps.

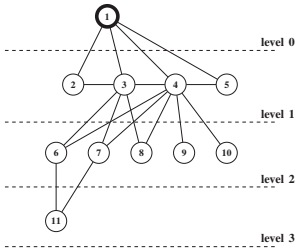


Fig. 4: Example of a level-based topology.

C. Detailed CMTS Description

Assume that each node s_j at level $(L-1)$ maintains three one-hop neighbor lists, namely: (1) $R^{l}_{(j)}$ which contains the IDs of the one-hop s_j's neighbors at the level below (L). A node in this list is called a *child* neighbor. (2) $R^{l-1}_{(j)}$ which contains the IDs of the one-hop s_j's neighbors at the same level $(L-1)$. A node in this list is called a *sibling* neighbor, and (3) $R^{l-2}_{(j)}$ which contains the IDs of the one-hop s_j's neighbors at the level above $(L-2)$. A node in this list is called a *parent* neighbor. In the given example, the one-hop neighbor lists $R^{2}_{(4)}$, $R^{1}_{(4)}$ and $R^{0}_{(4)}$ of node 4 are respectively $<6,7,8,9,10>$, $<3,5>$, and $<1>$.

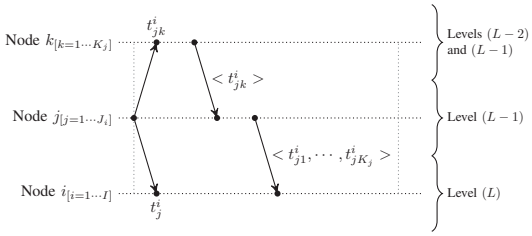


Fig. 5: Clock synchronization principle of CMTS protocol.

The clock synchronization model of CMTS is depicted in Fig 5. Let $i \in \{1,\ldots,I\}$ denotes a level-L node. Let $j \in R^{l-1}_{(i)} = \{1,\ldots,J_i\}$ denotes a node i's parent neighbor, where $J_i = \mid R^{l-1}_{(i)} \mid$. Let $k \in R^{l-1}_{(j)} \cup R^{l-2}_{(j)} = \{1,\ldots,K_j\}$

denotes either a sibling neighbor or a parent neighbor of node j, where $K_j = \mid R_{(j)}^{l-1} \cup R_{(j)}^{l-2} \mid$. A level-$L$ node i ($L \geq 2$) is synchronized as follows: Each node j broadcasts a beacon. Nodes i and k receive the beacon message at their local time t_j^i and t_{jk}^i, respectively. Node k sends back the beacon's arrival time t_{jk}^i to j. After collecting timestamps from each node k, node j broadcasts the timestamps list $\{t_{j1}^i, ..., t_{jK_j}^i\}$ to node i. So, node i will receive a list of timestamps from each node j. Using the received timestamps, node i can compensate its clock offset and skew.

Note that node k may receive several beacons originating from different nodes j. Thus, node k has to send back the arrival time for each of these beacons, which may results in severe congestion and collision. In order to reduce this communication overhead, node k concatenates the recorded arrival times into a single packet and broadcasts the combined packet. The packet piggybacks the following list $\{< ID_1, t_1 > , \ldots , < ID_z, t_z >, \ldots , < ID_Z, t_Z >\}$, where Z is the number of received beacons and t_z is the arrival time of beacon sent by node with identity ID_z. Upon receipt of this packet, node j reads the timestamp corresponding to its identity.

D. Clock Estimators and Analysis

A first order linear model is used to represent the relative clock between nodes, involving the effect of both clock offset and skew. Thus, the relative clock model between nodes i and k is given by:

$$t_{jk}^i = a_i . t_j^i + b_i + D_{jk} \tag{1}$$

where a_i and b_i stand, respectively, for the relative skew and offset between nodes i and k. Packet delay d can be divided into two parts: the *fixed* portion of delay in up- and down-link (d^{fixed}) and the *variable* portion of delay in up- and down-link (d^{var}). Let d_j and d_{jk} be a delay between nodes j and i, and nodes j and k, respectively. Therefore, $d_j = d_j^{fixed} + d_j^{var}$ and $d_{jk} = d_{jk}^{fixed} + d_{jk}^{var}$. The fixed portions are assumed to be equal and the variable portions to be Gaussian distributed random variables (rv) with the same parameters, i.e., $d_j^{var}, d_{jk}^{var} \sim \mathcal{N}(\mu, \sigma_0^2)$. It follows that $d_{jk} - d_j = d_{jk}^{var} - d_j^{var}$. Let us denote $d_{jk}^{var} - d_j^{var}$ by D_{jk}. Since i.i.d random delay D_{jk} follows Gaussian distribution, the likelihood function based on the observations $\{t_j^i\}_{1 \leq j \leq J_i}$ and $\{t_{jk}^i\}_{1 \leq j \leq J_i}^{1 \leq k \leq K_j}$, is given by:

$$f(b_i, a_i, \sigma^2 \mid DS) = \prod_{j=1}^{J_i} \left(\prod_{k=1}^{K_j} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(D_{jk})^2} \right) = \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^S e^{\frac{-1}{2\sigma^2} \sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} (D_{jk})^2 \right)} \tag{2}$$

Differentiating the log-likelihood function with respect to a_i and b_i and setting the result to zero produces

$$\hat{a}_i = \frac{S \sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_{jk}^i . t_j^i \right) - \sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_{jk}^i \right) \times \sum_{j=1}^{J_i} t_j^i . K_j}{S \sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} (t_j^i)^2 \right) - \left(\sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_j^i \right) \right)^2} \tag{3}$$

$$\hat{b}_i = \frac{1}{S} \left[\sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_{jk}^i \right) - \hat{a}_i \times \sum_{j=1}^{J_i} t_j^i . K_j \right] \tag{4}$$

The *CRLB* can be obtained by taking the inverse of the $[i,i]th$ element of the Fisher information matrix (i.e., $var(\hat{\theta}_i) \geq [I^{-1}(\theta)]_{ii}$), where the inverse of the Fisher information matrix, $I^{-1}(\theta)$ is given by:

$$I^{-1}(\theta) = \frac{\sigma^2}{S \sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} (t_j^i)^2 \right) - \left(\sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_j^i \right) \right)^2} \times \begin{bmatrix} \frac{\sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} (t_j^i)^2 \right)}{\sigma^2} & -\frac{\sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_j^i \right)}{\sigma^2} \\ -\frac{\sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_j^i \right)}{\sigma^2} & \frac{S}{\sigma^2} \end{bmatrix} \tag{5}$$

Consequently, the CRLBs of clock offset and skew are respectively given by:

$$Var(\hat{b}_i) \geq (I^{-1}(\theta))_{1,1} = \frac{\sigma^2 \sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} (t_j^i)^2 \right)}{S \sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} (t_j^i)^2 \right) - \left(\sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_j^i \right) \right)^2} \tag{6}$$

$$Var(\hat{a}_i) \geq (I^{-1}(\theta))_{2,2} = \frac{S\sigma^2}{S \sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} (t_j^i)^2 \right) - \left(\sum_{j=1}^{J_i} \left(\sum_{k=1}^{K_j} t_j^i \right) \right)^2} \tag{7}$$

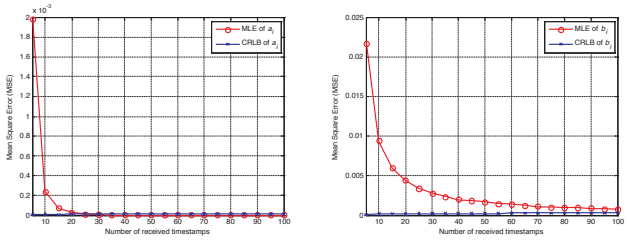


Fig. 6: MSE performance of clock skew a_i and clock offset b_i estimation vs. number of observations

The mean square error (MSE) performance of the proposed estimators has been numerically evaluated and compared with the corresponding CRLB. The obtained results, depicted in Figure 6, show that the the performance of the proposed estimators improves and quickly converges to the CRLB as the number of observations S increases.

IV. Performance Evaluation

CMTS is validated through simulation using Avrora simulator. The performances of CMTS is compared to ATS [7] and FTSP [5]. The simulation study is conducted using 16 nodes organized into a multi-level 4×4 grid-based topology as depicted in Figure 7. The sink node 1 initiates a time synchronization round once every 5 seconds. Nodes 2 and 3 are designated as master nodes and their clocks are synchronized to the sink node RO methodology. The protocols are evaluated in terms of synchronization error and convergence rate. The time synchronization is defined as the time difference between the real clock reading of one sensor node and the estimated clock value by another node. The convergence rate refers to the number of synchronization rounds needed to achieve a steady-state synchronization in the network; that is, all clocks reach a common global time. Note that the simulation results reflect the average of 15 independent runs.

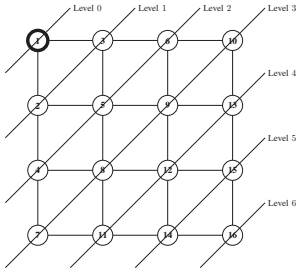


Fig. 7: A 7 levels 4×4 grid-based topology used to evaluate the performance of CMTS, ATS, and FTSP.

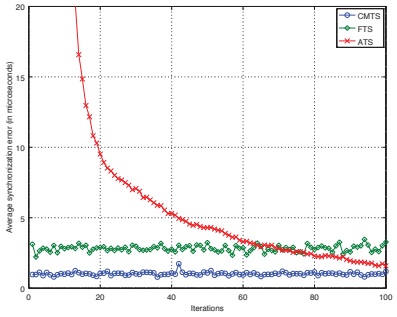


Fig. 8: Comparison of CMTS, ATS and FTSP on accuracy

After the completion of each synchronization round, we use an additional external node which broadcasts a message to all nodes. The difference between the reception time of that message in different nodes gives us the synchronization error. Figure 8 plots the simulation results of the average synchronization error of CMTS, ATS and FTSP protocols. The obtained results demonstrate that CMTS achieves an average synchronization error of around $0.9\mu s$. From Figure 8, it is clear that CMTS outperforms both ATS and FTSP thanks to the combined benefits of consensus-based scheme, multi-level topology, master node synchronization, and MAC-layer timestamping.

The convergence rate is reported for CMTS, ATS and FTSP in Figure 9. The results demonstrate the superiority of CMTS over ATS with regard to the number of iterations required to reach convergence. CMTS needs only one synchronization round to converge towards an average synchronization error of less than $2\mu s$ by exchanging 42 timing messages. Meanwhile, ATS could reach such a precision after 90 rounds and by exchanging 16 timing messages per round; thus, a total of 1440 timing messages. The obtained results show that both CMTS and FTSP converge to a common global time since the first synchronization round, but CMTS still offer a better synchronization accuracy.

V. Conclusion

In this paper, we presented a new distributed time synchronization approach for WSN, the Consensus-based Multi-hop Time Synchronization (CMTS) protocol, which conducts both the skew and offset compensations. Using a first-order linear clock model, the Maximum Likelihood Estimators (MLEs)

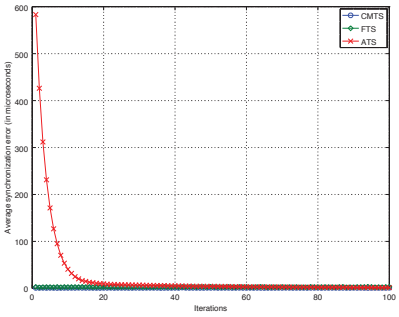


Fig. 9: Comparison of CMTS, ATS and FTSP on convergence time

of joint offset/skew model have been derived, as well as the corresponding Cramer-Rao Lower Bounds (CRLBs). CMTS is validated through simulation on a Micaz platform using Avrora simulator. The validation results have demonstrated that CMTS achieves higher accuracy and faster convergence compared to competing schemes, thanks to the combined benefits of consensus-based scheme, multi-level topology, master node synchronization, and MAC-layer timestamping. Performance evaluation of CMTS through real-world experimentation is a perspective to this work.

References

[1] C. Benzaid, M. Bagaa, and M. Younis. An efficient clock synchronization protocol for wireless sensor networks. *Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 718 – 723, Aug. 2014.

[2] J. Elson and D. Estrin Birman. Fine-grained network time synchronization using reference broadcast. *In Proc. of the 5th Symp. on Oper. Syst. Design and Implementation (OSDI'02)*, pages 147–163, Dec. 2002.

[3] S. Ganeriwawal, R. Kumar, and S. M. Timing-sync protocol for sensor networks. *In Proc. of the 1st ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, pages 138–149, Nov. 2003.

[4] J. He, P. Cheng, L. Shi, and J. Chen. Time synchronization in wsns: A maximum value based consensus approach. *In Proc. of CDC-ECC*, pages 7882–7887, 2011.

[5] M. Maróti, B. Kusy, G. Simon, and A. Lédezi. The flooding synchronization protocol. *In Proc. of the 2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys'04)*, pages 39–49, Nov. 2004.

[6] K. L. Noh, E. Serpedin, and K. Qaraqe. A new approach for time synchronization in wireless sensor networks: Pairwise broadcast synchronization. *IEEE Trans. Wireless Comm.*, 9:3318–3322, 2008.

[7] L. Schenato and F. Fiorentin. Average timesynch: a consensus-based protocol for time synchronization in wireless sensor networks. *Automatica*, 47(9):1878–1886, 2011.

[8] P. Sommer and R. Wattenhofer. Gradient clock synchronization in wireless sensor networks. *In Proc. of the 2009 Int. Conf. on Info. Processing in Sensor Net. (IPSN'09)*, pages 37–48, 2009.

[9] J. Wu, L. Jiao, and R. Ding. Average time synchronization in wireless sensor networks by pairwise messages. *Journal of Computer Communications*, pages 221–233, 2012.

[10] J. Wu, L. Zhang, Y. Bai, and Y. Sun. Cluster-based consensus time synchronization for wireless sensor networks. *IEEE Sensors Journal*, 15(3):1404–1413, Mar. 2015.

V-Hadoop: Virtualized Hadoop Using Containers

Srihari Radhakrishnan
University of Waterloo
E-mail: s2radhak@uwaterloo.ca

Bryan J. Muscedere
University of Waterloo
E-mail: bmuscede@uwaterloo.ca

Khuzaima Daudjee
University of Waterloo
E-mail: kdaudjee@uwaterloo.ca

***Abstract*— MapReduce is a popular programming model used to process large amounts of data by exploiting parallelism. Open-source implementations of MapReduce such as Hadoop are generally best suited for large, homogeneous clusters of commodity machines. However, many businesses cannot afford to invest in such infrastructure and others are reluctant to use cloud services due to data security and privacy concerns. In this paper, we present V-Hadoop, a framework that leverages Linux containers to allow users to run Hadoop jobs efficiently without requiring large, expensive, physical machine clusters. We describe our design and implementation of V-Hadoop and show that it can effectively support cluster-level parallelism. We experimentally demonstrate that V-Hadoop is a viable solution that performs competitively compared to solutions designed for large clusters.**

I. Introduction

The MapReduce [1] framework is a specialized programming model for processing large amounts of data at scale across large clusters of commodity machines. Apache Hadoop [2] is a popular open-source implementation of MapReduce and currently supports an ecosystem of tools [3] used widely in industry. Since Hadoop's shift to open-source in 2011, the MapReduce framework remains an integral part of data processing.

Companies such as Google, Facebook, and Amazon run MapReduce on datacenter-scale clusters to process hundreds of petabytes of data a day [4]. As other businesses look to deploy Hadoop to meet their data processing needs [2], many of them do not have the need or the capital to purchase and maintain large clusters of physical machines. Since MapReduce is optimized for highly parallelized, distributed tasks spread across a homogeneous cluster, small or heterogeneous clusters restrict the benefits of parallelism and result in batch jobs running for undesirable lengths of time. This is because running MapReduce jobs on clusters with a smaller number of nodes restricts the amount of parallelism [2] due to the default 1:1 mapping between Hadoop nodes and physical machines. While the advent of public cloud services has introduced some degree of flexibility in terms of renting large clusters as opposed to buying them, many businesses are reluctant to migrate their data to low-cost commercial cloud-computing datacenters due to increasing security threats to personal and confidential data [5].

In view of these challenges, there has been an emphasis on developing better solutions to run distributed jobs while being hardware agnostic. Recent related work [6], [7] explores the use of virtualization to provide a unified view of a cluster of physical machines as a single resource. Experiments that examine the benefits of virtualizing Hadoop nodes using Virtual Machines (VMs) [8], [9] show that virtualizing Hadoop is promising due to improved scheduling and resource utilization. Though VMs pave the way for virtualizing Hadoop, performance issues with virtualized stacks exist because VMs introduce an unnecessary resource overhead from having to run their own kernel on top of the host operating system.

To address the lack of performant solutions in this space, we present the **V-Hadoop** system which leverages container-based virtualization to overcome the performance issues associated with deploying Hadoop using VMs. By leveraging container-based virtualization, our V-Hadoop framework is capable of starting and managing multiple Hadoop nodes across multiple physical machines where the number of Hadoop nodes is typically greater than the number of physical machines. This decouples the degree of parallelism from the number of physical nodes, effectively increasing parallelism in the system.

V-Hadoop enables a Hadoop cluster to scale to additional nodes when computer resources are overutilized and scale down to fewer nodes during machine underutilization. We demonstrate that V-Hadoop has distinct performance advantages over traditional Hadoop clusters for specific categories of jobs and is generally performance competitive with traditional Hadoop clusters.

II. Background

Also known as operating system-level virtualization, container-based virtualization is an emergent technology that allows users to run multiple user-space instances across a single host. As the popularity of this virtualization method has grown rapidly over the past several years, solutions such as Linux Containers (LXC), Docker and OpenVZ have emerged. LXC is popular due to its ease of use, useful API, and small package size [10].

In Linux-based hosts, container-based virtualization software leverages two separate features of the Linux kernel: control groups (cgroups) and namespaces [11]. Namespaces is a Linux kernel feature that places computer resources into an abstraction so that processes in each namespace believe they have exclusive use of that resource. Cgroups is a Linux kernel feature that segregates processes into groups, allowing the OS to allocate and limit the processes given to each group. Both these features form the basis for Linux-based container virtualization [11].

978-1-5090-3216-7/16/$31.00

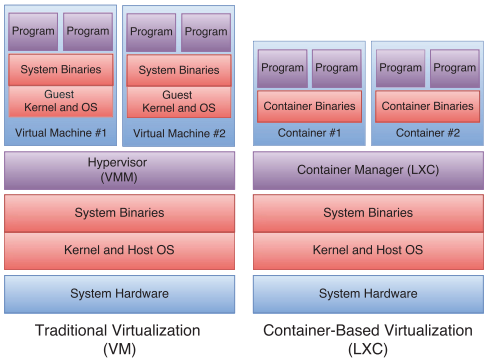


Fig. 1: *The typical program stack for hypervisor-based virtualization (left) compared to container-based virtualization (right).*

Although containers appear to be similar in function to VMs that use hypervisor-based virtualization, there are many key differences. Figure 1 shows the high-level difference between hypervisor-based virtualization and container-based virtualization. First, for hypervisor virtualization, since each VM is guaranteed a fixed amount of physical resources set by the user, resources allocated to a VM can be underutilized. Comparatively, containers avoid this problem by sharing physical resources with their host machine so that machine resources are used only when required. Additionally, since VMs emulate full machines, they introduce the overhead of an entire operating system. This adds unnecessary bulk to application specific computing clusters such as ones used for MapReduce jobs. A recent study [12] comparing the boot times of Linux containers and VMs showed that containers boot six times faster than VMs. The differences between containers and VMs make containers a more attractive option for efficiently running processes in isolation without having to manually set resource limits.

III. V-Hadoop System

The V-Hadoop framework carries out three tasks: *a*) cluster elasticity; *b*) container setup; and *c*) container management. Each of these operations is important in meeting the high-level goals of elasticity, hardware agnosticism, and management of a Hadoop cluster. The remainder of this section describes V-Hadoop in the context of these three tasks.

A. Cluster Elasticity

Similar to how Hadoop Distributed File System (HDFS) and Yarn [13] manage data and MapReduce tasks across Hadoop nodes, V-Hadoop dynamically manages the containerized Hadoop nodes to optimize physical resources and provide real time elasticity of the virtual cluster without manual intervention. V-Hadoop is organized into a master-worker architecture where one physical machine is denoted as the *master* and the rest of the machines in the cluster are denoted as *workers*. To easily maintain a running V-Hadoop cluster in the presence of machine failures, the master must be the same machine that has the containers running the master Hadoop node. The NameNode and ResourceManager of the Hadoop cluster must also reside on that master machine. This ensures that container crashes of the master node will be quickly resolved. Traditionally, the master node is protected by a failover mechanism such as Zookeeper [13].

In our framework, the purpose of the master/worker dichotomy is important. The master machine manages the placement of containers on the system and makes decisions regarding scaling the cluster up or down whereas the workers carry out decisions made by the master node and are responsible for starting and stopping containers and running Hadoop configuration tasks. Further, the workers continually check the state of their respective systems for container crashes or resource utilization issues. If a worker detects a problem, it contacts the master for a decision. The master is responsible for this decision since it has a global view of the cluster; it can take the states of all workers into account before deciding where containers need to be spawned and/or shut down.

B. Container Setup

V-Hadoop initially starts all required containers, makes copies of containers where necessary, and configures Hadoop on each container. This phase is invoked by the master machine and propagates to other machines over a period of time. When the master is started, it detects all the physical machines in the cluster and configures the worker processes on each machine.

Since this phase can experience long periods without any communication between master and worker, we use heartbeat messages to check the status of the physical machines in the cluster. Once deployed, the master continually sends a HeartbeatRequest message to each worker. The workers respond with a HeartbeatResponse message. In the event that the master does not receive a reply from a worker within a set interval, it assumes the machine is dead. Since heartbeat messages have to be sent out continuously, the messages are small. HeartbeatRequest messages are sent at a fixed interval, TPROBE, which can be set programmatically. The payload and time-to-live (TTL) of the Heartbeat messages are small enough that even frequent messages between the master and worker do not generate significant network traffic.

Once the master has determined which workers are running, it initiates the setup of the V-Hadoop cluster. Based on the user's initial requirements for the number of nodes in the Hadoop cluster, the master starts and configures virtual Hadoop nodes on the local machine and instructs the workers to do the same on their machines. Since V-Hadoop is specifically concerned with managing a physical cluster consisting of a small number of high performance machines, we fill up the machines with containers greedily, starting with the master machine and proceeding to do the same on worker machines in order of lowest round-trip time first. The logic for this placement is handled by the container management aspect of our framework.

When new nodes need to be started on a worker, the master issues a network request to that worker with information

on the number of nodes required. The desired worker then creates the required number of containers and responds to the master with a SUCCESS message and payload consisting of a list of the currently running nodes with their network addresses. Many techniques such as asynchronous network requests, non-blocking network I/O, and callback mechanisms can be applied to reduce delays at the master as it waits for responses from workers while creating multiple nodes. These details are implementation specific, as is the creation and replication of containers and depends on the type of container technology being used (Section III-D).

Once the master has received a list of running Hadoop nodes from all the workers, it notifies the user that Hadoop can now be used. Since HDFS and Yarn are unaware of the underlying node-container abstraction, the code managing the NameNode, DataNodes, and NodeManager remains unchanged from Hadoop's fully-distributed mode. This natural abstraction offered by virtualization can be used to streamline several features in Hadoop such as the distributed file system.

C. Container Management

Our framework defines a container management mode that occurs once a MapReduce job is running on a V-Hadoop cluster. The primary purpose of this mode is to periodically scan the physical machines in the V-Hadoop cluster and check for underutilization, overutilization and failed containers.

The worker on each machine periodically checks machine health by aggregating machine statistics and comparing it against a pre-determined ceiling for CPU, memory and disk usage. Each worker also checks the health of the containers on the system with the same periodicity. If a worker detects that a machine is underutilized or overutilized or that a container has crashed, it sends a DecisionRequest message to the master indicating that a container management decision needs to be made. DecisionRequest messages queue up in temporal order in a queue on the master. The master picks up DecisionRequests from the DecisionQueue and processes them one by one. Each DecisionRequest causes the master to send messages to all the workers so that it can obtain a global view of the utilization of all machines and the health of all containers. With this global snapshot of the physical cluster, the master balances the system load by shutting down containers in overutilized machines and spawning new ones in underutilized ones. DecisionRequests triggered by crashed containers are also dealt with in a similar fashion.

The V-Hadoop framework ensures fair resource utilization across the physical machines in the cluster and automatically scales up or scales down the size of the V-Hadoop cluster based on the state of currently running jobs.

D. Implementation

We implemented a prototype called the V-Hadoop Container Manager (VCM) which implements the V-Hadoop framework and integrates a container management system. Overall, the VCM's tasks are: *a*) start and stop containers on a system; *b*) configure Hadoop on all containers running across all physical machines to form a cluster; *c*) pull resource information from each physical machine in the V-Hadoop cluster; and *d*) perform simple container management including adding and removing containers on-demand.

Due to the flexibility of the V-Hadoop framework, the VCM can be run on a single machine or across a small cluster of physical machines. To allow for this, VCM is comprised of two processes: *MainVCM* and *VCMLite*. These processes form a master-worker architecture where the MainVCM process coordinates the cluster.

Based on this, if running the VCM on a single machine, it will run in *local mode* with the MainVCM process running by itself. In *distributed mode*, where multiple machines are running the VCM, one machine will run the MainVCM process whereas the rest run VCMLite. In this mode, the physical machine running MainVCM is denoted as the master and the other machines are the workers.

IV. Evaluation

In this section, we compare the performance of MapReduce jobs on V-Hadoop with the two standard Hadoop modes, fully-distributed, and standalone (pseudo-distributed).

A. Test Environment

Our evaluation environment consisted of a 10-machine cluster. Each machine contained twelve Intel(R) Xeon(R) E5-2630 v2 @ 2.60GHz CPUs with 6 cores, 256GB of RAM, and 2TB of disk storage split across 3 physical disks. Our tests used three widely recognized Hadoop benchmarks: TestDFSIO, MRBench, and TeraSort. They were specifically chosen to compare the system performance of V-Hadoop against the two Hadoop modes with respect to different resources: TestDFSIO for disk intensive jobs, MRBench for CPU intensive jobs, and TeraSort for CPU, disk and network intensive jobs. The machines in the fully-distributed setup were connected using a single TopOfRack(ToR) switch on an Intel I350 Gigabit network connection.

For standalone mode, all tests were run on a single machine. V-Hadoop tests were conducted on a single machine running a virtual cluster of 10 containers. The containers used with V-Hadoop were configured with Ubuntu *14.04* and Hadoop *2.7.1*. We configured HDFS to map to the container's file system and set the replication factor to 1 since replicating blocks on the same physical machine is redundant. Further, replication on the same disk could slow down jobs due to disk contention.

B. TestDFSIO Benchmark

TestDFSIO is a benchmark that stresses the Hadoop Distributed File System (HDFS) and is split into read and write tests. The TestDFSIO benchmark uses one map task per file. We ran TestDFSIO by varying the number of files and file sizes and by measuring the benchmark execution time of writes and reads. Fig 2 shows a plot of execution time versus the file size for a ten-node cluster. In comparing the TestDFSIO

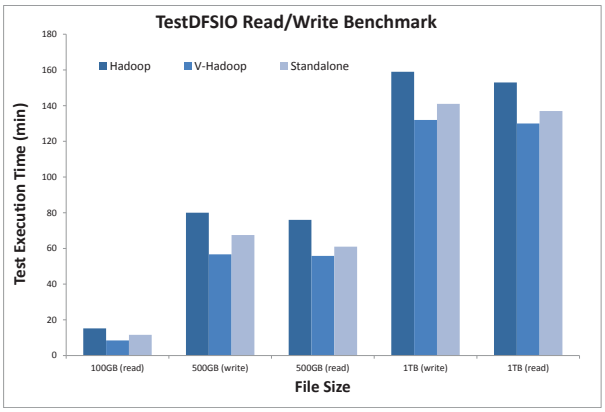


Fig. 2: *TestDFSIO read/write benchmarks execution time for different Hadoop modes (10GB file size).*

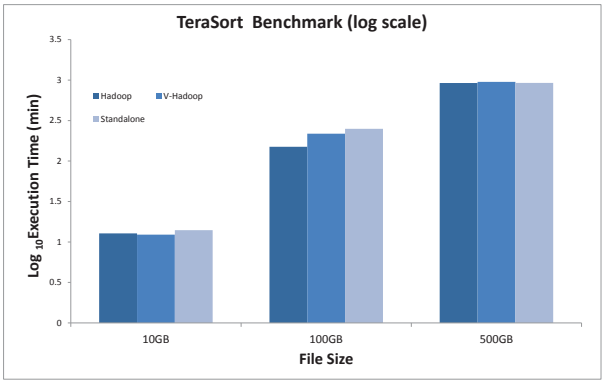


Fig. 3: *TeraSort benchmark results for the three Hadoop modes (log scale execution time).*

benchmark throughput for the three modes, we find that V-Hadoop and standalone Hadoop perform better for writes and reads compared to the fully-distributed mode, primarily because V-Hadoop and standalone modes obviate the need for network I/O.

We observe that network I/O in the fully-distributed mode is significant enough to cause a marked difference in execution times, despite the fully-distributed mode is writing and reading files from 10 machines, thereby having much lower disk and memory contention than V-Hadoop and standalone modes. Fig 2 shows that V-Hadoop performs better than Hadoop for I/O-intensive operations and that this performance difference is maintained as the size of the dataset is scaled up.

C. TeraSort Benchmark

The TeraSort benchmark is used to test the HDFS and Yarn layers of the Hadoop cluster. It is a popular benchmark often used by industry to measure the standard of a Hadoop cluster. The TeraSort benchmark is a good measure of how well the Hadoop cluster is configured in terms of the number of Map/Reduce tasks, and how well they are balanced compared to the number of disks, cores and machines in the cluster. The goal of TeraSort is to sort a large number of 100-byte records as quickly as possible. A full TeraSort benchmark run consists of the following three steps: *a*) generating the input data using TeraGen; *b*) running TeraSort on the input data; and *c*) validating the sorted output data via TeraValidate. Through these three steps, the benchmark performs considerable amount of computation and I/O and is considered to be representative of real MapReduce programs.

TeraSort overrides the specified replication factor so only one copy is written to HDFS. TeraValidate reads the sorted data and verifies whether it is in order using one map task per file and combines the results using a single reduce task to check if the files are contiguous.

We ran the three stages of TeraSort on data sets ranging from 1GB to 0.5TB on the three Hadoop modes, for increasing block sizes. Fig 3 shows a comparison of the TeraSort benchmark runtimes averaged over all three phases (TeraGen, TeraSort and TeraValidate) for the three modes. The benchmark was run on datasets of sizes 0.01TB, 0.1TB, and 0.5TB with block size 512MB on a logarithmic scale of execution time (minutes). We observed that for smaller file sizes, the difference in execution speeds was negligible. For larger files, V-Hadoop performed comparably to the fully-distributed mode even though it faced heavier disk contention. The lower network I/O and better CPU utilization allows V-Hadoop to perform a CPU and I/O intensive task such as TeraSort comparably to an equivalent cluster of physical nodes.

D. MRBench Benchmark

MRBench is a benchmark that iterates a short MapReduce program a number of times. MRBench checks whether small job runs are responsive and running efficiently on the cluster. MRBench is complementary to the TeraSort benchmark. We chose MRBench as a way to compare the MapReduce layer of V-Hadoop to that of a fully-distributed Hadoop cluster. The MRBench tests were performed on a minute long MapReduce job which was looped over 50, 500, 1000 and 5000 times.

Fig 4 shows a comparison of V-Hadoop, standalone and fully-distributed Hadoop for each of these runs. We observe that the execution times between V-Hadoop and Hadoop remain comparable as the number of runs is scaled up from 50 to 5000 even though V-Hadoop runs on a single machine, as opposed to the fully-distributed cluster running on 10 machines. Comparing execution times of V-Hadoop and standalone mode, we see that V-Hadoop consistently performs better, and the gap between the execution times continues to widen as the number of runs are scaled up. This experiment illustrates why container based virtualization is a clear winner for performance reasons alone – we can achieve results comparable to a fully-distributed cluster on a physical cluster which is an order of magnitude smaller in size. V-Hadoop uses more virtual nodes to maximize the memory and CPU utilization of each physical machine, as opposed to the 1:1 mapping between nodes and machines in a fully-distributed Hadoop cluster.

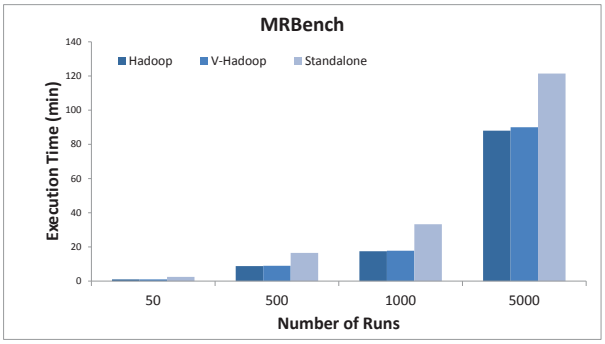


Fig. 4: *MRBench benchmark results for the three Hadoop modes.*

E. Discussion

V-Hadoop provides the best of both worlds by combining the resource utilization of the standalone mode with the scalability of the fully-distributed mode. It utilizes machine resources more efficiently by design while still providing the flexibility of a fully-distributed Hadoop cluster as new nodes can span physical machines by leveraging containers. Our experiments show that even though fully-distributed Hadoop performs slightly better on CPU intensive benchmarks than V-Hadoop, the performance gain is not significant enough to justify the use of more than one powerful physical machine. Additionally, while V-Hadoop and standalone Hadoop run on identical hardware and have comparable performance in the TeraSort benchmark, V-Hadoop is the better choice mainly because of the distinct advantage it offers in highly parallel jobs (such as MRBench) and I/O heavy jobs (as shown in the TestDFSIO benchmark experiments). Furthermore, V-Hadoop provides a layer of abstraction atop a physical cluster that allows users to easily scale-up or scale out as well as expand their Hadoop cluster while being agnostic to the underlying hardware.

V. Related Work

Previous work [9], [14] has examined the feasibility of running a virtual Hadoop cluster using a collection of virtual machines on a physical machine. Research from VMWare has looked at the performance of running a virtual Hadoop cluster using proprietary hypervisor virtualization technology [8]. While VMWare's study demonstrates that running multiple Hadoop instances on a single machine can improve throughput of MapReduce jobs, virtualizing using VMs can have significant overhead. Other work [15] has evaluated the performance of virtualized Hadoop and found that the performance of I/O-intensive jobs is more sensitive to the virtualization overhead than that of CPU-intensive jobs due to shared I/O.

Apache Mesos is a cluster management tool that provides users with the ability to share clusters between distributed computing frameworks such as Hadoop [7]. While Mesos is different from V-Hadoop in specific functionality, Mesos takes advantage of containerization to provide resource isolation between distributed computing frameworks and applications [7]. Mesos does not run multiple containers of the same framework on a single worker machine.

VI. Conclusion

In this paper, we presented V-Hadoop, a framework that allows users to run a virtual Hadoop cluster across any number of machines while being agnostic to the underlying hardware. The prototype version of our V-Hadoop framework can manage a V-Hadoop cluster across multiple physical machines, and can perform simple, resource-based scheduling of Linux containers across physical machines by adding or removing containers in the cluster dynamically based on resource usage and availability. Our experimental evaluation shows that V-Hadoop performs comparably to a fully-distributed Hadoop cluster. V-Hadoop allows for elastic clusters that can utilize the resources of the underlying physical infrastructure, providing significant management and cost benefits to the user.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[2] T. White, *Hadoop: The Definitive Guide*, ser. Oreilly and Associate Series. O'Reilly, 2012. [Online]. Available: https://books.google.ca/books?id=drbI_aro20oC

[3] J. R. et al. The hadoop ecosystem table. [Online]. Available: https://hadoopecosystemtable.github.io/

[4] R. Bohn and J. Short, "How Much Information? 2010 Report on Enterprise Server Information," Dec. 2011. [Online]. Available: http://hmi.ucsd.edu/howmuchinfo_research_report_consum_2010.php

[5] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security & Privacy*, no. 6, pp. 24–31, 2010.

[6] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, no. 3, pp. 81–84, 2014.

[7] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," University of California, Berkeley, Tech. Rep., 2010.

[8] J. Buell, "A benchmarking case study of virtualized hadoop performance on vmware vsphere 5," VMWare, Tech. Rep., 2011.

[9] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, "Evaluating mapreduce on virtual machines: The hadoop case," in *Cloud Computing*. Springer, 2009, pp. 519–528.

[10] D. Lezcano. (2015, oct) Lxc api documentation. [Online]. Available: https://linuxcontainers.org/lxc/documentation/

[11] M. Cohen and C. Pereira, "Cisco application-centric infrastructure (aci) and linux containers," Cisco, Tech. Rep., 2014.

[12] K.-T. Seo, H. Hwang, I. Moon, O. Kwon, and B. Kim, "Performance comparison analysis of linux container and virtual machine for building cloud," 2014.

[13] A. S. Foundation. (2015, jul) Apache hadoop 2.4.1 documentation. [Online]. Available: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html

[14] M. Gomes Xavier, M. Veiga Neves, and C. Fonticielha de Rose, "A performance comparison of container-based virtualization systems for mapreduce clusters," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, Feb 2014, pp. 299–306.

[15] M. Ishii, J. Han, and H. Makino, "Design and performance evaluation for hadoop clusters on virtualized environment," in *Information Networking (ICOIN), 2013 International Conference on*. IEEE, 2013, pp. 244–249.

A Hardware and Software Web-Based Environment for Energy Consumption Analysis in Mobile Devices

Sidartha A. L. Carvalho, Rafael N. Lima, Daniel C. Cunha and Abel G. Silva-Filho
Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), Recife-PE, Brazil
Emails: {salc, rnl, dcunha, agsf} at cin.ufpe.br

***Abstract*—In recent times, the number of mobile Android devices has been growing exponentially not only in the expanding popularity of low-cost mobile devices but also for the great number of functionalities and applications. This new range of features has highlighted the need for greater capacity in mobile batteries to provide an extended time of use. Several studies have been done to minimize the impact on the uncontrolled growth of applications, as well as analyze the suitable hardware configuration for a range of applications. In this sense, the objective of this work is to provide a Web-based environment which helps the designer to characterize mobile devices through automated testing and multiple devices simultaneously. The Web environment allows the designer to make assessments on the mobile device, remotely, without the need for measurement environment available. The use of multiple devices allows the designer to perform in different parallel measurements simultaneously. As the case study, an analysis involving video streaming, CPU processor load, and CPU fixed-frequency algorithms versus dynamic frequency scaling techniques were performed for two types of Android smartphones.**

I. Introduction

The number of mobile devices, such as smartphones and tablets, using Android operating system (OS) has grown in large scale in the last years, increasing from 200.000 devices activated per day in 2010 to 1.5 million devices activated per day in 2013 [1]. Nowadays, Android is the most used OS in mobile devices around the world, with a smartphone market share of 82.8% in 2015 [2]. Much of this growth is related to the increasing popularity of low-cost mobile devices. Besides that, mobiles devices have been manufactured with a rising number of functionalities, including communication technologies (Bluetooth, Wi-Fi, and near field communications), entertainment/multimedia options (for example, games and cameras), GPS-based applications, etc. These new features also contribute to the accelerated dissemination of the mobile devices.

Equipment with more features requires more processing power, which has led to the adoption of multicore processors on latest mobile devices. In this way, the development of more and more powerful applications in conjunction with the use of multicore platforms results in energy-hungry battery-powered mobile devices. In the face of this, the extending of the battery lifetime and, consequently, the time of use, is a concern of most mobile device design.

978-1-5090-3216-7/16/ $31.00

For developing techniques that allow energy efficiency, it is necessary to understand the energy consumption problem on mobile devices. To perform the mobile power characterization, it is indispensable to have an energy measurement infrastructure to obtain energy consumption profiles of mobile devices, in particular, smartphones. With this in mind, the motivation of this work is the lack of measurement environments (hardware and software integration) that assists in characterizing the energy consumption of mobile devices and allows automate tests.

In this paper, we propose a hardware and software Web-based environment that helps to obtain an energy characterization of mobile devices, automating tests and allowing this to multiple devices simultaneously. The proposed energy measurement environment provides the necessary infrastructure for measuring and testing functionalities or applications on mobile platforms. It is noteworthy to mention that this idea can also be applied for other mobile OSes, such as iOS and Windows Phone.

II. Related Works

Software testing on mobile devices is necessary to combat the fragmentation problem. Testing can be scalable when the same test is performed simultaneously on different devices. It saves time in testing. Several companies offer testing service in the cloud using real mobile devices, which reduces costs on equipment purchase and software deployment time on each device.

The proposed environment aims to complement the related works described in Table I. None of the reviewed studies enable software testing with energy analysis, even though this last one is considered essential in the development of techniques that increase usage time of battery-dependent devices.

The Power Monitor from Monsoon Solutions [3] offers a robust system for measuring the energy consumption of battery-dependent devices with a high sampling, but it is more expensive than Energino [4], an innovative data acquisition low-cost system. Energino is a real-time energy monitoring kit using Arduino microcontroller and an open platform with broad participation of the community.

Xamarin [5] has as the primary objective to take the suffering of testers, automating software tests quickly and with the use of a Web interface. In this tester, real devices are connected

Table I
RELATED WORKS FEATURES

	Our	[3]	[4]	[5]	[6]
Energy Consumption	√	√	√	X	X
Frequency Switching	√	X	X	X	X
On/Off CPU Cores	√	X	X	X	X
Web Interface	√	X	X	√	√
Statistical Analysis	√	X	X	X	X
Multiple Device, Platform, and Language	√	X	X	√	√

to a server that interacts through a Web interface allowing the user to install applications on different smartphones and tablets with different operating systems. The system uses Cucumber language to program the tests, which were converted to the target operating system commands and run on the device.

Monkey Mobile Cloud [6] is a service of testing replication with real devices too. A portion of Monkey Mobile Cloud environment is open source and offers a free library (MonkeyTalk Library) that translates commands written in proper language for equivalent commands in Android and iOS systems, allowing replication of multiplatform tests. The company also provides services in the cloud.

III. PROPOSED ENVIRONMENT

The proposed environment provides a Web-based platform for measurement and analysis of power consumption on multiple Android mobile devices under pre-defined experiments. The environment allows the user to select features and scenarios to test. For example, it is possible to select the data network the device should use (WiFi, 2G, 3G or 4G), the CPU operating frequency of each processor core, Android governor should be utilized, which devices will be tested and the number of iterations.

A large combination of scenarios and features configuration can be created using the proposal. Each scenario is composed of a list of action, for example, making phone calls, sending e-mails and/or SMS, opening Web browsers, playing videos or user actions within a non-native application in Android OS. Non-native application activities are characterized by a record of user touch events that the environment replicates in the specified applications. The proposed environment installs the mentioned application and opens automatically in each iteration; after that, the user recorded actions are replicated.

Fig. 1 illustrates the architecture of the proposed Web-based environment, which is divided into hardware environment (HE) and software environment (SE). The primary responsibility of the HE is the physical infrastructure needed to perform energy measurements on devices. The SE is responsible for processing data, communication with the platform components, Web interface, and communication with the user and results generation.

A. Hardware Environment

The HE is composed of a power supply that provides the power required for the operation of the smartphone, the measuring integrated circuit, the smartphone under test and the computer server. Any hardware that requires external power to work can be included in the HE, such as smartphones, tablets, embedded computers and sensors.

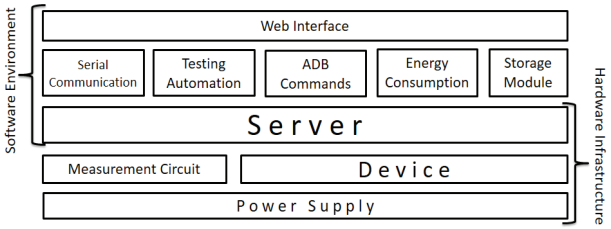


Figure 1. Architecture of the proposed Web-based environment.

1) Measurement Circuit + Power Supply: The measurement circuit consists of a microcontroller, an integrated circuit responsible for collecting current and voltage samples, a USB mini converter, and other electronic components. The circuit responsible for collecting current and voltage samples is connected between the power supply and the device being charged. The circuit sends data to the microcontroller that, in turn, transmit them to the server via USB connection.

Other devices perform measurements on hardware like the measurement circuit proposed. In a practical way and with a large number of samples per second, we have MonSoon Power Monitor [3] with 5k samples per second and Energino [4] with 10k samples per second. Building a new circuit was necessary to have more direct control and communication with the measuring circuit to automate the process. MonSoon Power Monitor above mentioned having a closed architecture that does not allow the communication machine-to-machine easily, which means that it is a black-box product in the market.

2) Devices (Smartphones): In the proposed infrastructure, it was included two smartphones with Android OS. Technical specifications of the devices are available in Table II.

Table II
SMARTPHONES USED TO TEST THE PROPOSED WEB-BASED ENVIRONMENT

	SP1	SP2
SoC	Exynos 4412	Snapdragon 600
Processor	ARM Cortex A9	Krait 300
Number of Cores	4	4
Frequencies (MHz)	200 to 1400	384 to 1890
Android Version	4.1.2	4.4.2

New smartphones or tablets can be added to the proposal even with other OSes such as iOS, Windows Phone, and others. The iOS can adapt ADB commands used in Android through an SSH connection established between the server and the device being measured. This connection allows running commands in the device shell, as it is done using ADB commands on Android. To automatize the process of adding new devices, information from each device may be obtained by ADB commands.

B. Software Environment

The SE consists of modules responsible for the generation of Web user interface, serial communication with the measuring circuit, testing automation using MonkeyTalk library, ADB commands management as network definitions and other low-level settings in the device under test, energy consumption analysis and storage module of the measurements.

1) Server: In Server side, Java and HTML5 (HTML + JavaScript) were used to build the Web interface. Android Debug Bridge (ADB) was used for communication with smartphones, while Java Simple Serial Connector (JSSC) library was used to connect the measurement circuit and the server. MonkeyTalk library was employed to aid in test replication, and PostGreSQL database to store energy measurements. At last, R language was employed to do statistical analysis.

The computer used as server application is composed of modules that allow the control and synchronization of peripheral devices. MonkeyTalk Library can translate MonkeyTalk commands (similar to natural language) into Android and iOS commands.

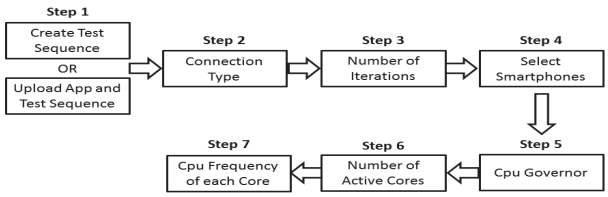


Figure 2. Web-based user interface use flow.

The communication control and the transmission of samples of current and voltage between the computer and the measurement circuit are serially performed by a USB cable. To establish a connection between smartphone and server, ADB commands are used for configuration setting, control parameters obtaining and Web interface updating. The Energy Consumption module is used to analyze and convert the sample values of current and voltage in energy consumed (measured in Joules), using numerical integration.

The use proposal flow is presented in Fig. 2. In the first step, it is possible to create a new test with some limited predefined features inside the Web interface or upload an application with a file defining a more complex test. The environment will translate this test to the appropriate OS and send commands in a time sequence to the analyzed device. In the second step, it is possible to choose the data network (2G, 3G, 4G or Wifi) that will be used in the test. After that, we should set the number of test iterations (Step 3), which devices should run the test (Step 4) followed by the Android governor to scale CPU frequency (Step 5). In the sixth step, we should establish the number of active cores of the processor and, in the last step, the maximum and minimum frequency of each processor core.

IV. Experimental Results

Three experiments were conducted to demonstrate the utilization of the Web-based proposed environment. In all experiments, the two smartphones previously described in Table II were used. In the first experiment, we selected the CPU frequencies that had the lower (best case), mean and higher (worst case) energy consumption. In the second experiment, the processor frequencies selected in Experiment one were compared with Android governors available on the devices.

In the third experiment, an Android application was developed to stress the processor to the desired use percentage CPU load. First, The proposed environment installs and executes the application. Then, all devices perform the test sequence. A test was created to fit the application to stress the processor on 25%, 50%, and 75% CPU load. All experiments were repeated 30 times to obtain statistical reliability in the results.

A. Experiment One: YouTube Video Streaming with Processor Frequencies

In this experiment, YouTube was initialized with a video link that was played for 120 seconds. A test sequence was created manually according to the first step of the Web-based user interface use flow (see Fig. 2). The connection type was assumed to be WiFi, the number of active cores was set to four (maximum), and all CPU frequencies were analyzed. The Android governor was set to UserSpace, where we can define a single CPU frequency without scaling.

We analyzed the average consumed energy for 30 iterations of video streaming experiment for the SP1 and SP. All iterations have a low standard deviation error based on the power values, lower than 5 points. The frequencies of 200 MHz and 300 MHz for SP1 and 384 MHz for SP2 were excluded because they presented a low performance, invalidating the test execution. For both cases, the frequency of 700 MHz appears to be more energy efficient for the analyzed context and tested smartphones. This information can be used to generate intelligent scheduling algorithms (as a new Android governor) to recognize system background and choose the appropriate CPU operating frequency for each context.

These results may indicate that the upcoming 700 MHz frequencies are the most energy-efficient options in a video streaming context for both smartphones. It gives us the insight to make a more detailed analysis of fixed-frequency and dynamic frequency scaling (DFS) algorithms.

B. Experiment Two: YouTube Video Streaming with Governors

Similar to the first experiment, we created a test sequence (Step 1 of the Web-based user interface use flow), set the number of active cores to four and assumed a WiFi connection. In each test execution, we utilized the UserSpace governor, and the range of CPU frequencies presented in Table II.

The boxplots graphs of the energy consumption for smartphones SP1 and SP2 were analyzed. For the SP1 device, we analyzed the fixed-frequency algorithms: 700 MHz, 1100 MHz, and 1400 MHz, as well as the DFS algorithms: OnDemand (OD), Performance (PE), PowerSave (PS), and PegasusQ (PQ). For the SP2 device, we analyzed the fixed-frequency algorithms: 702 MHz, 1134 MHz, and 1890 MHz, as well as the DFS algorithms: Interactive (IT), OD, PE, and

PS. The fixed-frequency algorithms are those that had lower (best case), mean and higher (worst case) energy consumption for each mobile device in the previous experiment.

The boxplots suggest that the fixed-frequency algorithms of 700 MHz (SP1) and 702 MHz (SP2) are the lower energy-efficient frequencies, with average energy consumption equal to 180.44 J (SP1) and 230.5 J (SP2). It is important to highlight that the fixed-frequency algorithms present lower energy consumption than the DFS algorithms. A possible justification is that a high rate of frequency changing can increase the energy consumption.

To confirm our hypotheses that using fixed-frequency algorithms (no frequency scaling) is more energy efficient than using DFS algorithms, we have to resort to statistical analysis. Firstly, we need to know if data have normality to choose the adequate method. Non-parametric Shapiro-Wilk and Lilliefors (Kolmogorov-Smirnov) tests were applied to check adherence data for normality. After that, we used the Friedman test, the non-parametric version of the analysis of variance (ANOVA), while the Nemenyi post-test was applied to determine which algorithms have significant differences.

The fixed-frequency algorithms around 700 MHz (700 MHz for the SP1 device and 702 MHz for SP2 one) can provide a low energy consumption, but they can lead to some delay in real-time applications that are CPU-bound. It is important to have another study that analyzes energy consumption and runtime to have a greater vision of this problem.

C. Experiment Three: Processor Load Stress

In the latter experiment, the energy measurement of the developed CPU stress Android application was made. We used the Upload App and Test Sequence option (Step 1 of the Web-based user interface use flow), set CPU governor to UserSpace and network type to none, the number of active cores (1, 2, 3, or 4) and the same CPU frequency to all cores. In the uploaded test sequence file, we defined touch events on the screen to simulate user actions in our uploaded Android application to stress mobile CPU accordingly with the user-defined load CPU percentage.

The application consisted of creating threads that perform mathematical operations to overload the processor to the desired level. The desired levels in the experiments were 25%, 50%, and 75% processor load. Initially, the percentage of 100% was tested but was removed from the operations, since it causes crashes on devices, inhibiting the tests.

We assumed the highest available CPU frequency in this experiment. As expected, for a fixed number of active cores, the higher the load processor percentage, the greater the energy consumption. Also, for an established load percentage, the increase in the number of active cores implies the increase of the energy consumption, but not in the same proportion.

From current consumption, it can be perceived that a CPU-bound process is more energy-efficient when it uses the maximum amount of active cores. In reaching this conclusion, proportion to knowing the average amount of current necessary for the execution of 1% of the load was performed. Making the aspect ratio to four active cores we have 25%, 50% and 75% load respectively: 640/25 = 25.6mA, 750/50 = 15mA and 900/75 = 12mA for each 1% processing load. Lower energy consumption is achieved using the same amount of work with a larger quantity of cores. The result followed true for all combination of active cores and both smartphones.

V. Concluding Remarks and Future Works

In this work, we proposed a hardware and software Web-based environment that helps to obtain an energy characterization of mobile devices and provides the necessary infrastructure for measuring and testing functionalities or applications on mobile platforms.

To test the proposal, three macro experiments involving video streaming, CPU processor load, and an analysis of CPU fixed-frequency algorithms versus dynamic frequency scaling techniques were performed for two smartphones with Android OS. The first and second experiments showed that CPU fixed-frequency algorithms around 700 MHz for video streaming are more energy-efficient than DFS algorithms, resulting in energy saving. In load processor experiment, it was possible to indicate the use of the maximum number of cores at the maximum frequency for CPU-bound processes. This information can provide energy saving for heavy-processing processes.

For future work, we intend to analyze more scenarios to identify the optimal-powered CPU frequency as it was performed for video streaming and CPU load stress application. This knowledge can be compiled in the development of a new governor for mobile OS devices. Also, the releasing of the proposed environment as a service through the Web interface and availability of source code for the academic community will permit researchers to be able to characterize the energy consumption of their techniques on real devices in an easy manner and adapt the proposal to an improved version. At last, we highlight that the proposed approach can incorporate other mobile devices, like iPhone and Windows Phone, allowing to test the behavior of other OSes.

VI. Acknowledgments

This research is supported by Motorola Mobility, LLC. Also, the authors thank to CNPq and FACEPE (IBPG-0731-1.03/12 and IBPG-1269-1.03/14), both Brazilian agencies, for partial financial support.

References

[1] Statista. (2014) The statistics portal. [Online]. Available: http://www.statista.com/statistics/278305/daily-activations-of-Android-devices

[2] IDC. (Q2, 2015) Smartphone os market share. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[3] MonSoon. (2014) Power monitor. [Online]. Available: https://www.msoon.com/LabEquipment/PowerMonitor

[4] K. Gomez *et al.*, "Energino: A hardware and software solution for energy consumption monitoring," *in Proc. WiOpt*, 2012.

[5] Xamarin. (2015) Mobile app development and app creation software. [Online]. Available: http://www.xamarin.com

[6] MonkeyTalk. (2014) Monkeytalk mobile app testing tool - cloudmonkey. [Online]. Available: https://www.cloudmonkeymobile.com

Energy Efficient File Distribution Problem and its Applications

Kshitiz Verma
Universidad Carlos III de Madrid, Spain
LNMIIT Jaipur, India
Email: vermasharp@gmail.com

Alberto García-Martínez
Universidad Carlos III de Madrid, Spain
Email: alberto@it.uc3m.es

Samar Agnihotri
IIT Mandi, India
Email: samar@iitmandi.ac.in

***Abstract*—Energy efficient networking has gained momentum in past one decade due to the Internet's ever increasing share in world's total energy consumption. It is crucial to study and reduce the energy consumption of tasks that are very heavily used in the Internet. In this paper, we focus on one such task, file distribution, and study its energy efficiency. We prove lower bounds on energy consumption for different scenarios that are relevant in today's Internet and design schemes that achieve the lower bounds when all the hosts have equal upload and equal download capacities. We evaluate our theoretical results numerically to generalized scenarios as well. We show that our schemes of file distribution can save as much as 50% energy compared to other energy efficient P2P methods proposed in the literature.**

I. Introduction

It is a need of the hour to consider energy efficiency while designing any engineering artifact for both economical as well as environmental reasons. Internet with all its associated devices consumes 2-10% of the total world power consumption [1] [2]. The root cause for energy waste in the Internet is because all the devices are powered on all the times even though the rate of utilization of these devices varies drastically over time. Apart from the fact that most of the hardware is agnostic to energy consumption, also the services and protocols that run on the hardware do not have energy as a metric.

In this paper, we focus on an extremely common task in the Internet – file sharing. There have been studies confirming that users leave PCs on just for file downloading [3]. In such a scenario, it is important to manage the energy consumed by the hosts so that the download takes place without wasting energy. For this purpose, there is a need to design file distribution algorithms to minimize energy consumption. We propose a model for energy efficient file distribution, prove lower bounds and design schemes achieving the bounds. However, due to lack of space, instead of including all the algorithms we present examples for case $u > d$ in Fig. 2 and Fig. 4.

The idea behind our algorithms is that all the hosts who want to receive the file, should upload as well as download. Doing only one of the two operations is sub-optimal and should be minimized. Note that we consider only those cases for which all the hosts have the same upload and download capacity. Moreover, we also assume that one capacity is an integral multiple of the other. If the upload capacity is higher than the download capacity, then download capacity is the bottleneck, implying that all the hosts should receive at their full capacity. On the other hand, if download capacity is higher than the upload capacity then all the hosts should upload at their full capacity. The schemes depend very much on the power consumption of the hosts. We do not put any restriction on the power consumption of hosts for the case in which upload capacity is greater than the download capacity. However, if the download capacity is greater, then we assume that all the hosts consume the same power.

Finally, we compare our schemes with the existing energy efficient methods for P2P file distribution that use proxy [4] and the family of algorithms from [5] and the legacy energy agnostic BitTorrent. We find that our schemes save at least 50% energy compared to the existing energy efficient approaches and up to three orders of magnitude compared to the legacy BitTorrent.

A. Our Contributions

The contributions of this paper are summarized next.

- Lower bounds and schemes for different cases (Fig. 1). The problem at the root is computationally tractable which is further divided into two cases depending on the relationship between upload and download capacities. Each case is further subdivided depending on the relation between the number of blocks and the number of clients.

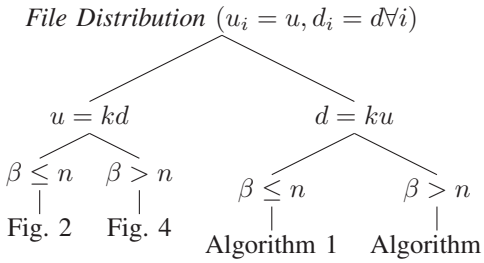


Fig. 1: This figure summarizes the algorithmic results in the paper. Throughout the paper, u and d represent upload and download capacities, $k \geq 2$ is an integer, β and n represent the number of blocks and the number of clients respectively.

978-1-5090-3216-7/16/$31.00

- Having high download to upload capacity ratio is not very crucial. If all the hosts have the same power consumption, then $\frac{d}{u} = 2$ is close to optimal.
- Having high upload to download capacity ratio improves energy efficiency only for the server, i.e., even if all the hosts other than the server have $\frac{u}{d} = 2$, schemes can be optimal consuming the same energy as for any $k > 2$.
- Through numerical evaluation, we show that our schemes are at least 50% more energy efficient than the similar other approaches. We also study the impact of the block size on energy consumption.

The rest of the paper is organized as follows: Section II lays down the assumptions and system model. Section III presents the energy efficient algorithms designed in the paper. Due to lack to space we have not included all the algorithms, we rather provide examples of one of them. Section IV presents the numerical evaluation of the algorithms. Section V discusses the related work and we finally conclude in Section VI.

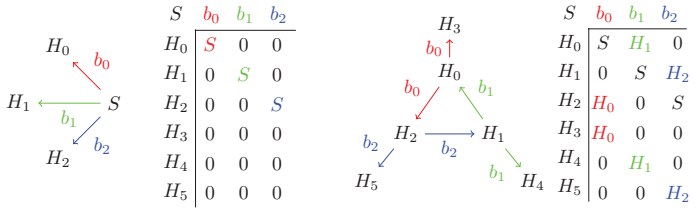
II. Assumptions and System Model

Consider n hosts and a server S, i.e., total $n+1$ hosts. The server has a file that is to be received by all the hosts. A host i or H_i for short, has upload capacity u_i, download capacity d_i and power consumption P_i. For the server these values are u_S, d_S and P_s respectively. The file is divided into β blocks, each of size s. We assume a complete graph, i.e., all the hosts are reachable to all the other hosts. A host can start uploading a block to another host only if it has received a block completely. One block of file can be downloaded only from one host, i.e., two hosts cannot upload the same block to a host simultaneously. However, there is no restriction on the number of blocks a host can download or upload. A host is on if and only if it is at least uploading or downloading a block. We also assume that switching on/off happens instantaneously, and hence, no energy is consumed in switching on/off. However, we relax this condition in the numerical evaluation.

As proven in [6], the problem with variable upload capacities is NP-hard. Hence, from now on, we assume that $u_S = u_i = u,\ \forall i \in \{0, 1, 2, \cdots n-1\}$. We also assume that $d_S = d_i = d,\ \forall i \in \{0, 1, 2, \cdots n-1\}$. Both u and d are related by a positive integer k, such that, $k = \frac{u}{d}$ or $k = \frac{d}{u}$, depending on whether $u > d$ or $d > u$. For case $d = u$, user may refer to [6]. We also assume that power consumption of a host is constant irrespective of whether a host is idle, receiving and/or uploading. and does not change with the operations that a host may perform during the file distribution process. We assume that time is slotted and an arbitrary time slot is represented as τ, then the duration of each slot is given as,

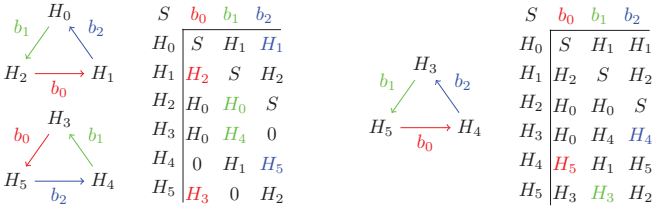
$$\tau_{\text{duration}} = \frac{s}{\min\{u, d\}} \tag{1}$$

Transfer in a slot can be modeled as a directed graph (called *transfer graph*) in which there is a directed edge from sender to receiver. We use this graph to show the transfers in the examples of algorithms in the next section.



(a) The server can serve at most three blocks in a slot.

(b) Some hosts upload to multiple hosts.



(c) Two disjoint transfer graphs in the same slot.

(d) The three hosts form another cycle.

Fig. 2: Example of how our algorithm works when $u = 3d$, for $n = 6$ and $\beta = 3$. Note that in the first slot, the server serves all the three blocks and switches off. In the next slot, the fact that $u > d$ is used and three hosts upload to two hosts. All the hosts form a cycle to serve the blocks to each other.

III. Energy Optimal Algorithms

A. Upload > Download Capacity

Theorem 1: The energy required by any scheme z to distribute a file divided into β blocks among n clients when $k = \frac{u}{d} > 1$, satisfies

$$E(z) \geq \lceil \frac{\beta}{k} \rceil \cdot P_S \cdot \frac{s}{d} + \beta \cdot \frac{s}{d} \cdot \sum_{i=0}^{n-1} P_i \tag{2}$$

Sketch of proof: Each host has to be active for at least β slots to receive the complete file. The server, however, can upload to k different hosts. It needs to be on for at least $\lceil \frac{\beta}{k} \rceil$ slots.

An example of our algorithms for this case is shown in Fig. 2 which achieves the lower bound mentioned in Equation 2. Hence, we have optimal schemes for the case $u = kd$.

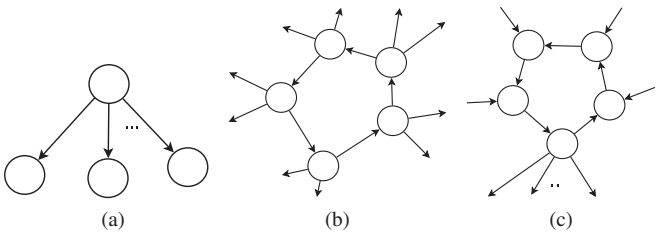


Fig. 3: Transfer graphs in case of upload > download capacity.

Fig. 3 discusses various kinds of transfer graphs. Observe that transfer graphs in Fig. 3c cannot be part of any optimal scheme because it has hosts which are receiving from multiple hosts. Likes of Fig. 3b are used in optimal algorithms for case $n > \beta$, as illustrated in Fig. 2 because hosts can upload while they are cycling among each other. However, it is worth noticing that having $\frac{u}{d} > 2$ for the hosts other than the server provides no energy gains.

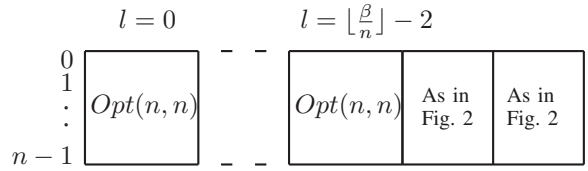


Fig. 4: A representation of optimal algorithm for $\beta > n$ when $u = kd$.

Basically the scenario with $\beta > n$ is divided into $\lfloor \frac{\beta}{n} \rfloor - 1$ subroutines of $\beta = n$, as shown in Fig. 4. The server serves diagonal in all of them and then they cycle as in Fig. 2c. The remaining number of blocks is divided into two subproblems in which the blocks are transfered after one subproblem finishes. These subproblems can be solved in the similar manner as the example presented in Fig. 2

B. Download > Upload Capacity

We assume that all the hosts have the same power consumption, i.e., $P_i = P \ \forall i \in \{S, 0, 1, \cdots, n-1\}$. We have the following theorem for the upper bound on the lower bound for energy consumption.

Theorem 2: If all the hosts have equal power consumption, with $k \geq 2$, $E_{\min}(z) \leq n(\beta+1) \cdot P\frac{s}{u}$, if $\beta > n$, then Algorithm 1 describes a distribution scheme with energy consumption

$$E(z) \leq \left(n(\beta+1) + \left\lceil \frac{2\beta}{n(n-1)} \right\rceil \right) \cdot P\frac{s}{u} \qquad (3)$$

Thus, Algorithm 1 provides a scheme for homogeneous case which is off from the lower bound by an additive factor of $O(\frac{1}{n^2})$ and hence is quasi-optimal. We can apply Algorithm 1 to cases with condition $\beta \leq n$ as well. Also note that Algorithm 1 uses at most $k = 2$. Hence, having high download to upload capacity does not reduce energy consumption.

Algorithm 1 can be used for both the cases, i.e, $\beta \leq n$ and $\beta > n$. The hosts can be made to form groups of size n_i in a group G_i such that $\frac{n_i(n_i+1)}{2} \leq \beta$. Keep on repeating this until all the hosts get the file.

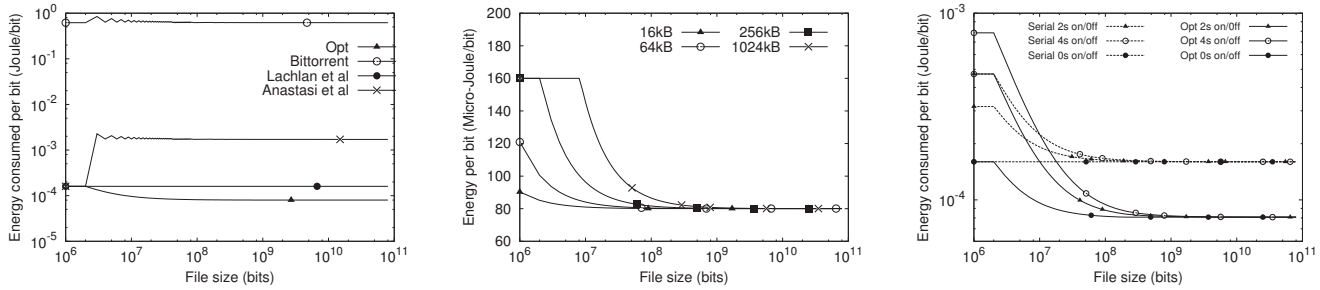
Algorithm 1 Scheme for case $d = ku$

for slot $j = 0 : n-1$
  $S \xrightarrow{j} H_j$
while $(\beta > 1 + \sum_{i=1}^{\text{var}} (n-i))$ **do**
  var++
end while
var = var -1
for $(\xi = 1; \xi \leq \text{var}; \xi++)$ **do**
  for slot $j = \sum_{i=1}^{\xi} (n-i) + 1 : \min\left\{ \sum_{i=1}^{\xi+1} (n-i), \beta - 1 \right\}$
    $S \xrightarrow{j} H_{n-1}$
    $H_0 \xrightarrow{j - \sum_{i=1}^{\xi} (n-i) - 1} H_{n-\xi}$
    for $i = 1 : n-1$
      $H_i \xrightarrow{(i+j-n) \bmod \beta} H_{i-1}$
end for
$\xi = 1$
dif=0
for slot $j = \beta : \textbf{\textit{var}} + \sum_{i=1}^{\textbf{\textit{var}}+1} (n-i)$
if $\xi = \text{var}$ & $\left(\sum_{i=1}^{\text{var}+1} (n-i) - (\beta - 1) \right) \neq 0$ **then**
    for $i = 1 : n - \xi$
      $H_i \xrightarrow{(i+j-n) \bmod \beta} H_{i-1}$
    $H_0 \xrightarrow{\min\left\{ \sum_{i=1}^{\xi+1} (n-i), \beta-1 \right\} - \sum_{i=1}^{\xi} (n-i) + \text{dif}} H_{n-\xi}$
  dif++
else
    for $i = 1 : n - \xi$
      $H_i \xrightarrow{(i+j-n) \bmod \beta} H_{i-1}$
    $H_0 \xrightarrow{\min\left\{ \sum_{i=1}^{\xi+1} (n-i), \beta-1 \right\} - \sum_{i=1}^{\xi} (n-i)} H_{n-\xi}$
  ξ++
end if
for slot $j : n + \beta - 2$
  $H_0 \xrightarrow{(j - \text{var}) \bmod \beta} H_{n-\text{var}-1}$
  for $i = 0 : n - \text{var}$
    $H_i \xrightarrow{(i+j-n) \bmod \beta} H_{i-1}$

IV. Numerical Evaluation

Fig. 5a presents a comparison between the optimal algorithms (*Opt*) for $d > u$ (results are similar for $d < u$) other proposals. We observe that the BitTorrent without energy efficiency is very energy expensive. The other two approaches considered, improve the energy consumption of BitTorrent and already achieve three to four orders of magnitude improvement in energy savings. Our algorithm is the best among all these that provides 50% improvement over the next best *Lachlan et. al* [5] and more than an order of magnitude compared to *Anastasi et. al* [4]. To compute the energy consumed by *Anastasi et. al*, we assume that there are 50 different corporations each having 20 hosts. Thus, fifty hosts participate in P2P file distribution. Once they receive the file they send to their hosts. The three energy efficient schemes coincide when the file is divided in only one block. In this case, all the energy efficient schemes upload each block to each host one by one. Only the hosts uploading and downloading are kept on. However, this is not true for BitTorrent, since all the hosts are on. Therefore, the energy consumption is high even when the file consists of just one block. This increases the energy consumption per bit for BitTorrent.

Fig. 5b shows the impact of block size on energy consumed by our algorithms. As the file size increases, impact of block size decreases for larger blocks because energy is higher for small files. It is so because lesser number of blocks are there and distribution is more sequential. However, as the file size increases, the parallelism in the optimal schemes is exploited and energy consumption is lowered.



(a) Comparison between 3 P2P schemes and ours. (b) Impact of block size on *Opt*. (c) Impact of non-zero on/off energy cost.

Fig. 5: All the graphs assume that P=80W, u=10Mbps, n=1000. Block size is 256kB unless mentioned otherwise. The results are similar for n=5000, 500 or 50.

Fig. 5c presents the energy consumed by our scheme in comparison to the *serial* scheme considering a switch on/off time equal to 2 and 4s. In *serial* scheme all the hosts receive all the blocks from the server one by one. They switch off after receiving. As expected, the on/off costs increase the energy per bit consumed by all schemes. This increment is more pronounced for small file sizes, where we see that on/off costs make the performance of our scheme closer (but still better) to the serial scheme. Conversely, for medium/large file sizes, the contribution of on/off costs to the total energy consumed by a scheme becomes marginal, and the performance of both the optimal scheme and the serial scheme approaches the one in the case without on/off costs.

V. Related Work

Energy efficiency in file distribution is a well studied problem in the literature. However, no characterization of its complexity and analytical analysis exists beyond for some basic cases as provided by [6]. Sucevic et. al. have studied the same problem [7], [8] but their analysis is really limited and works only for very small number of hosts. [9] investigates green bittorrent via simulations. Studies like [10]–[12] have explored the file distribution problem from the point of view of optimizing time, but as shown in [5], the two optimization problems are different. For a more comprehensive collection of the state of the art methods for energy efficiency in P2P file sharing, we refer the reader to [13], [14]. The work presented in this paper is close to [6] but their analysis is very limited. They provide analytical results for case $d = u$ only. In this paper, we have explored almost all the tractable versions of the problem. The problem of file sharing has been studied in many contexts, particularly from the point of view of finishing time. However, the schemes optimizing the finish time can penalize energy very much, up to an order of magnitude [5].

VI. Conclusion

File distribution is a ubiquitous task in the Internet and having even little energy savings can lead to high absolute energy gains. In this paper, we present algorithms that enable energy savings on top of the savings from other methods. Our methods are fairly general to be applied to not just P2P file sharing but they can be used in software distribution, replicating files in cloud, synchronization of servers of a content distribution network, etc. It is also worth emphasizing that our techniques improve with advances in energy efficient hardware. The quicker the hardware can go to sleep and wake up, the better our algorithms will perform. Hence, our results will become more relevant as the energy efficiency of the devices increases.

References

[1] M. Gupta and S. Singh, "Greening of the Internet," in *SIGCOMM*, 2003.

[2] G. Fettweis and E. Zimmermann, "Ict energy consumption-trends and challenges," in *Proceedings of the 11th International Symposium on Wireless Personal Multimedia Communications*, vol. 2, no. 4, 2008, p. 6.

[3] G. Anastasi, S. Brienza, G. L. Re, and M. Ortolani, "Energy efficient protocol design," *Green Communications: Principles, Concepts and Practice*, pp. 339–360, 2015.

[4] G. Anastasi, I. Giannetti, and A. Passarella, "A bittorrent proxy for green Internet file sharing: Design and experimental evaluation," *Computer Communications*, vol. 33, no. 7, pp. 794–802, 2010.

[5] L. L. Andrew, A. Sucevic, and T. T. Nguyen, "Balancing peer and server energy consumption in large peer-to-peer file distribution systems," in *IEEE Online Conference on Green Communications (GreenCom)*, 2011, pp. 76–81.

[6] K. Verma, G. Rizzo, A. Fernández Anta, R. C. Rumín, A. Azcorra, S. Zaks, and A. García-Martínez, "Energy-optimal collaborative file distribution in wired networks," *Peer-to-Peer Networking and Applications*, pp. 1–20, 2016. [Online]. Available: http://dx.doi.org/10.1007/s12083-016-0453-4

[7] A. Sucevic, L. Andrew, and T. Nguyen, "Powering down for energy efficient peer-to-peer file distribution," *ACM Sigmetrics Workshops, GreenMetrics*, 2011.

[8] A. Sucevic, L. L. Andrew, T. T. Nguyen *et al.*, "Minimising peer on-time for energy efficient peer-to-peer file distribution," 2012.

[9] J. Blackburn and K. Christensen, "A simulation study of a new green bittorrent," in *Communications Workshops, ICC*, 2009, pp. 1–6.

[10] K.-S. Goetzmann, T. Harks, M. Klimm, and K. Miller, "Optimal file distribution in peer-to-peer networks," in *Algorithms and Computation*. Springer, 2011, pp. 210–219.

[11] J. Mundinger, R. Weber, and G. Weiss, "Optimal scheduling of peer-to-peer file dissemination," *Journal of Scheduling*, vol. 11, no. 2, pp. 105–120, 2008.

[12] G. M. Ezovski, A. Tang, and L. L. Andrew, "Minimizing average finish time in P2P networks," in *IEEE Infocom*, 2009.

[13] A. Malatras, F. Peng, and B. Hirsbrunner, "Energy-efficient peer-to-peer networking and overlays," *Handbook on Green Information and Communication Systems*, 2012.

[14] S. Brienza, S. E. Cebeci, S. S. Masoumzadeh, H. Hlavacs, G. Anastasi *et al.*, "A survey on energy efficiency in P2P systems: File distribution, content streaming, and epidemics," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 36, 2015.

On the Use of Nonlinear Methods for Low-Power CPU Frequency Prediction Based on Android Context Variables

Sidartha A. L. Carvalho, Daniel C. Cunha and Abel G. Silva-Filho
Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), Recife-PE, Brazil
Emails: {salc, dcunha, agsf} at cin.ufpe.br

***Abstract*—The objective of this paper is to analyze the use of nonlinear models to predict the CPU frequency that reaches the lowest power consumption of a smartphone based on Android OS context variables. Artificial neural networks (ANNs) and k-nearest neighbors (k-NN) techniques are investigated, and their results are compared to those obtained by the linear method (LM). Experimental results indicate the k-NN technique is the best option in terms of model accuracy and performance when compared to the other prediction models.**

I. INTRODUCTION

In recent years, energy consumption has become a critical aspect not only in the design of embedded systems and mobile operating systems but also in the development of mobile applications. Predicting and optimization of power consumption of mobile devices has emerged as a research topic in embedded and mobile architectures [1]. Various power consumption management strategies have been developed for Android systems. For example, some schemes utilize dynamic voltage and frequency scaling (DVFS) or dynamic power management (DPM) to extend the battery life [2]. DVFS and DPM schemes attempt to achieve power reduction based on the central processing unit (CPU) utilization. When CPU usage is low, there is no demand for high performance, and because of that, the CPU frequency is set lower [2].

Other relevant aspects related to the power consumption management are the running application characteristics [3]. For a clear understanding of the behavior of the power consumption of mobile devices, it is important to identify the presence of patterns in user activity [1]. By using application-specific knowledge, it is possible to set the CPU frequency closer to the low-power frequency value, thus saving energy [3]. For identifying this optimal configuration, we should have a fine-grained model characterizing contexts to measure the influence of the CPU frequency on the energy consumption.

Energy consumption measuring (also called profiling) can be done either at hardware or operating system (OS) level. At OS level, power profiling generates a model where it is possible to measure power consumption given some input features. Linear regression has been widely used for power modeling of processors [4]. In this context, the most common approach to generate CPU power consumption models is the linear method (LM) [5]. However, the relationship between user activity and power consumption does not follow a linear relation. For example, employing the DVFS technique, it is possible to reduce the CPU operating frequency to decrease the consumed power [5]. On the other hand, in current smartphones, the power consumption depends not only on the processor but also on a lot of subsystems and applications running in the background. This aspect indicates that a nonlinear power modeling can be an attractive option to investigate.

978-1-5090-3216-7/16/ $31.00

With this in mind, the objective of this paper is to analyze the use of nonlinear models to predict the CPU frequency that reaches the lowest power consumption of a smartphone based on Android OS context variables. The nonlinear models to be investigated are artificial neural networks (ANNs) and k-nearest neighbors (k-NN) techniques. Results obtained by these nonlinear methods are compared each other and with the LM results.

The remainder of this paper is organized as follows. Section II highlights some key related works, while Section III describes the proposed approach. In Section IV, our experimental results and models evaluation are described. Finally, conclusion and future works are drawn in Section V.

II. RELATED WORKS

Some researchers have been using ML algorithms in a mobile user context to obtain energy savings, [6] uses linear and nonlinear classification models were used to predict user data/location based on spatiotemporal and device contexts aiming energy savings. By using ANN and k-NN, it was achieved a prediction accuracy around 90%, while LMs ranged an accuracy between 60% and 90%. The used k-NN reached 25% of improvement compared to rate logging algorithm (VRL) in energy savings. k-NN presented low memory and CPU usage in training time, but with a high overhead in prediction time when using a large quantity of data (5 days or more). Low memory and CPU usage are ideal characteristics in a mobile environment, reinforcing our results.

In [7], a mobile power model based on the number of active cores and CPU frequency, and a new CPU governor algorithm called Medusa was proposed. This governor set the CPU frequency to an optimal value based on CPU load and some threshold, turning on all the cores before setting frequency up to the threshold. Using Medusa governor and the low-power CPU frequency values, it was obtained an energy consumption

reduction from 12% to 26%. We use a self-learning strategy to predict low-power frequency, while in [7] the low-power frequency values were obtained for three benchmarks and set manually inside the proposed governor.

In [8], the proposed ML power model was based on Linux system calls, a low-level method that provides a fine-grained power prediction. A fuzzy neural network was used to train data and compare the predicted power value with real hardware measurements. To validate the proposal, three Android applications (Facebook, Google Chrome, and Gmail) were used as benchmarks. In our proposal, a system level energy measurement was done, we tried to explore the low-power CPU frequency impacts into the Android system environment.

Power Tutor (PT) is an Android application that uses LM to predict power consumption from a variety of smartphone features like the processor, display, WiFi, GPS, cellular, and so on [9]. The PT power model was built based on three types of smartphones and exported to others. When we compared with our model, we used the coefficients presented in the open source code of that application [10] with a phone model that is the closest to ours, when in [9], only the LM coefficients from a smartphone with two CPU frequencies were presented. From the gap between smartphones characteristics and features, the PT power model predicted a very low power consumption compared with our hardware measurements, because of that, we decided to exclude the comparison in the results.

III. PROPOSED APPROACH

A. Proposed Methodology

The flow of the proposed methodology is illustrated in Figure 1. Our proposal can be represented by an off-line procedure and an on-line one. The off-line procedure is composed of a training stage and a testing stage. In the first stage, the nonlinear prediction model is built by using the collected database. In the table data, we have the number of active CPU cores, the CPU frequency, the CPU load, and the running application. Finally, the target or output variable is the consumed power $P_f(\cdot)$, measured in mW. The first stage of the off-line procedure is shown in Figure 1(a).

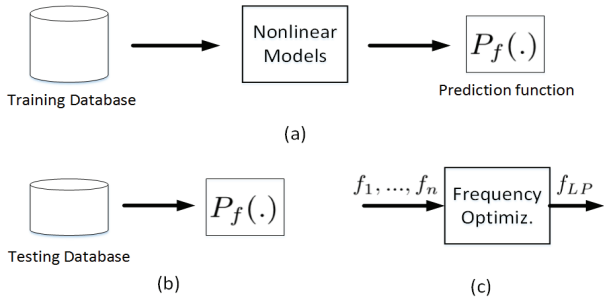


Figure 1. Flow diagram of the proposed approach: (a) Training stage (prediction of the power consumption). (b) Testing stage (validation of the nonlinear model). (c) Frequency optimization (obtaining of the low-power CPU frequency).

The second stage of the off-line procedure, indicated by Figure 1(b), is called testing stage. In this step, a testing database is used to validate the nonlinear model, computing the consumed power. After the model validation, a frequency optimization is executed using an on-line procedure. The objective of the on-line procedure is to find the low-power CPU frequency (f_{LP}) by means of the minimization of the power consumption. For this purpose, all available CPU frequencies are used. This on-line procedure is represented by the Figure 1(c).

B. Experimental Setup and Model Evaluation Methodology

A data acquisition board was developed to get current and voltage samples with a periodicity of approximately 300 samples per second. A PS-1500 ICEL power supply is used to power up the mobile. For measuring current values, Adafruit integrated circuit contained in INA219 was used. An Arduino UNO was utilized to read data from the Adafruit circuit and send them to a computer. At last, a Motorola XT1033 smartphone with an ARM Cortex A7 processor and running Android OS 4.4.4 KitKat was employed in the experimental environment, since this is the most used OS version on Android smartphones [11].

An Android application called Context Logger (CL) was evolved to capture variables from Android OS. This application runs as an Android service and captures context and device attribute at a 0.25-second interval. Our CL application collects 65 system attributes, such as processor information, date and time, battery level and characteristics, network and sensors data, display and applications running in foreground and background. For our purpose, we will use only the features that were mentioned in Subsection III-A.

Four benchmarks were carried on for all CPU frequencies available in the smartphone. The executed benchmarks were Google Chrome, Facebook (FB), Video Decoder (VD) and YouTube (YT). User actions were simulated in these benchmarks, except in VD. We collected samples from 5 minutes of each available frequency (300, 384, 600, 787, 998, 1094, and 1190 MHz) for each benchmark. On average, we had 1200 context variables samples and 90000 power samples for each CPU frequency, 8400 context variables and 630000 power samples at all.

The collected variables from CL application were unified to the power consumption samples. As the sampling rates of the data acquisition board and the CL application are different (300 and four samples per second, respectively), we had to do an adjustment. Specifically, for each CL sample, we computed the average of 75 samples from the power data acquisition board.

To evaluate our methodology, we analyze these data using the nonlinear models adopted (ANN and k-NN regression) and the LM using four input variables and one output variable.

About nonlinear models, we firstly used a feedforward ANN with random initialization weights. The ANN was trained using the H2O R package and the impact of the number of neurons in accuracy and training time was analyzed. The second nonlinear model was the k-NN algorithm, that was

implemented by the FNN R package. In this case, the impact of the number of neighbors (k) was analyzed.

To attest our proposal, we employ a validation technique called K-fold cross-validation, a method used to evaluate the model accuracy [12]. This approach is characterized by a re-sampling training and testing data in which the samples are randomly split into K sets of approximately equal size to be used as the training and testing K-folds. The common value adopted for K is 10. Also, the data are partitioned into subsets with 80% and 20% for training and testing, respectively.

IV. NUMERICAL RESULTS

In this section, we performed an analysis to identify, on average, the CPU frequency that reaches the lowest power consumption (f_{LP}) for each benchmark using the nonlinear models explained before. The results obtained by the LM approach was used as a reference.

A. Low-Power CPU Frequency Prediction

For each benchmark and CPU frequency value, both mentioned in Subsection III-B, we applied the LM and the nonlinear models to predict the power consumption. The objective is to find the f_{LP} during a continuous use of the smartphone.

Figure 2 illustrates the average consumed power for all available CPU frequencies considering the Google Chrome benchmark. By using the observed data (real values), we can see that, from 600 MHz on, the higher the CPU frequency, the higher the power consumption. Besides, the real values show us that f_{LP} = 600 MHz for the Google Chrome benchmark. Concerning the prediction models, we can observe that the nonlinear models fit the real data well and also result in 600 MHz as the power-optimal CPU frequency. Even though, the LM prediction is not a good option for all CPU frequency range, since its predicted values were higher than the measured mean power (Obs).

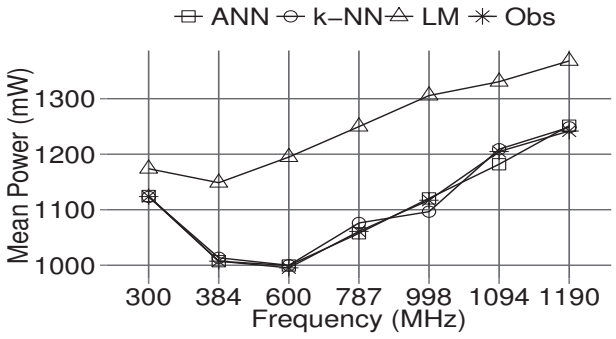


Figure 2. Average consumed power for all available CPU frequencies considering the Google Chrome benchmark.

In the FB benchmark, we can notice a different behavior of the mean power with the increasing of the CPU frequency when compared to the previous benchmark. Again, using the observed data, we can see that f_{LP} = 787 MHz. Similar to the Chrome benchmark, the LM approach does not prove to be a good predictor at 300, 384 and 600 MHz. In spite of that, the LM predictor fit approximately the observed data from 787 MHz on. Regarding the nonlinear models, ANN and k-NN predicted very close values when compared to real data (except for the two highest frequencies in k-NN) and result in 787 MHz as the f_{LP}.

In the VD benchmark, the LM predicted f_{LP} = 300 MHz, in spite of the observed values indicate that the f_{LP} = 384 MHz. On the other hand, the nonlinear models predicted the f_{LP} similar to that expressed by the observed values. It is important to highlight that both nonlinear models fit the observed data in all frequency range reliably. At the same time, the predictions obtained by the LM were higher than the observed values for all frequencies, as it was verified for Chrome benchmark.

Finally, in the YT benchmark, similar to the VD, the LM also predicted 300 MHz as the f_{LP}, although 384 MHz is the f_{LP} specified by the observed values. Again, not only ANN but also k-NN matched the observed values in all frequency range and also reached the f_{LP}.

The fact that higher CPU frequency reached low power consumption can be justified by the CPU load and the leakage power. The same CPU frequency executing with higher CPU load lead to higher power consumption [13], also high CPU loads inhibits the CPU to enter in deeper idle states. Higher CPU frequencies imply in more heat (power dissipation) that leads to more leakage power [14] leading to higher power consumption, even with lower CPU load.

B. Model Accuracy Evaluation

For measuring the model accuracy, we used three error metrics: MSE, MAE, and MAPE. Together with, we used the 10-fold cross-validation technique to compute the average of the error metrics and the results are shown in Table I.

Table I
AVERAGE ERROR METRICS OBTAINED BY USING 10-FOLD CROSS-VALIDATION.

Model	MSE	MAE	MAPE
LM	47636.47	171.75	0.16
ANN ($n = 3$)	41042.97	115.18	0.12
ANN ($n = 10$)	44783.54	120.21	0.12
ANN ($n = 100$)	44537.46	120.11	0.12
ANN ($n = 1000$)	46884.86	124.43	0.13
k-NN ($k = 3$)	41246.90	140.17	0.12
k-NN ($k = 10$)	30517.03	120.08	0.10
k-NN ($k = 100$)	26623.58	111.61	0.09
k-NN ($k = 1000$)	36728.76	150.32	0.14

ANN was evaluated for four values of neurons: $n = 3, 10, 100$, and 1000. For k-NN model, we evaluated the performance with the following number of neighbors: $k = 3, 10, 100$, and 1000.

Concerning ANNs, all tested configurations had approximately the same average error metrics. Indeed, the ANN with three neurons overcame the other ANNs with more neurons. One possible explanation is that a higher number of neurons can cause over-fitting in ANNs, so instead of improving, many neurons can worse the prediction accuracy.

Concerning k-NN, the configuration with $k = 100$ neighbors, overcame all the other algorithms for the whole error metrics. The worst case was for the LM because it has the highest error values for MSE, MAE, and MAPE. It is important to observe that k-NN for $k = 1000$ obtained a bad accuracy when compared with other k-NNs ($k = 3, 10, 100$) because a high number of neighbors considered to calculate the k-NN distance can lead to include classes very far from the analyzed point and predict values out of the scope.

Another analysis that can be done refers to the complexity of the prediction models. For this purpose, we define the training time t_{tr} as the required time to adjust the model to a lower training error metric, and the prediction time t_{pd} as the time to use the built model to predict the output value based on some input variables. Table II shows the average computational cost for each prediction model considered in this work. For the hardware platform considered, the lowest times (training and prediction) were in the order of magnitude of tenths of seconds. The prediction model with the lowest training time was the LM. However, this model presented the worse accuracy in error metrics, as we mentioned before. About the prediction time, all k-NN models presented around the same minimum value being $k = 3$ and $k = 10$, followed by LM and ANNs.

Table II
AVERAGE COMPUTATIONAL COST FOR EACH PREDICTION MODEL.

Model	**Average t_{tr}**	**Average t_{pd}**
LM	1.00	15.00
ANN ($n = 3$)	65.41	4545.00
ANN ($n = 10$)	73.25	4590.00
ANN ($n = 100$)	105.41	4695.00
ANN ($n = 1000$)	293.53	4805.00
k-NN ($k = 3$)	3.07	1.00
k-NN ($k = 10$)	4.02	1.00
k-NN ($k = 100$)	4.43	1.50
k-NN ($k = 1000$)	15.46	1.50

Based on the results, it can be seen that k-NN model has the best complexity-performance trade-off when compared with the other prediction models. Thus, the performed analysis suggest that k-NN ($k = 100$) is the best option among all tested prediction models for our test database. We believe that this value of k is justified by the highest accuracy and a relative low training and prediction times.

Finally, we present a summary of the comparison between the related works presented in Section II. Table III highlights some of our contributions as the use of DVFS reaching energy savings; the Context Logger App used to capture the smartphone/user context, and the proposed methodology to acquire the f_{LP}.

Table III
SUMMARY OF THE RELATED WORKS.

	DVFS	CL App	f_{LP} Acquiring	Energy Savings
Proposal	$\surd$	$\surd$	Automatic	NA
[6]	NA	$\surd$	NA	12% to 82%
[8]	NA	$\surd$	NA	NA
[7]	$\surd$	NA	Manual	12% to 26%
[9]	NA	$\surd$	NA	NA

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we analyzed the use of nonlinear models to predict the CPU frequency that reaches the lowest power consumption of a smartphone based on Android OS context variables. The nonlinear models investigated were artificial neural networks and k-nearest neighbors techniques. Our proposal can be applied in a new Android governor to save energy consumption of the smartphone. In addition, the low-power CPU frequency can be dynamically found during the smartphone use. Among the nonlinear models that we verified, the best option was the k-NN using 100 neighbors, considering the complexity-performance trade-off.

Work is in progress to implement this proposal as a governor policy in Android kernel and evaluate power savings and performance gains. Also, we intend to analyze not only other structures of ANNs and k-NN techniques with different kernels, but also other methods for regression aiming power prediction in mobile devices. Finally, we have in mind to share the built CL Android application with the research community.

ACKNOWLEDGMENT

This research is supported by Motorola Mobility, LLC. Also, the authors thank to CNPq and FACEPE (IBPG-1269-1.03/14), both Brazilian agencies, for partial financial support.

REFERENCES

[1] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," in *Proc. MICRO*, 2009, pp. 168–178.
[2] H. B. Jang *et al.*, "Intelligent governor for low-power mobile application processors," in *Proc. ISOCC*, 2013, pp. 206–207.
[3] X. Liu, P. Shenoy, and M. D. Corner, "Chameleon: Application-level power management," *IEEE Transactions on Mobile Computing*, vol. 7, pp. 995–1010, 2008.
[4] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," in *Proc. ACM SIGARCH*, 2009, pp. 46–55.
[5] S. Tarkoma *et al.*, *Smartphone Energy Consumption*. Cambridge University Press, pp. 18-35, 2014.
[6] B. Donohoo *et al.*, "Exploiting spatiotemporal and device contexts for energy-efficient mobile embedded systems," in *Proc. DAC*, 2012, pp. 1274–1279.
[7] A. Carroll and G. Heiser, "Unifying dvfs and offlining in mobile multicores," in *Proc. RTAS*, 2014, pp. 287–296.
[8] D.-R. Chen *et al.*, "A machine learning method for power prediction on the mobile devices," *Journal of Medical Systems - Mobile Systems*, pp. 1–11, 2015.
[9] L. Zhang *et al.*, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc. CODES+ISSS*, 2010, pp. 105–114.
[10] P. Tutor. (2015) Power tutor software. [Online]. Available: http://ziyang.eecs.umich.edu/projects/powertutor/
[11] A. Developer. (2015) Android, the world's most popular mobile platform. [Online]. Available: http://developer.android.com/about/index.html
[12] M. Kuhn and K. Johnson, *Applied Predictive Modeling*. New York: Springer, 2013.
[13] S. Daud *et al.*, "The effects of cpu load & idle state on embedded processor energy usage," in *Proc. ICED*, 2014, pp. 30–35.
[14] K. Sekar, "Power and thermal challenges in mobile devices," in *Proc. MobiCom*, 2013, pp. 363–368.

A Distributed Self-Reconfiguration Algorithm for Cylindrical Lattice-Based Modular Robots

André Naz, Benoît Piranda, Julien Bourgeois
Univ. Bourgogne Franche-Comté
FEMTO-ST Institute, UMR CNRS 6174
25200 Montbéliard, France
Email : {andre.naz, benoit.piranda, julien.bourgeois}@femto-st.fr

Seth Copen Goldstein
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email : seth@cs.cmu.edu

***Abstract*—Modular self-reconfigurable robots are composed of independent connected modules which can self-rearrange their connectivity using processing, communication and motion capabilities, in order to change the overall robot structure. In this paper, we consider rolling cylindrical modules arranged in a two-dimensional vertical hexagonal lattice. We propose a parallel, asynchronous and fully decentralized distributed algorithm to self-reconfigure robots from an initial configuration to a goal one. We evaluate our algorithm on the millimeter-scale cylindrical robots, developed in the Claytronics project, through simulation of large ensembles composed of up to ten thousand modules. We show the effectiveness of our algorithm and study its performance in terms of communications, movements and execution time. Our observations indicate that the number of communications, the number of movements and the execution time of our algorithm is highly predictable. Furthermore, we observe execution times that are linear in the size of the goal shape.**

***Index Terms*—Distributed algorithm, Self-reconfiguration algorithm, Modular robotic, Programmable Matter, Ensembles**

I. Introduction

Modular Self-reconfigurable Robots (MSR) [1] are distributed robotic systems composed of independent connected modules which are able to collaborate and coordinate their activities in order to achieve common goals. Every module has its own computation and communication capabilities, sensors and actuators. MSR have a wide range of potential applications. This work is part of the Claytronics project [2], [3] in which we envision massive-scale MSR, composed of up to millions of modules, to build programmable matter, i.e., matter that can change its physical properties under program control.

The most used algorithm in MSRs is the self-reconfiguration algorithm which causes the modules to move from one configuration (the *initial shape*) to another one (the *goal shape*) (see Figure 1). Self-reconfiguration has several applications. In the context of programmable matter, it enables an MSR to assume different shapes. Self-reconfiguration can also be used to adapt MSR to changes in the environment or to specific tasks. For instance, in [4], the authors use the self-reconfiguration to rearrange modules connectivity in order to reach an optimal network topology.

Self-reconfiguration algorithms pose several challenges. Firstly, planning is challenging as the number of possible unique configurations is huge: $(c \cdot w)^n$ where n is the number of modules, c the number of possible connections per module and w the ways of connecting the modules together [5]. Depending on the physical constraints, modules can often move concurrently which makes the configuration space grow at the rate of $O(m^n)$ with m the number of possible movements and n the number of modules free to move [6]. The exploration space for reconfiguration between two random configurations is therefore exponential in the number of modules which prevents finding a complete optimal planning for all but the simplest configurations. The optimal self-reconfiguration planning for chain-type MSRs is then an NP-complete problem [7], and, to the best of our knowledge, nothing has been proved so far for lattice-based MSR. Secondly, in addition to the path planning problem, the self-reconfiguration process is also challenging as it is a distributed process that requires distributed coordination of mobile autonomous modules connected in time-varying ways. In particular, modules have to coordinate their motions in order to not collide with each other.

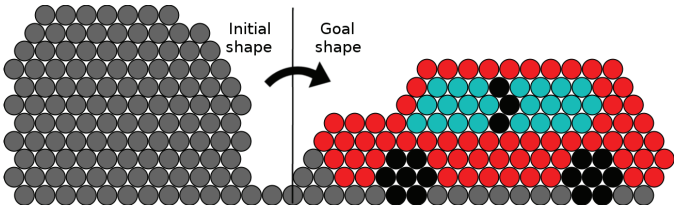


Fig. 1: Example of initial and goal shapes. Self-reconfiguration is the process during which the initial clump of modules on the left self-reconfigures into the car shape on the right.

Self-reconfiguration algorithms are tailored for a specific class of modular robots, with specific motion constraints [8], for example using cubes sliding on the floor, some motions need a cooperation process that complicates motion algorithms [9]. In this paper, we base our model on the millimeter-scale cylindrical robots [10], [11] (see Figure 2), called 2D Catoms, developed in our project. Catoms are the basic unit for Claytronics. 2D Catoms have been partially validated with the realization of a hardware prototype. In this paper, we assume 2D Catoms can communicate together using neighbor-to-neighbor communications and move by rolling around each

978-1-5090-3216-7/16/$31.00

other as long as they respect some motion constraints (see section II).

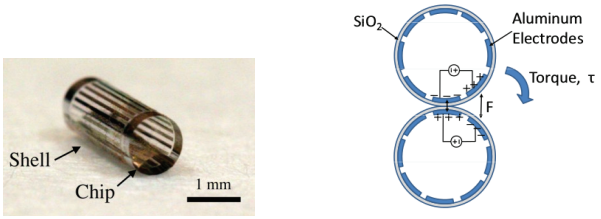


Fig. 2: The 2D Catom. A fabricated prototype (on the left) and the actuation scheme (on the right) [10].

The contribution of this paper is to propose the Cylindrical-Catoms Self-Reconfiguration (C2SR) algorithm which is asynchronous, deterministic, fully decentralized and able to manage almost any kind of initial and goal compact shapes (see section IV). Although our work is focused on the algorithm, we carry out our analysis with respect to hardware constraints based on the 2D Catoms prototype developed in [10], [11]. C2SR is a step toward realizing programmable matter.

We implemented our algorithm in C++ and evaluated it through simulations with our simulator, VisibleSim [12], [13]. We show the effectiveness of C2SR on large-scale ensembles composed of up to ten thousands of modules. We also show the effectiveness of our algorithm and study its performance in terms of communications, movements, and execution time. Our observations indicate that the number of communications, the number of movements and the execution time of our algorithm is predictable. Furthermore, its execution time appears to be linear in the size of the goal shape.

The rest of this paper is organized as follows. In section II, we define the system model and assumptions. Afterwards, we discuss the related work in section III. In section IV, we present the general idea of C2SR and in section V, we describe its implementation. In section VI, experimental results are presented and analyzed. Section VII, concludes this paper and section VIII proposes some directions for future work.

II. System Model and Assumptions

In this paper, we consider the millimeter-scale cylindrical robots [10], [11] (see Figure 2), called 2D Catoms, developed in the Claytronics project. Some of the 2D Catoms functionalities have been validated using this prototype.

A 2D Catom consists of a 6-mm long and 1-mm diameter cylindrical shell. A high voltage CMOS die is attached inside the tube. The chip includes a storage capacitor and a simple logic unit. The tube has electrodes used for power transfer, communications and actuation. The power is spread from a powered floor through the ensemble using neighbor-to-neighbor power transfer.

We assume that 2D Catoms are organized into a horizontal pointy-topped hexagonal lattice where modules have up to six neighbors. Modules can communicate together using neighbor-to-neighbor communications. We assume that modules automatically discover their neighbors using communications after becoming attached. We assume that moving modules cannot communicate with any other module. $\mathcal{N}_{C_i}^N$ denotes the network neighbors of the module C_i. Catoms on the periphery have clockwise (CW) and counter-clockwise (CCW) neighboring Catoms that also belong to the periphery. For instance, in Figure 3, C_9 is C_6's CW peripheral neighbor and C_{10}'s CCW one. C_{11} is both C_{12}'s CW and CCW peripheral neighbor.

$p_{C_i} = (x_{C_i}, y_{C_i})$ denotes the coordinates of the 2D Catom C_i in the horizontal hexagonal lattice. $p_{C_i}.x$ denotes C_i's column in the lattice, while $p_{C_i}.y$ denotes C_i's height. For instance, in Figure 3, $p_{C_2}.y = 0$ and $p_{C_9}.y = 2$. We assume that, at any time, modules know both their coordinates in the lattice and the coordinates of their neighbor through an external algorithm, e.g., [14] or a distributed and incremental version of [15].

Moreover, a 2D Catom can roll CW or CCW around a stationary module. During an atomic move, a module rotates $60°$ going from one cell of the lattice to its adjacent cell. We assume that a 2D Catom has only the capability to lift itself, it cannot carry or push other modules. A module can move if it satisfies the freedom of movement rule (see Rule 1).

Rule 1 (the freedom of movement rule). *Because of possible mismatching issues due to physical constraints, a 2D Catom can only move from/into a cell if this cell is currently unoccupied and no two symetrically opposing cells adjacent to that cell are occupied (see Figure 3). Furthermore, we consider the floor as if it were filled with 2D Catoms. If a 2D Catom, C_i, satisfies the freedom of movement rule, $free(C_i)$ is true, otherwise it is false.*

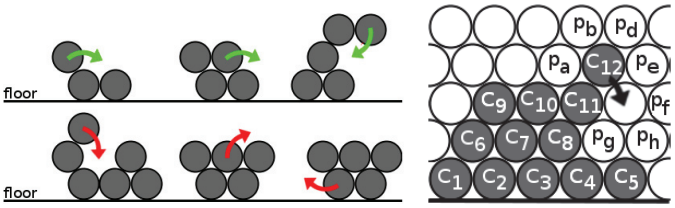


Fig. 3: On the left, motion constraints: examples of feasible (on the top) and infeasible moves (on the bottom). On the right, a labeled system: gray cells are occupied by a module whereas white cells are empty. Some of the empty cells are labeled with their position (e.g., p_a, p_b, etc.).

In the current design, a 2D Catom is able to perform a revolution in 1.67 seconds or 3.35 seconds [11], which corresponds to an average speed of 1.88 $mm \cdot s^{-1}$ or 0.94 $mm \cdot s^{-1}$. We assume that 2D Catoms are not provided with any hardware mechanism to handle collision. Thus, collisions have to be prevented by the self-reconfiguration algorithm, using communications.

We use $\mathcal{N}_p^K$ to denote the set of modules geographically adjacent to position p. A module C_i, moving from p_{C_i} to p'_{C_i}, is somewhere between these two positions, and thus, C_i belongs to the set of geographically adjacent modules of all

the cells adjacent to p_{C_i} or p'_{C_i}. For instance, in the labeled system depicted in Figure 3, module C_{12} is moving and, thus it belongs to $\mathcal{N}^K_{p_a}$, $\mathcal{N}^K_{p_b}$, $\mathcal{N}^K_{p_d}$, $\mathcal{N}^K_{p_{C_{12}}}$, $\mathcal{N}^K_{p_e}$, $\mathcal{N}^K_{p_{C_{11}}}$, $\mathcal{N}^K_{p'_{C_{12}}}$, $\mathcal{N}^K_{p_f}$, $\mathcal{N}^K_{p_g}$ and $\mathcal{N}^K_{p_h}$. Note that in the presence of moving modules, $\mathcal{N}^K_{p_{C_i}}$ may be different from $\mathcal{N}^N_{C_i}$. Also notice that the construction of the $\mathcal{N}^K$ sets is not automatic. 2D Catoms are not equipped with any presence sensor. Maintaining on Catoms the $\mathcal{N}^K$ set of some specific nearby positions, using only communications, is one of the key operations in the implementation of our algorithm.

$\mathcal{I}$ and $\mathcal{G}$ respectively denote the initial and the goal shapes. We assume that every module stores a representation of the shape geometry of $\mathcal{G}$. Our algorithm also assumes some admissibility conditions for $\mathcal{I}$ and $\mathcal{G}$ (see section IV).

In this paper, colors are used for illustration purposes only. The current prototype is not equipped with any mechanism to glow with color. It is possible to do so, but the weight of that color mechanism will probably change the 2D Catom motion speed.

Furthermore, we assume a failure-free environment, i.e., we assume there is no module, communication, move or lattice failure during the algorithm execution.

III. Related Work

Self-reconfiguration and self-assembly have attracted a lot of attention in the last two decades. Algorithms have been proposed for modules of different shapes, with different physical motion constraints and arranged in various ways. In this paper, we consider self-reconfiguration of 2D Catom systems, rolling elements organized in a vertical and two-dimensional hexagonal lattice. Algorithms also differ by their restriction on the initial and goal shapes. Our algorithm can manage almost any kind of initial and goal compact shapes (see section IV). Algorithms also vary in their control properties. In particular, they can be centralized or distributed and synchronous or asynchronous. In this paper, we propose a distributed and asynchronous algorithm.

In [16], the authors propose a synchronous distributed algorithm to perform chain-to-chain self-reconfiguration in a hexagonal lattice. This work was latter extended to allow self-reconfiguration from a chain configuration to an arbitrary shape with some admissibility conditions [17], [18]. These algorithms assume less restrictive motion constraints than the motion constraints we assume for the 2D Catoms. For instance, these algorithms allow the two first motions described as infeasible in Figure 3, starting from the left.

Self-reconfiguration presented in [19], [20] consists in using map-less representation for describing shapes. The benefit lies in a reduced memory footprint, but the number of supported goal shapes is limited. Proposed distributed algorithms manage to construct square shapes with spherical modules arranged in a two-dimensional hexagonal lattice. Due to the fact that initial and goal shapes are fixed, the number of movements can be predicted.

Algorithms to reconfigure an initial clump of modules arranged in a hexagonal lattice to a chain configuration were proposed in [21], [22]. These algorithms do not require message passing and do not use any pre-processing. In these algorithms, modules can both rotate and slide over other modules. Thus, these algorithms assume less restrictive motion constraints than ours.

In [23], the authors propose a distributed shape formation algorithm based on hole motions, for ensembles arranged in a hexagonal lattice. This algorithms can construct various shapes by randomly moving empty spaces within the ensemble. Although a wide variety of shapes can be built, this algorithm requires less restrictive motion constraints than ours, e.g., it allows the two first infeasible motions in Figure 3.

In [24], the authors propose a parallel, decentralized and asynchronous algorithm for the Kilobot swarm system [25] to self-assemble almost any kind of compact two-dimensional shapes. This algorithm has been applied on hardware systems with more than a thousand individual robots per swarm entities. However, these swarm robots have different physical motion constraints. During the self-assembly process, Kilobots may collide with one another. While this is possible with Kilobots, this is not acceptable in our system.

Existing protocols contain interesting ideas but consider different physical motion constraints, different restrictions on the initial and the goal shapes and different control properties. The contribution of this paper is to propose a distributed, fully decentralized, asynchronous and parallel self-reconfiguration algorithm for 2D Catoms that can manage almost any kind of initial and final compact shapes.

IV. C2SR Algorithm at a Glance

In this section, we present the general idea of the Cylindrical-Catoms Self-Reconfiguration (C2SR) algorithm[1] that reconfigures a robot composed of modules from an initial shape $\mathcal{I}$ to a goal one $\mathcal{G}$.

Both shapes have to satisfy some admissibility conditions. We provide some intuitions about them in this paragraph and in Figure 4. A more formal description of the conditions and their demonstration are left for future work. Both shapes are compact, i.e., they do not contain holes, they are homeomorphic to a sphere. Moreover, both shapes are next to each other and intersect in one or more bottom cells. Let the peripheral path be the path formed from the empty cells on the periphery of both shapes, starting from and ending at the second horizontal layer (see Figure 4). This path has to be large enough to allow some modules, which progress along that path in the same direction with an empty space of at least one cell between successive modules, to move without violating our motion constraints and without risking colliding/getting attached with one another (see Figure 4 and Rule 1). Note that this condition implies that, at the upper layers, the horizontal space between the initial and the goal shapes has to be sufficiently large to enable these modules to move between the two shapes. Furthermore, the number of 2D

[1]Some examples of self-reconfiguration with C2SR are available online in video at https://youtu.be/XGnY-oS4Nw0

Catoms in $\mathcal{I}$ has to be greater or at least equal to the number of target positions in $\mathcal{G}$ (i.e., $|\mathcal{I}| \geq |\mathcal{G}|$).

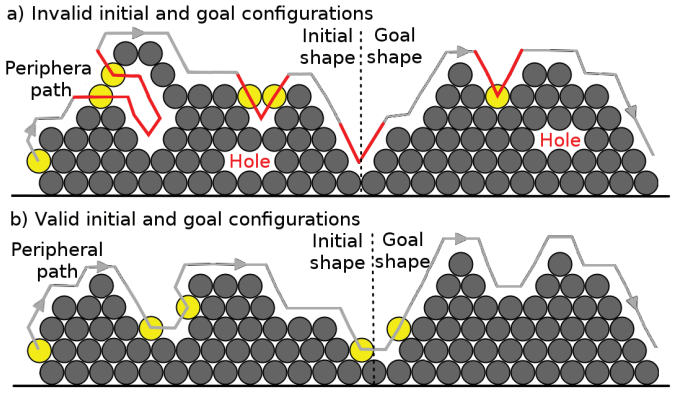


Fig. 4: Invalid (on the top) and valid (on the bottom) initial and goal configurations. Modules in yellow, which are not part of the initial or the goal shapes, progress along the peripheral path in the same direction with an empty space of at least one cell between successive modules. The configurations on the top are not valid for several reasons. First, they do not intersect in at least one cell. Second, they both contain a hole. Third, the peripheral path is not large enough in locations in red. Indeed, modules in yellow could not move without violating our motion constraints and without getting attached with each other.

During the execution of C2SR with shapes individually composed of only continuous horizontal layers, the goal shape is progressively constructed from the bottom layer to the top one by stripping the initial shape, module by module in the reverse order (see Figure 5). Because of physical constraints, at a given instant, only modules on the periphery can move. In order to avoid module collisions and deadlocks, peripheral modules form a stream: modules roll in the same direction d (CW in Figures 1 and 5), and maintain an empty cell between each other using message exchanges. Modules in the stream do not overtake each other.

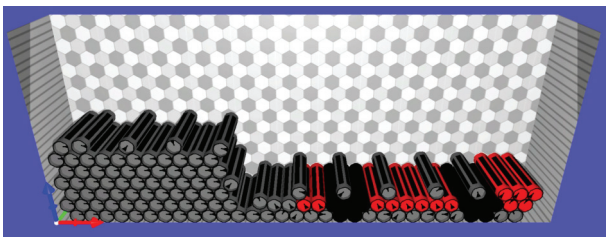


Fig. 5: Screenshot during the self-reconfiguration process with the initial and goal shapes of Figure 1. Modules in the stream progress by rotating CW.

A module locally decides to start taking part in the stream if it satisfies the stream entrance rule (see Rule 2). Intuitively, a free module enters the stream if moving in the direction d consists in: moving around a module on the ground, or descending $\mathcal{I}$, or moving around $\mathcal{G}$, or moving in $\mathcal{G}$ without leaving it and without going up.

Rule 2 (the stream entrance rule). *Let us consider two modules C_i and C_j such that both C_i and C_j are on the periphery and C_j is the next peripheral neighbor of C_i in the direction of rotation, d. p'_{C_i} denotes the position that C_i would occupy after its rotation around C_j. C_i decides to take part in the stream if the following logical condition is satisfied:*

$$\begin{aligned} stream(C_i) :- \; & free(C_i) \\ & \wedge \; (\; (p_{C_i} \notin \mathcal{G} \wedge p_{C_j}.y = 0) \\ & \quad \vee \; (p_{C_i} \notin \mathcal{G} \wedge p'_{C_i}.y \leq p_{C_i}.y) \\ & \quad \vee \; (p_{C_i} \notin \mathcal{G} \wedge p_{C_j} \in \mathcal{G}) \\ & \quad \vee \; (p_{C_i} \in \mathcal{G} \wedge p'_{C_i} \in \mathcal{G} \wedge p'_{C_i}.y \leq p_{C_i}.y) \;) \end{aligned}$$

A module in the stream decides to move if it satisfies the stream progression rule (see Rule 3). More precisely, a module in the stream can move if the set of modules geographically adjacent to its destination cell contains no more than three modules and none of them, except the module itself, belongs to the stream (see Figure 6). This rule requires local interactions with neighbors adjacent to its source and destination positions. These modules are at most two cells away. The admissibility conditions on $\mathcal{I}$ combined with the two rules above, guarantee that these modules are at most five network hops away.

Rule 3 (the stream progression rule). *A module C_i can move from its position p_{C_i} to the position p'_{C_i} if the following condition is satisfied:*

$$\begin{aligned} progression(C_i) :- \; & stream(C_i) \\ & \wedge \; |\mathcal{N}^K_{p'_{C_i}}| \leq 3 \\ & \wedge \; \nexists C_j \in \mathcal{N}^K_{p'_{C_i}} \; | \; C_j \neq C_i \wedge stream(C_j) \end{aligned}$$

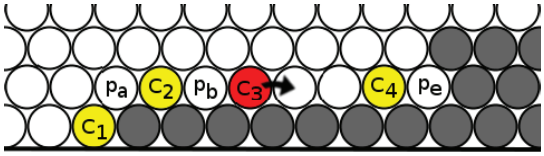


Fig. 6: Stream progression rule: a simple example. Modules should rotate CW. White cells are empty and some of them are labeled with their position in the lattice (e.g., p_a, p_b, etc.). Modules C_1, C_2, C_3 and C_4 are in the stream. C_3 is moving. C_1 cannot move because C_2 is in the stream and $C_2 \in \mathcal{N}^K_{p_a}$. C_2 cannot move because C_3 is in the stream and $C_3 \in \mathcal{N}^K_{p_b}$. C_3 can move to p'_{C_3} because $\mathcal{N}^K_{p'_{C_3}}$ contains only three modules and none of them is in the stream, except C_3. C_4 cannot move because $|\mathcal{N}^K_{p_e}| = 5$.

Rule 3 prevents collisions. The admissibility conditions on $\mathcal{I}$ and $\mathcal{G}$, combined with Rules 2 and 3 prevent deadlock. Note that, because of the stripping order and the construction order, our algorithm also guarantees that at all time the system remains connected.

Each module checks for convergence using Rule 4 at the initialization and after every move. A module has converged if it is initially in a goal position, or if it has reached $\mathcal{G}$ and moving in direction d will cause it to leave $\mathcal{G}$ or to go up.

Rule 4 (the local convergence rule). *Let us consider two modules C_i and C_j such that both C_i and C_j are on the periphery and C_j is the next peripheral neighbor of C_i in the direction of rotation. p'_{C_i} denotes the position that C_i would occupy after its rotation around C_j. C_i has converged if it satisfies the following condition:*

$$\begin{aligned} converged(C_i) :- \ & (p_{C_i} \in \mathcal{I} \wedge p_{C_i} \in \mathcal{G}) \\ & \vee (p_{C_i} \in \mathcal{G} \wedge p'_{C_i} \notin \mathcal{G}) \\ & \vee (p_{C_i} \in \mathcal{G} \wedge p'_{C_i} \in \mathcal{G} \wedge p'_{C_i}.y > p_{C_i}.y) \end{aligned}$$

Applying these rules in a distributed asynchronous system with parallel communications and motions is challenging. It is especially complex to maintain $\mathcal{N}^K$ sets using only communications. A complete implementation that overcomes this challenge is presented in the next section.

V. C2SR Implementation

In this section, we provide a detailed implementation of C2SR[2]. Algorithm 1 shows the input and local variables of C2SR along with its initialization pseudo-code. Every module knows its position in the lattice, the goal shape, $\mathcal{G}$, and the rotation direction, d. Algorithm 2 describes some helper functions used in the description of our implementation of C2SR. Algorithm 3 provides the message handler pseudo-code of C2SR. Algorithm 4 gives the pseudo-code executed by a module after it finishes an atomic move. We assume that interrupts are disabled during message and event handler execution.

```
   Input:
   p_{C_i} // position of C_i
   d ∈ {CW, CCW} // direction of rotation
   G // goal shape
   Local Variables:
   state // state of C_i
   Movings // cells from/into which a neighbor
      module is moving
   Pendings // pending clearance requests
   clearance // clearance for the current move (if
      any)
 1 Initialization of C_i:
 2 Movings ← ∅; Pendings ← ∅; clearance ←⊥;
 3 if p_{C_i} ∈ G then
 4   state ← GOAL;
 5 else if isInStream() then
 6   state ← WAITING;
 7   requestClearance();
 8 else
 9   state ← BLOCKED;
10 end
```

Algorithm 1: C2SR algorithm input, local variables and initialization detailed for any module C_i.

```
 1 Function hasConverged():
     // The local convergence rule (Rule 4)
 2   return converged(C_i);
 3 end

 4 Function areAdjacentCells(p_1, p_2):
 5   return true if cells at positions p_1 and p_2 are adjacent in the
       hexagonal lattice, false otherwise;
 6 end

 7 Function opppositeDirection(d):
     // d ∈ {CW, CCW}
 8   return the opposite direction of d;
 9 end

10 Function isFree():
     // The freedom of movement rule (Rule 1)
11   return free(C_i) considering both N^N_{C_i} and Movings;
12 end

13 Function isInStream():
     // The stream entrance rule (Rule 2)
14   return stream(C_i) considering both N^N_{C_i} and Movings;
15 end

16 Function getNeighbor(dir):
17   return the peripheral neighbor in direction dir (see Section II);
18 end

19 Function getNeighbor(dir, pos):
20   return C_k ∈ N^N_{C_i} such that C_i is connected to C_k on the
       connected interface that immediately follows the interface pointing
       to position pos in direction dir;
21 end

22 Function requestClearance():
23   C_k ← getNeighbor(d);
24   p'_{C_i} ← position after rotation in direction d around C_k;
25   r ← (src ← p_{C_i}, dest ← p'_{C_i}, cnt ← 0);
26   send CLEARANCE_REQUEST(r) to C_k;
27 end

28 Function forwardClearance(c(src, dest), C_j):
29   if areAdjacentCells(c.src, p_{C_i}) then
30     C_k ← getNeighbor(oppositeDirection(d), c.src);
31     if C_k ≠ C_j ∧ areAdjacentCells(c.src, p_{C_k}) then
32       send CLEARANCE(c) to C_k;
33     else
34       Movings ← Movings ∪ {c.src};
35       send CLEARANCE(c) to C_l | p_{C_l} = c.src;
36     end
37   else if areAdjacentCells(c.dest, p_{C_i}) then
38     C_k ← getNeighbor(oppositeDirection(d), c.dest);
39     send CLEARANCE(c) to C_k;
40   end
41 end

42 Function forwardEndOfMove(c(src, dest), C_j):
43   if areAdjacentCells(c.src, p_{C_i}) then
44     C_k ← getNeighbor(oppositeDirection(d), c.src);
45     if C_k ≠ C_j ∧ areAdjacentCells(c.src, p_{C_k}) then
46       send END_OF_MOVE(c) to C_k;
47     end
48   else if areAdjacentCells(c.dest, p_{C_i}) then
49     C_k ← getNeighbor(oppositeDirection(d), c.dest);
50     send END_OF_MOVE(c) to C_k;
51   end
52 end
```

Algorithm 2: C2SR helper functions detailed for any module C_i.

[2]The complete source code is available online at https://github.com/claytronics/visiblesim

```
1  When CLEARANCE_REQUEST(r(src, dest, cnt)) is received by
   C_i from C_j do:
2  if state = WAITING then
3  |  send DELAYED_CLEARANCE(r) to C_j;
4  |  return;
5  end
6  if r.dest ∈ Movings then
7  |  Pendings ← Pendings ∪ {r};
8  |  return;
9  end
10 if state = BLOCKED ∨ state = GOAL then
11 |  if r.cnt = 3 then
12 |  |  send DELAYED_REQUEST(r) to C_j;
13 |  |  return;
14 |  end
15 |  r.cnt ← r.cnt + 1;
16 end
17 C_n ← getNeighbor(d, r.dest);
18 if C_n ≠ C_j ∧ areAdjacentCells(p_{C_n}, r.dest) then
19 |  send CLEARANCE_REQUEST(r) to C_n;
20 else
21 |  c ← (r.src, r.dest);
22 |  Movings ← Movings ∪ {r.dest};
23 |  forwardClearance(c, ⊥);
24 end

25 When CLEARANCE(c(src, dest)) is received by C_i from C_j do:
26 if c.src = p_{C_i} then
27 |  clearance ← c;
28 |  send START_TO_MOVE to C_j;
29 else
30 |  forwardClearance(c, C_j);
31 end

32 When DELAYED_CLEARANCE(r(src, dest, cnt)) is received by
   C_i from C_j do:
33 if r.src ≠ p_{C_i} then
34 |  Pendings ← Pendings ∪ {r};
35 end

36 When START_TO_MOVE is received by C_i from C_j do:
37 send START_TO_MOVE_ACK to C_j;

38 When START_TO_MOVE_ACK is received by C_i from C_j do:
39 state ← MOVING;
40 C_k ← getNeighbor(d);
41 move around C_k in direction d;

42 When END_OF_MOVE(c(src, dest)) is received by C_i from C_j do:
43 Movings ← Movings − {c.src, c.dest};
44 forwardEndOfMove(c, C_j);
45 if isInStream() then
46 |  state ← WAITING;
47 |  requestClearance();
48 else if ∃r ∈ Pendings | r ∈ areAdjacentCells(r.dest, c.src) then
49 |  C_n ← getNeighbor(d, r.dest);
50 |  if areAdjacentCells(r.dest, p_{C_n}) then
51 |  |  send CLEARANCE_REQUEST(r) to C_n;
52 |  else
53 |  |  cl ← (r.src, r.dest);
54 |  |  Movings ← Movings ∪ {cl.dest};
55 |  |  forwardClearance(cl, ⊥);
56 |  end
57 end
```

Algorithm 3: C2SR algorithm message handler detailed for any module C_i.

```
1  When C_i has finished to move do:
2  p_{C_i} ← clearance.dest;
3  send END_OF_MOVE(clearance) to getNeighbor(d);
4  clearance ← perp;
5  if hasConverged() then
6  |  state ← GOAL;
7  else if isInstream() then
8  |  state ← WAITING;
9  |  requestClearance();
10 end
```

Algorithm 4: C2SR algorithm event handler detailed for any module C_i.

In our implementation, modules can have different states: BLOCKED, GOAL, WAITING or MOVING. WAITING and MOVING modules belong to the stream. At the initialization and during the execution, modules locally decide their state using Rules 1, 2 and 4. Modules in the stream move in rotation direction d around their peripheral neighbor in the d direction. Before moving, modules have to ensure that the stream progression rule (Rule 3) is satisfied. WAITING modules send CLEARANCE_REQUEST messages to get the authorization to move. Clearance requests are composed of the module source position and of its destination. These requests travel around the module destination cell. At each hop, modules check if the requested move satisfies the stream progression rule (see Algorithm 3, lines 1-24). If the stream progression rule is not satisfied the clearance request either has to be stored locally (see Algorithm 3, lines 6-9) or to be stored at the previous module using a DELAYED_CLEARANCE message (see Algorithm 3, lines 2-5, 11-14 and 32-35). If the stream progression rule is satisfied, the clearance is granted (see Algorithm 3, lines 20-24). The clearance is then progressively forwarded back to the module that initiated the request (see Algorithm 3, lines 25-31).

To prevent collision, modules maintain a list of neighbor cells from/into which a module is moving. After having moved to a new position, modules send an END_OF_MOVE (EOM for short) message that is progressively forwarded around the cell of their previous position (see Algorithm 4, line 3 and Algorithm 3, lines 42-57). Upon reception, of an EOM message, delayed clearances are potentially re-activated (see Algorithm 3, lines 48-57).

START_TO_MOVE and START_TO_MOVE_ACK messages guarantee that no message is lost when a module decides to actually move (see Algorithm 3, lines 36-41).

Modules never need to communicate with modules farther than two cells away in the lattice, which means that, due to our requirements, modules never need to send messages that have to travel more than five hops. Thus, our algorithm uses only local interactions between modules.

VI. Experimental Evaluation

We implemented C2SR in C++ and evaluated it using VisibleSim [12], a simulator for modular robots. This section presents our experimental results. Through our experiments, we show the effectiveness of C2SR and its efficiency in terms of communications, movements and execution time.

VisibleSim enables one to perform simulations with different and variable motion and communication delays. In our

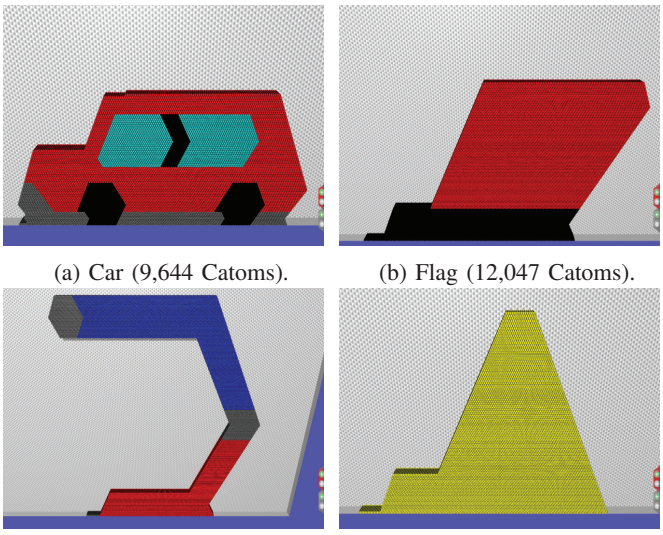
evaluation, we assume that neighboring modules communicate together using 8-N-1 serial communications. Hence, we assume the effective bit-rate is equal to 80% of the link bit-rate. We assume the effective average communication bit-rate between two neighboring modules follows a Gaussian distribution. Moreover, we assume the average motion speed during atomic moves of a 2D Catom also follows a Gaussian distribution. We do not simulate delays due to processing and interruptions because we assume them to be negligible in comparison to communication and motion delays.

Unless explicitly mentioned, we assume the following simulation parameters. We consider the effective average communication bit-rate during message exchanges between two neighboring modules has a distribution centered on 38.9 $kbps$ with a standard-deviation of 389 bps (1% of the mean). Moreover, we assume the average motion speed during atomic moves of a module has a distribution centered on 1.88 $mm \cdot s^{-1}$ with a standard-deviation of 0.0188 $mm \cdot s^{-1}$ (1% of the mean).

We evaluate C2SR on the self-reconfiguration of random clumps of 2D Catoms into four kinds of shapes, namely a car, a flag, a magnet and a pyramid shape (see Figures 1 and 7). For each target shape, we generated different versions of the goal configurations using different scales ranging from a dozens to ten thousands of modules.

A. Effectiveness Evaluation

As shown in Figure 7, C2SR is able to self-reconfigure ensembles composed of more than 10,000 2D Catoms.



(a) Car (9,644 Catoms). (b) Flag (12,047 Catoms).

(c) Magnet (10,220 Catoms). (d) Pyramid (8,033 Catoms).

Fig. 7: Screenshots of VisibleSim at the end of the simulation of C2SR with different kinds of goal shapes composed of about 10,000 2D Catoms.

B. Communication Evaluation

Figure 8 shows the total number of messages sent during the execution of C2SR according to the size of the goal shape. For the shapes we considered, the number of messages seems to depend on the size of the goal configuration and not on the actual shape of the arrangement. Moreover, the standard-deviation is very small, so small, that it is not visible on the figure. Thus, for a goal shape of a given size, C2SR always sends approximately the same number of messages. Furthermore, as shown in Figure 8 by the curve of best fit $y(x) = 20.29x^{1.53}$, this number of messages is highly predictable and increases polynomially with the size of the goal shape.

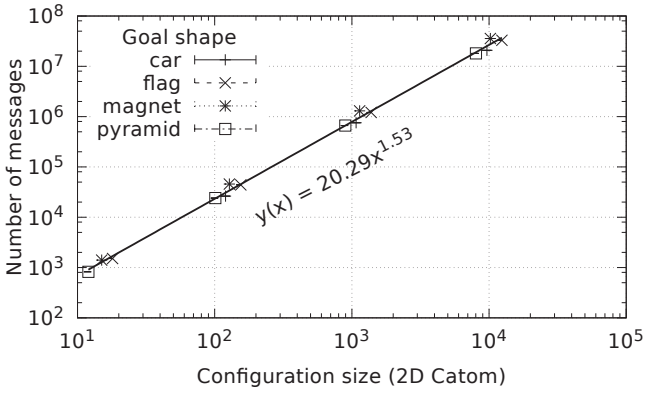


Fig. 8: Average total number of messages (± standard-deviation) versus the size of the system for different goal shapes. For each point, 10 executions were performed.

Figure 9 indicates that a few modules tend to send a lot more messages than the other modules. Intuitively, modules that stay at the boundary between $\mathcal{I}$ and $\mathcal{G}$ are communication hotspots because many modules have to communicate with them before rolling over them in order to reach $\mathcal{G}$ (see Figure 13).

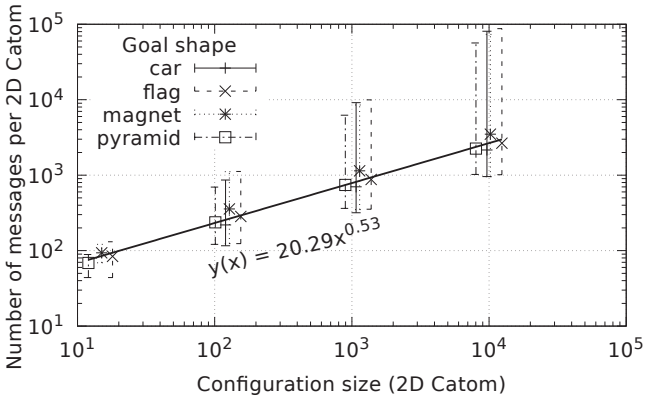


Fig. 9: Average number of messages sent per 2D Catom (± min/max) versus the size of the system for different goal shapes. For each point, 10 executions were performed.

Figure 10 shows the maximum message queue size reached by the modules during the execution of C2SR, taking into account both the incoming and the outgoing messages. The maximum message queue size is constant and equal to two regardless of the shape of the goal configuration and regardless of its size. We recall that messages generated by C2SR have

a small and constant size. As a consequence, the traffic generated by C2SR is well controlled and modules do not require a lot of memory space to store incoming and outgoing messages.

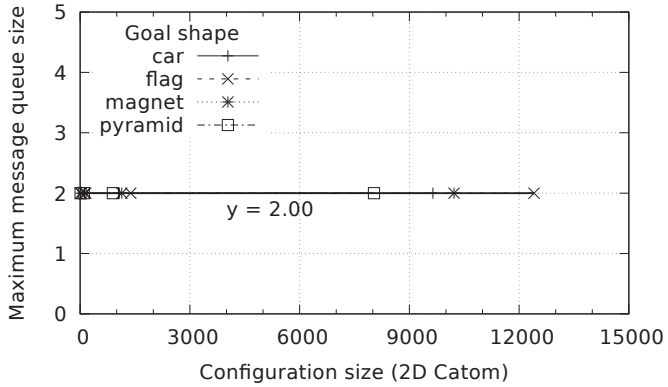


Fig. 10: Maximum reached message queue size (incoming and outgoing messages) versus the size of the system. For each point, 10 executions were performed.

Figure 11 shows the average number of hops traveled by the packets during the execution of C2SR. The average and the maximum number of hops traveled by the packets is small and relatively constant regardless of the shape of the goal configuration and regardless of its size. This confirms that C2SR only involves local interactions, as announced in the previous section.

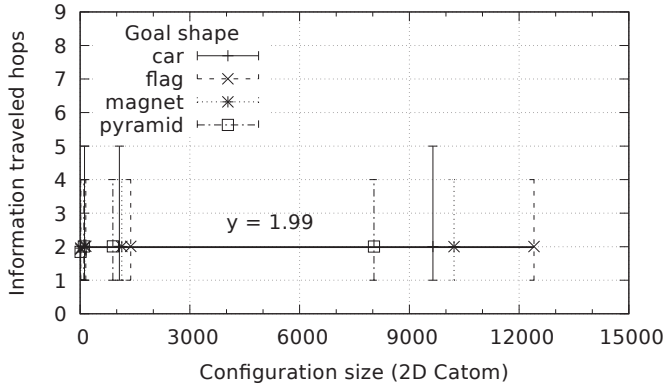


Fig. 11: Average number of hops data have traveled (± min/max) versus the size of the system. For each point, 10 executions were performed.

C. Motion Efficiency

Figure 12 shows the total number of atomic moves performed during the execution of C2SR according to the size of the system for different goal shapes. Note that this figure is really similar to Figure 8. Here again, the number of atomic moves seems to only depend on the size of the goal configuration and not to the actual shape of the arrangement. As shown in Figure 12 by the curve of best fit $y(x) = 2.09x^{1.53}$, the number of atomic moves is highly predictable and increases polynomially with the size of the goal shape. Notice that the number of messages is approximately equal to ten times the number of moves (see Figures 8 and 12). Thus, an atomic move requires in average 10 messages.

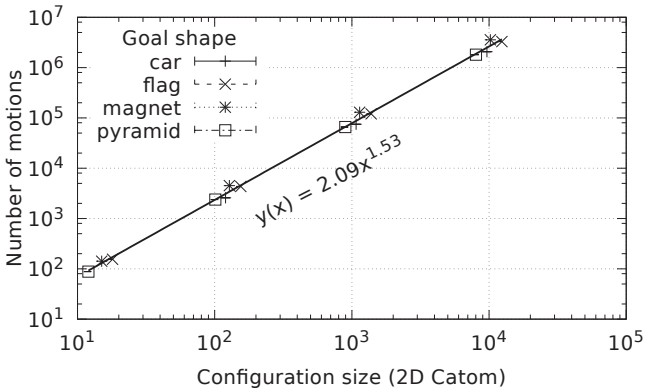


Fig. 12: Average total number of atomic moves (± standard-deviation) versus the size of the system for different goal shapes. For each point, 10 executions were performed.

As shown in Figure 13, many modules can move concurrently during the execution of C2SR. Thus, although the self-reconfiguration process may require many atomic moves, it remains reasonably time efficient as shown in the next subsection.

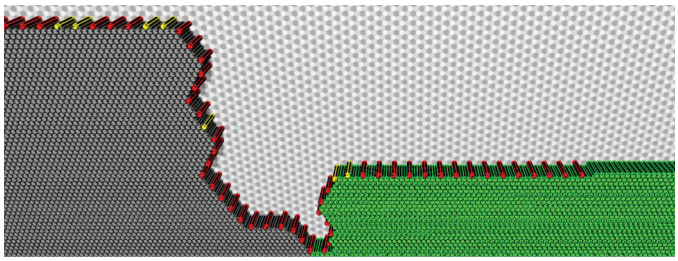


Fig. 13: Screenshot of VisibleSim during a self-reconfiguration process. Modules in the stream progress by rotating CW. Blocked modules are in gray, waiting ones in yellow, moving ones in red and modules that have converged are in green.

D. Execution Time Efficiency

Figure 14 shows the average simulated time of C2SR execution according to the size of the system. For the different goal shapes we considered, this time seems to only depend on the size of the configuration and not to the actual shape of the arrangement. Moreover, the standard-deviation is very small and not visible on the figure. Thus, for goal shape of a given size, C2SR always approximately lasts the same duration. As shown in Figure 14 by the curve of best fit $y(x) = 0.017x + 0.149$, the simulated time is highly predictable and increases linearly with the size of the goal shape. The slope of the line gives the reconfiguration speed: C2SR fills on average $\frac{1}{0.017} \approx 59$ goal cells per minute.

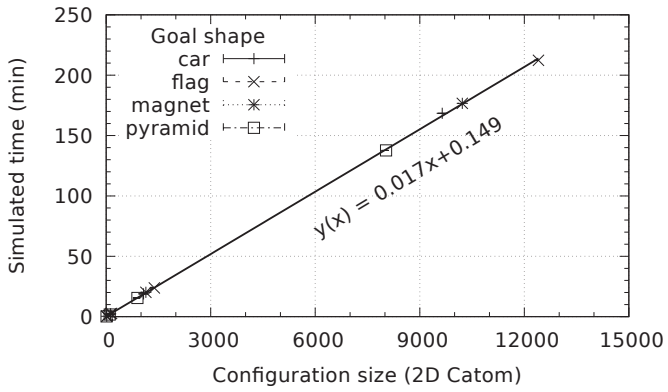


Fig. 14: Average simulated time ($\pm$ standard-deviation) versus the size of the system for different goal shape. For each point, 10 executions were performed.

Figure 15 shows the average simulated time of C2SR execution according to the average communication bit-rate for the two different motion speeds supported by the 2D Catoms. We consider the usual bit-rates of serial communications. We conducted this experiment for the car goal shape composed of 1,073 modules. Until 38.9 $kbps$, the self-reconfiguration process becomes much more faster as the average communication bit-rate increases. Beyond 38.9 $kbps$, the self-reconfiguration speed increases less quickly and tends to stabilize.

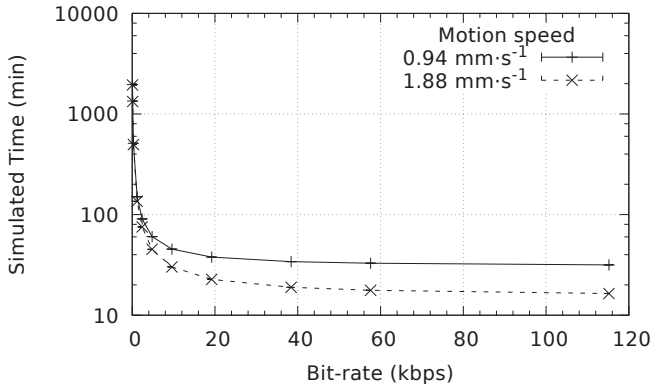


Fig. 15: Average simulated time ($\pm$ standard-deviation) versus the communication bit-rate (random initial configuration to the car of $1,073$ 2D Catoms). For each point, 10 executions were performed.

VII. Conclusion

We have proposed Cylindrical-Catoms Self-Reconfiguration (C2SR), a parallel, asynchronous and fully decentralized distributed algorithm to self-reconfigure lattice-based MSR from an initial shape to a goal one.

The evaluation of C2SR has been conducted with real executions under a simulated physical environment (VisibleSim). These simulations show our algorithm to have nice properties.

The time for reconfiguration is linear in the number of modules and this time is predictable and seems to only depends on the number of modules. The number of messages sent is also predictable such that added with the number of movements, it can give an estimate of the power consumption of the algorithm. Communications are local such that no routing protocol is needed and the message queue of each module is always bounded by two on our simulation. The needed bandwidth is reasonable, as it uses less than 40 $kbps$ on one example without slowing down the reconfiguration process.

VIII. Future Work

In future works, we will demonstrate the correctness of C2SR, i.e., we will prove that the goal configuration can be built if the shape admissibility conditions are satisfied. Moreover, we will study the performance of C2SR on other types of shapes and compare it to existing algorithms. We will also study the distribution of both the number of messages sent per module and the number of atomic moves performed per module. Our observations seem to indicate that our algorithm is highly predictable and that its execution time is linear to the size of the goal shape. A further step would be to prove it. Furthermore, we would like to reduce the memory usage of our algorithm induced by the storage of the goal shape representation. Indeed, hardware modules have limited memory capacity and cannot afford to store the complete representation of large goal shapes [3], [20], [19]. We envision two approaches to address this storage limitation, namely to use a compressed representation of the goal shape and/or to disseminate and share the representation of the goal shape between all modules [3].

Acknowledgments

This work has been funded by the Labex ACTION program (contract ANR-11-LABX-01-01) and ANR/RGC (contracts ANR-12-IS02-0004-01 and 3-ZG1F) and ANR (contract ANR-2011-BS03-005).

References

[1] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[2] S. C. Goldstein and T. C. Mowry, "Claytronics: An instance of programmable matter," in *Wild and Crazy Ideas Session of ASPLOS*, Boston, MA, October 2004.

[3] J. Bourgeois, B. Piranda, A. Naz, H. Lakhlef, N. Boillot, H. Mabed, D. Douthaut, and T. Tucci, "Programmable matter as a cyber-physical conjugation," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. Budapest, Hungary: IEEE, October 2016.

[4] H. Lakhlef, H. Mabed, and J. Bourgeois, "Distributed and dynamic map-less self-reconfiguration for microrobot networks," in *Network Computing and Applications (NCA), 2013 12th IEEE International Symposium on*. IEEE, 2013, pp. 55–60.

[5] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration methods in modular robots," *International Journal for Robotics Research*, vol. 27, no. 3-4, pp. 403–421, March/April 2008.

[6] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991.

[7] F. Hou and W.-M. Shen, "Graph-based optimal reconfiguration planning for self-reconfigurable robots," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 1047 – 1059, 2014.

[8] K. Stoy and H. Kurokawa, "Current topics in classic self-reconfigurable robot research," in *Proceedings of the IROS Workshop on Reconfigurable Modular Robotics: Challenges of Mechatronic and Bio-Chemo-Hybrid Systems*, 2011.

[9] B. Piranda and J. Bourgeois, "A distributed algorithm for reconfiguration of lattice-based modular self-reconfigurable robots," in *PDP 2016, 24th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing*. Heraklion Crete, Greece: IEEE, feb 2016, pp. 1–9.

[10] M. E. Karagozler, S. C. Goldstein, and J. R. Reid, "Stress-driven mems assembly + electrostatic forces = 1mm diameter robot," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS '09)*, October 2009.

[11] M. E. Karagozler, "Design, fabrication and characterization of an autonomous, sub-millimeter scale modular robot," Ph.D. dissertation, Carnegie Mellon University, 2012.

[12] D. Dhoutaut, B. Piranda, and J. Bourgeois, "Efficient simulation of distributed sensing and control environments," in *iThings 2013, IEEE Int. Conf. on Internet of Things*, Beijing, China, Aug. 2013, pp. 452–459.

[13] B. Piranda, "Visiblesim: Your simulator for programmable matter," in *Algorithmic Foundations of Programmable Matter (Dagstuhl Seminar 16271), May 2016*, S. Fekete, A. Richa, K. Römer, and C. Scheideler, Eds.

[14] S. Funiak, P. Pillai, M. P. Ashley-Rollman, J. D. Campbell, and S. C. Goldstein, "Distributed localization of modular robot ensembles," *International Journal of Robotics Research*, vol. 28, no. 8, pp. 946–961, 2009.

[15] M. Dermas, P. Canalda, and F. Spies, "First evaluation of a system of positioning of microrobot with ultra-dense distribution," in *IPIN 2016, International Conference on Indoor Positioning and Indoor Navigation*. IEEE, October 2016.

[16] J. E. Walter, J. L. Welch, and N. M. Amato, "Distributed reconfiguration of metamorphic robot chains," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*. ACM, 2000, pp. 171–180.

[17] J. E. Walter, E. M. Tsai, and N. M. Amato, "Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots," *IEEE transactions on Robotics*, vol. 21, no. 4, pp. 621–631, 2005.

[18] J. Bateau, A. Clark, K. McEachern, E. Schutze, and J. Walter, "Increasing the efficiency of distributed goal-filling algorithms for self-reconfigurable hexagonal metamorphic robots," in *Proceedings of the International Conference on Parallel and Distributed Techniques and Applications*, 2012, pp. 509–515.

[19] H. Lakhlef, J. Bourgeois, H. Mabed, and S. C. Goldstein, "Energy-aware parallel self-reconfiguration for chains microrobot networks," *Journal of Parallel and Distributed Computing*, vol. 75, pp. 67–80, 2015.

[20] H. Lakhlef and J. Bourgeois, "Fast and robust self-organization for micro-electro-mechanical robotic systems," *Computer Networks*, vol. 93, pp. 141–152, 2015.

[21] S. Wong and J. Walter, "Deterministic distributed algorithm for self-reconfiguration of modular robots from arbitrary to straight chain configurations," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 537–543.

[22] S. Wong, S. Zhu, and J. Walter, "Unpacking a cluster of modular robots," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. WorldComp, 2015, p. 103.

[23] M. De Rosa, S. Goldstein, P. Lee, J. Campbell, and P. Pillai, "Scalable shape sculpting via hole motion: Motion planning in lattice-constrained modular robots," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 1462–1468.

[24] M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, no. 6198, pp. 795–799, 2014.

[25] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3293–3298.

Analysis of the Propagation Time of a Rumour in Large-scale Distributed Systems

Yves Mocquard
Université de Rennes 1/IRISA,
yves.mocquard@irisa.fr

Samantha Robert
Université de Nantes/IRISA,
samantha.robert@hotmail.fr

Bruno Sericola
INRIA Rennes - Bretagne Atlantique,
bruno.sericola@inria.fr

Emmanuelle Anceaume
CNRS/IRISA,
emmanuelle.anceaume@irisa.fr

***Abstract*—The context of this work is the well studied dissemination of information in large scale distributed networks through pairwise interactions. This problem, originally called *rumor mongering*, and then *rumor spreading* has mainly been investigated in the synchronous model. This model relies on the assumption that all the nodes of the network act in synchrony, that is, at each round of the protocol, each node is allowed to contact a random neighbor. In this paper, we drop this assumption under the argument that it is not realistic in large scale systems. We thus consider the asynchronous variant, where at time unit, a single node interacts with a randomly chosen neighbor. We perform a thorough study of T_n the total number of interactions needed for all the n nodes of the network to discover the rumor. While most of the existing results involve huge constants that do not allow for comparing different protocols, we prove that in a complete graph of size $n \geq 2$, the probability that $T_n > k$ for all $k \geq 1$ is less than $\left(1 + \frac{2k(n-2)^2}{n}\right)\left(1 - \frac{2}{n}\right)^{(k-1)}$. We also study the behavior of the complementary distribution of T_n at point $c\mathbb{E}(T_n)$ when n tends to infinity for $c \neq 1$. We end our analysis by conjecturing that when n tends to infinity, $T_n > \mathbb{E}(T_n)$ with probability close to 0.4484.**

***Keywords*—rumor spreading, pairwise interactions, Markov chain, analytical performance evaluation.**

I. Introduction

Randomized rumor spreading is an important mechanism that allows the dissemination of information in large and complex networks through pairwise interactions. This mechanism initially proposed by Deemers et al [12] for the update of a database replicated at different sites, has then been adopted in many applications ranging from resource discovery [19], data-aggregation [22], complex distributed applications [8], or virus propagation in computer networks [6], to mention just a few.

A lot of attention has been devoted to the design and study of randomized rumor spreading algorithms. Initially, some rumor is placed on one of the vertices of a given network, and this rumor is propagated to all the vertices of the network through pairwise interactions between vertices. One of the important questions of these protocols is the *spreading time*, that is time it needs for the rumor to be known by all the vertices of the network.

Several models have been considered to answer this question. The most studied one is the synchronous push/pull model, also called the synchronous random phone call model. This model assumes that all the vertices of the network act in synchrony, which allows the algorithms designed in this model to divide time in synchronized rounds. During each synchronized round, each vertex i of the network selects at random one of its neighbor j and either sends to j the rumor if i knows it (push operation) or gets the rumor from j if j knows the rumor (pull operation). In the synchronous model, the spread time of a rumor is defined as the number of synchronous rounds necessary for all the nodes to know the rumor. In one of the first papers dealing with the push operation only, Frieze and Grimmet [16] proved that if the underlying graph is complete, then asymptotically almost surely the number of rounds is $\log_2(n) + \log(n) + o(\log n)$ where n is the number of nodes of the graph. Further results have been established (see for example [21], [7] and the references herein), the most recent ones resulting from the observation that the rumor spreading time is closely related to the conductance of the graph of the network [17], [18]. Investigations have also been done in different topologies of the network [9], [11], [14], [25], in the presence of link or vertices failures (see [13]), and dynamic graphs [10].

All the above studies assume that all vertices of the network act synchronously. In distributed networks, and in particular in large scale distributed systems, such a strict synchronization is unrealistic. Several authors have recently dropped this assumption by considering an asynchronous model. Boyd et al [28] consider that each node has a clock that goes off at the time of a rate 1 Poisson process. Each time the ring of a node goes off, the push or pull operations are triggered according to the knowledge of the node. Acan et al. [1] go a step further by studying rumor spreading time for any graph topology. They show that both the average and guaranteed spreading time are $\Omega(n)$, where n is the number of nodes in the network. Further investigations have been made for different network topologies [26], [15].

This work was partially funded by the French ANR project SocioPlug (ANR-13-INFR-0003), and by the DeSceNt project granted by the Labex CominLabs excellence laboratory (ANR-10-LABX-07-01).

978-1-5090-3216-7/16/$31.00

a) Our contributions : In this paper we consider the population protocol model, which turns out to resemble to the discrete-time version of the asynchronous spreading model. This model provides minimalist assumptions on the computational power of the nodes: nodes are finite-state automata, identically programmed, they have no identity, they do not know how numerous they are, and they progress in their computation through random pairwise interactions. Their objective is to ultimately converge to a state from which the sought property can be derived from any node [5]. In this model, the spreading time is defined as the number of interactions needed for all the nodes of the network to learn the rumor. Angluin et al [3] analyze the spreading time of a rumor by only considering the push operation (which they call the one-way epidemic operation), and show that with high probability, a rumor injected at some node requires $O(n \log n)$ interactions to be spread to all the nodes of the network.

In the present paper we go a step further by considering a more general problem namely, that is all the nodes of the network initially receive an input value, and the objective for each node is to learn the maximal value initially received by any node. Note that the rumor spreading problem is a particular instance of this problem when two input values 1 and 0 are considered respectively representing the knowledge and the absence of knowledge of the rumor. We present a thorough analysis of the number of interactions needed for all the nodes to converge to the correct response. Specifically, we study the expectation, variance and an exact formulation of the distribution of the number of interactions needed to propagate a rumor.

This formulation being hardly usable in practice once n becomes too large, a tight bound is derived. This bound is all the more interesting as usual probabilistic inequalities fail to provide relevant results in this case. Finally, we study the asymptotic behavior of the spreading time when the size of the network tends to infinity.

b) Road map: The remainder of this paper is organized as follows. Section II presents the population protocol model. Section III specifies the problem addressed in this work. Analysis of the spreading time is proposed in Section IV, while we study in Section V its asymptotic behavior. We have simulated our protocol to illustrate our theoretical analysis. Finally, Section VI concludes.

II. Population protocols model

In this section, we present the population protocol model, introduced by Angluin et al. [2]. This model describes the behavior of a collection of nodes that interact pairwise. The following definition is from Angluin et al [4]. A population protocol is characterized by a 6-tuple $(Q, \Sigma, Y, \iota, \omega, f)$, over a complete interaction graph linking the set of n nodes, where Q is a finite set of states, Σ is a finite set of input symbols, Y is a finite set of output symbols, $\iota : \Sigma \to Q$ is the input function that determines the initial state of a node, $\omega : Q \to Y$ is the output function that determines the output symbol of a node, and $f : Q \times Q \to Q \times Q$ is the transition function that describes how two nodes interact and update theirs states. Initially all the nodes start with a initial symbol from Σ, and upon interactions with nodes update their state according to the transition function f. Interactions between nodes are orchestrated by a random scheduler: at each discrete time, any two nodes are randomly chosen to interact with a given distribution. Note that its is assumed that the random scheduler is fair, which means that the interactions distribution is such that any possible interaction cannot be avoided forever. The notion of time in population protocols refers to as the successive steps at which interactions occur, while the parallel time refers to as the successive number of steps each node executes [5]. Nodes do not maintain nor use identifiers (nodes are anonymous and cannot determine whether any two interactions have occurred with the same agents or not). However, for ease of presentation the nodes are numbered $1, 2, \ldots, n$. We denote by $C_t^{(i)}$ the state of node i at time t. The stochastic process $C = \{C_t,\ t \geq 0\}$, where $C_t = (C_t^{(1)}, \ldots, C_t^{(n)})$, represents the evolution of the population protocol. The state space of C is thus Q^n and a state of this process is also called a protocol configuration.

III. Spreading the maximum

We consider in this section the following problem. Each site has initially an integer value. At each discrete instant of time, two distinct nodes are successively chosen and they change their value with the maximum value of each node. More precisely, for all nodes a and b, with $a \neq b$, we consider the function f given by

$$f(a, b) = (\max\{a, b\}, \max\{a, b\}) .$$

We want to evaluate the time needed so that all the nodes get the same value.

Let $C = \{C_t,\ t \geq 0\}$ be a discrete-time stochastic process with state space $S = \mathbb{N}^n$. For every $t \geq 0$, the state at time t of the process is denoted by $C_t = (C_t^{(1)}, \ldots, C_t^{(n)})$, where $C_t^{(i)}$ is the integer value of node i at time t. At each instant t, two distinct indexes i and j are successively chosen among the set of nodes $1, \ldots, n$ randomly. We denote by X_t the random variable representing this choice and we suppose that this choice is uniform, i.e we suppose that

$$\mathbb{P}\{X_t = (i, j)\} = \frac{1}{n(n-1)} 1_{\{i \neq j\}}.$$

Once the couple (i, j) is chosen at time t, the process reaches state C_{t+1}, at time $t + 1$, given by

$$\begin{aligned} C_{t+1}^{(i)} = C_{t+1}^{(j)} &= \max\{C_t^{(i)}, C_t^{(j)}\} \\ \text{and } C_{t+1}^{(m)} &= C_t^{(m)} \text{ for } i \neq j. \end{aligned}$$

We denote by M the maximum initial value among all the nodes, i.e. $M = \max\{C_0^{(1)}, \ldots, C_0^{(n)}\}$. It is easily checked that for all $t \geq 0$, we have $M = \max\{C_t^{(1)}, \ldots, C_t^{(n)}\}$.

We consider the random variable T_n defined by

$$T_n = \inf\{t \geq 0 \mid C_t^{(i)} = M, \text{ for every } 1, \ldots, n\}.$$

The random variable T_n represents the number of interactions needed for all the nodes in the network to know the maximal value M.

We introduce the discrete-time stochastic process $Y = \{Y_t,\ t \geq 0\}$ with state space $\{1, \ldots, n\}$ defined, for all $t \geq 0$, by

$$Y_t = \left| \left\{ i \mid C_t^{(i)} = M \right\} \right|.$$

The random variable Y_t represents the number of nodes knowing the maximum value M at time t. The stochastic process Y is then a homogeneous Markov chain with transition probability matrix A. The non zero transition probabilities are given, for $i, j = 1, \ldots, n$, by

$$\begin{cases} A_{i,i} &= 1 - \dfrac{2i(n-i)}{n(n-1)}, \\ \\ A_{i,i+1} &= \dfrac{2i(n-i)}{n(n-1)}, \text{ for } i \neq n. \end{cases}$$

Indeed, when $Y_t = i$, in order to get $Y_{t+1} = i+1$, either the first node must be chosen among the ones with the maximum value (probability i/n) and the second agent must be chosen among the ones with the non maximum value (probability $(n-i)/(n-1)$) or the first node must be chosen among the ones with the non maximum value (probability $(n-i)/n$) and the second node must be chosen among the ones with the non maximum value (probability $i/(n-1)$).

The states $1, \ldots, n-1$ of Y are transient and state n is absorbing. The random variable T_n can then be written as

$$T_n = \inf\{t \geq 0 \mid Y_t = n\}.$$

It is well-known, see for instance [27], that the distribution of T_n is given, for every $k \geq 0$, by

$$\mathbb{P}\{T_n > k\} = \alpha Q^k \mathbb{1}, \tag{1}$$

where α is the row vector containing the initial probabilities of states $1, \ldots, n-1$, that is $\alpha_i = \mathbb{P}\{Y_0 = i\}$, Q is the sub-matrix obtained from A by deleting the row and the column corresponding to absorbing state n and $\mathbb{1}$ is the column vector of dimension $n-1$ with all its entries equal to 1.

For $i = 0, \ldots, n$, we introduce the notation

$$p_i = \frac{2i(n-i)}{n(n-1)}$$

and we denote by H_k the harmonic series defined by $H_0 = 0$ and $H_k = \sum_{\ell=1}^{k} 1/\ell$, for $k \geq 1$. Note that, for every $i = 0, \ldots, n$, we have $p_i = p_{n-i}$.

If we denote by S_i, for $i = 1, \ldots, n-1$, the total time spent by the Markov chain Y in state i, then conditionally on the event $Y_0 = i$, S_ℓ has a geometric distribution with parameter p_ℓ, for $\ell = i, \ldots, n-1$ and in this case, we have $T_n = S_i + \cdots + S_{n-1}$. It follows that

$$\mathbb{P}\{T_n > k \mid Y_0 = i\} = \mathbb{P}\{S_i + \cdots + S_{n-1} > k\},$$

which means that $\mathbb{P}\{T_n > k \mid Y_0 = i\}$ is decreasing with i and in particular that

$$\mathbb{P}\{T_n > k \mid Y_0 = i\} \leq \mathbb{P}\{T_n > k \mid Y_0 = 1\}. \tag{2}$$

IV. Analysis of the spreading time

In the following we study the expectation and the variance of T_n, the number of interactions needed for all the nodes in the network to know the maximal value M. We then provide an explicit expression of the distribution of T_n, and then a bound and an equivalent for the explicit distribution of T_n.

A. Expectation and variance of T_n

The mean time $\mathbb{E}(T_n)$ needed so that all the nodes get the same value is then given by

$$\mathbb{E}(T_n) = \alpha(I - Q)^{-1}\mathbb{1}, \tag{3}$$

where I is the identity matrix. This expectation can also be written as

$$\mathbb{E}(T_n) = \sum_{i=1}^{n-1} \alpha_i \mathbb{E}(T_n \mid Y_0 = i).$$

This conditional expectations are given by the following theorem.

Theorem *1:* For every $n \geq 1$ and $i = 1, \ldots, n$, we have

$$\mathbb{E}(T_n \mid Y_0 = i) = \frac{(n-1)(H_{n-1} + H_{n-i} - H_{i-1})}{2}.$$

Proof: If $Y_0 = n$, which means that all the nodes start with same values, then we have $T_n = 0$ and so $\mathbb{E}(T_n \mid Y_0 = n) = 0$. For $i = 1, \ldots, n-1$ we have

$$\begin{aligned} \mathbb{E}(T_n \mid Y_0 = i) &= \sum_{\ell=i}^{n-1} \mathbb{E}(S_\ell) \\ &= \sum_{\ell=i}^{n-1} \frac{1}{p_\ell} \\ &= \frac{n(n-1)}{2} \sum_{\ell=i}^{n-1} \frac{1}{\ell(n-\ell)} \\ &= \frac{n-1}{2} \sum_{\ell=i}^{n-1} \left(\frac{1}{\ell} + \frac{1}{n-\ell} \right) \\ &= \frac{(n-1)(H_{n-1} + H_{n-i} - H_{i-1})}{2}, \end{aligned}$$

which completes the proof. ∎

In particular, when the maximum value is initially unique, i.e. when $Y_0 = 1$ with probability 1, we have $\alpha_1 = 1$ and thus

$$\mathbb{E}(T_n) = \mathbb{E}(T_n \mid Y_0 = 1) = (n-1)H_{n-1} \underset{n \longrightarrow \infty}{\sim} n \ln(n).$$

More generally, from Relation (2), we have

$$\mathbb{E}(T_n) \leq \mathbb{E}(T_n \mid Y_0 = 1) = (n-1)H_{n-1} \underset{n \longrightarrow \infty}{\sim} n \ln(n).$$

The variance of T_n is obtained similarly.

Theorem *2:* For every $n \geq 1$ and $i = 1, \ldots, n$, we have

$$\begin{aligned} \mathrm{Var}(T_n \mid Y_0 = i) = \frac{(n-1)^2}{4} &\left(\sum_{\ell=i}^{n-1} \frac{1}{\ell^2} + \sum_{\ell=1}^{n-i} \frac{1}{\ell^2} \right) \\ &- \frac{\mathbb{E}(T_n \mid Y_0 = i)}{n}. \end{aligned}$$

Proof: If $Y_0 = n$, which means that all the nodes start with the same values, then we have $T_n = 0$ and thus $\mathrm{Var}(T_n \mid Y_0 = n) = 0$. For $i = 1, \ldots, n-1$ we have, using the independence of the S_ℓ,

$$\begin{aligned}
\mathrm{Var}(T_n \mid Y_0 = i) &= \sum_{\ell=i}^{n-1} \mathrm{Var}(S_\ell) = \sum_{\ell=i}^{n-1} \frac{1-p_\ell}{p_\ell^2} \\
&= \sum_{\ell=i}^{n-1} \frac{1}{p_\ell^2} - \sum_{\ell=i}^{n-1} \frac{1}{p_\ell} \\
&= \frac{n^2(n-1)^2}{4} \sum_{\ell=i}^{n-1} \frac{1}{\ell^2(n-\ell)^2} - \frac{n(n-1)}{2} \sum_{\ell=i}^{n-1} \frac{1}{\ell(n-\ell)} \\
&= \frac{(n-1)^2}{4} \sum_{\ell=i}^{n-1} \left(\frac{1}{\ell} + \frac{1}{n-\ell} \right)^2 - \frac{n(n-1)}{2} \sum_{\ell=i}^{n-1} \frac{1}{\ell(n-\ell)} \\
&= \frac{(n-1)^2}{4} \sum_{\ell=i}^{n-1} \left(\frac{1}{\ell^2} + \frac{1}{(n-\ell)^2} \right) - \frac{n-1}{2} \sum_{\ell=i}^{n-1} \frac{1}{\ell(n-\ell)} \\
&= \frac{(n-1)^2}{4} \sum_{\ell=i}^{n-1} \left(\frac{1}{\ell^2} + \frac{1}{(n-\ell)^2} \right) - \frac{\mathbb{E}(T_n \mid Y_0 = i)}{n} \\
&= \frac{(n-1)^2}{4} \left(\sum_{\ell=i}^{n-1} \frac{1}{\ell^2} + \sum_{\ell=1}^{n-i} \frac{1}{\ell^2} \right) - \frac{\mathbb{E}(T_n \mid Y_0 = i)}{n},
\end{aligned}$$

which completes the proof. ∎

In particular, when the maximum value is initially unique, i.e. when $Y_0 = 1$ with probability 1, we have $\alpha_1 = 1$ and thus

$$\begin{aligned}
\mathrm{Var}(T_n) &= \mathrm{Var}(T_n \mid Y_0 = 1) \\
&= \frac{(n-1)^2}{2} \sum_{\ell=1}^{n-1} \frac{1}{\ell^2} - \frac{n-1}{n} H_{n-1} \underset{n \longrightarrow \infty}{\sim} \frac{\pi^2 n^2}{12}.
\end{aligned}$$

More generally, from Theorem 2, we have

$$\begin{aligned}
\mathrm{Var}(T_n \mid Y_0 = i) &\leq \frac{(n-1)^2}{4} \left(\sum_{\ell=i}^{n-1} \frac{1}{\ell^2} + \sum_{\ell=1}^{n-i} \frac{1}{\ell^2} \right) \\
&\leq \frac{(n-1)^2}{2} \sum_{\ell=1}^{n-1} \frac{1}{\ell^2} \leq \frac{\pi^2 n^2}{12}.
\end{aligned}$$

It follows that

$$\mathrm{Var}(T_n) = \sum_{i=1}^{n-1} \alpha_i \mathrm{Var}(T_n \mid Y_0 = i) \leq \frac{\pi^2 n^2}{12}.$$

B. Explicit expression of the distribution of T_n

The distribution of T_n, for $n \geq 2$, which is given by Relation (1) can be easily computed as follows. Let $V(k) = (V_1(k), \ldots, V_{n-1}(k))$ be the column vector defined by $V_i(k) = \mathbb{P}\{T_n > k \mid Y_0 = i\}$. According to Relation (1), we have $V(k) = Q^k \mathbb{1}$. Since $V(0) = \mathbb{1}$, writing $V(k) = QV(k-1)$ for $k \geq 1$, we get for $i = 1, \ldots, n-2$,

$$\begin{cases}
V_i(k) = (1-p_i)\, V_i(k-1) + p_i V_{i+1}(k-1), \\
V_{n-1}(k) = (1-p_{n-1})\, V_{n-1}(k-1).
\end{cases} \tag{4}$$

Recall that we have $p_i = 2i(n-i)/(n(n-1))$. This recursion can be easily computed since we have, for $k \geq 0$,

$$V_{n-1}(k) = (1-p_{n-1})^k = \left(1 - \frac{2}{n}\right)^k. \tag{5}$$

In the next theorem, we derive from recursion (4) an explicit expression of the distribution of T_n.

Theorem *3:* For every $n \geq 1$, $k \geq 0$ and $i = 1, \ldots, n-1$, we have

$$\begin{aligned}
&\mathbb{P}\{T_n > k \mid Y_0 = n-i\} \\
&\qquad = \sum_{j=1}^{\lfloor n/2 \rfloor} \left(c_{i,j}(1-p_j) + k d_{i,j} \right) (1-p_j)^{k-1},
\end{aligned}$$

where the coefficients $c_{i,j}$ and $d_{i,j}$, which do not depend on k, are given, for $j = 1, \ldots, n-1$, by

$$c_{1,j} = 1_{\{j=1\}} \text{ and } d_{1,j} = 0$$

and for $i \in \{2, \ldots, n-1\}$ by

$$\begin{aligned}
c_{i,j} &= \frac{p_i c_{i-1,j}}{p_i - p_j} - \frac{p_i d_{i-1,j}}{(p_i - p_j)^2} && \text{for } i \neq j, n-j, \\
d_{i,j} &= \frac{p_i d_{i-1,j}}{p_i - p_j} && \text{for } i \neq j, n-j, \\
c_{i,i} &= 1 - \sum_{j=1, j\neq i}^{n/2} c_{i,j} && \text{for } i \leq n/2, \\
c_{i,n-i} &= 1 - \sum_{j=1, j\neq n-i}^{n/2} c_{i,j} && \text{for } i > n/2, \\
d_{i,i} &= p_i c_{i-1,i} && \text{for } i \leq n/2, \\
d_{i,n-i} &= p_i c_{i-1,n-i} && \text{for } i > n/2.
\end{aligned}$$

Proof: See [24] ∎

C. Bounds of the distribution of T_n

The exact expression of the distribution of T_n presented earlier is hardly usable in practice, and computation using formula (4) may take a long time for large values of n. To overcome this problem, we propose in this section a bound and an equivalent for the quantity $\mathbb{P}\{T_n > k \mid Y_0 = i\}$ derived from the recursive formula (4).

Theorem *4:* For all $n \geq 2$ and $k \geq 1$ we have

$$\mathbb{P}\{T_n > k \mid Y_0 = 1\} \leq \left(1 + \frac{2k(n-2)^2}{n}\right) \left(1 - \frac{2}{n}\right)^{k-1},$$

$$\mathbb{P}\{T_n > k \mid Y_0 = 1\} \underset{k \longrightarrow \infty}{\sim} \left(1 + \frac{2k(n-2)^2}{n}\right) \left(1 - \frac{2}{n}\right)^{k-1}$$

and for $i = 2, \ldots, n-1$ and $k \geq 0$,

$$\mathbb{P}\{T_n > k \mid Y_0 = i\} \leq \frac{(n-i)(n-2)}{i-1} \left(1 - \frac{2}{n}\right)^k,$$

$$\mathbb{P}\{T_n > k \mid Y_0 = i\} \underset{k \longrightarrow \infty}{\sim} \frac{(n-i)(n-2)}{i-1} \left(1 - \frac{2}{n}\right)^k.$$

Moreover, we have

$$\mathbb{P}\{T_n > k\} \leq \mathbb{P}\{T_n > k \mid Y_0 = 1\}.$$

Proof: The result is trivial for $n = 2$ since in this case we have $T_2 = 1$. We thus suppose that $n \geq 3$. Note that by definition of p_i we have $p_i = p_{n-i}$. Consider the sequence b_i defined for $i = 1, \ldots, n-2$, by

$$b_1 = 1 \text{ and } b_i = \frac{p_i b_{i-1}}{p_i - p_1}, \text{ for } i = 2, \ldots, n-2.$$

Observing that

$$b_i = \frac{i(n-i)b_{i-1}}{(i-1)(n-i-1)},$$

it is easily checked by recurrence that for $i = 1, \ldots, n-2$, we have

$$b_i = \frac{i(n-2)}{n-i-1}.$$

We show now by recurrence that for all $i = 1, \ldots, n-2$, we have

$$V_{n-i}(k) \leq b_i \left(1-p_1\right)^k, \text{ for all } k \geq 0$$
$$\text{and } V_{n-i}(k) \underset{k \longrightarrow \infty}{\sim} b_i \left(1-p_1\right)^k.$$

Both results are true for $i = 1$ since $V_{n-1}(k) = (1-p_{n-1})^k = (1-p_1)^k$. Suppose now that these results are true for a fixed integer i with $1 \leq i \leq n-3$. From Relations (4), we have

$$\begin{aligned} &V_{n-i-1}(k) \\ &\quad = (1-p_{n-i-1})V_{n-i-1}(k-1) + p_{n-i-1}V_{n-i}(k-1) \\ &\quad = (1-p_{i+1})V_{n-i-1}(k-1) + p_{i+1}V_{n-i}(k-1). \end{aligned}$$

Using the recurrence hypothesis, we obtain, for what concerns the inequality,

$$V_{n-i-1}(k) \leq (1-p_{i+1})V_{n-i-1}(k-1) + p_{i+1}b_i \left(1-p_1\right)^{k-1}.$$

Expanding this inequality and using the fact that $V_{n-i-1}(0) = 1$, this leads to

$$\begin{aligned} &V_{n-i-1}(k) \\ &\quad \leq (1-p_{i+1})^k + p_{i+1}b_i \sum_{j=0}^{k-1} (1-p_{i+1})^j (1-p_1)^{k-1-j} \\ &\quad = (1-p_{i+1})^k + p_{i+1}b_i \frac{(1-p_1)^k - (1-p_{i+1})^k}{p_{i+1} - p_1} \\ &\quad = (1-p_{i+1})^k + b_{i+1}\left((1-p_1)^k - (1-p_{i+1})^k\right) \\ &\quad = (1-b_{i+1})\left(1-p_{i+1}\right)^k + b_{i+1}(1-p_1)^k. \end{aligned}$$

Since $b_{i+1} \geq 1$, we get

$$V_{n-i-1}(k) \leq b_{i+1}(1-p_1)^k$$

In the same way, using a similar calculus, we obtain

$$V_{n-i-1}(k) \underset{k \longrightarrow \infty}{\sim} (1-b_{i+1})\left(1-p_{i+1}\right)^k + b_{i+1}(1-p_1)^k.$$

Since $p_{i+1} > p_1$, we also get

$$V_{n-i-1}(k) \underset{k \longrightarrow \infty}{\sim} b_{i+1}(1-p_1)^k.$$

We thus have shown that for all $i = 1, \ldots, n-2$, we have

$$V_{n-i}(k) \leq b_i \left(1-p_1\right)^k, \text{ for all } k \geq 0$$
$$\text{and } V_{n-i}(k) \underset{k \longrightarrow \infty}{\sim} b_i \left(1-p_1\right)^k.$$

In particular, for $i = n-2$ we obtain

$$V_2(k) \leq b_{n-2} \left(1-p_1\right)^k, \text{ for all } k \geq 0$$
$$\text{and } V_2(k) \underset{k \longrightarrow \infty}{\sim} b_{n-2} \left(1-p_1\right)^k.$$

Consider now the term $V_1(k)$. From Relations (4) and using the previous inequality, we have

$$\begin{aligned} V_1(k) &= (1-p_1)V_1(k-1) + p_1 V_2(k-1) \\ &\leq (1-p_1)V_1(k-1) + p_1 b_{n-2} \left(1-p_1\right)^{k-1}. \end{aligned}$$

Expanding this inequality and using the fact that $V_1(0) = 1$, this leads to

$$\begin{aligned} V_1(k) &\leq (1-p_1)^k + p_1 b_{n-2} \sum_{j=0}^{k-1} (1-p_1)^j (1-p_1)^{k-1-j} \\ &= (1-p_1)^k + p_1 b_{n-2} k (1-p_1)^{k-1} \\ &= \left(1 - p_1 + k p_1 b_{n-2}\right)\left(1-p_1\right)^{k-1} \\ &\leq \left(1 + k p_1 b_{n-2}\right)\left(1-p_1\right)^{k-1}, \end{aligned}$$

which gives

$$V_1(k) \leq \left(1 + \frac{2k(n-2)^2}{n}\right)\left(1 - \frac{2}{n}\right)^{k-1}.$$

In the same way, using a similar calculus, we obtain

$$V_1(k) \underset{k \longrightarrow \infty}{\sim} \left(1 + \frac{2k(n-2)^2}{n}\right)\left(1 - \frac{2}{n}\right)^{k-1}.$$

Finally, since $\mathbb{P}\{T_n > k \mid Y_0 = i\}$ is decreasing with i, we have

$$\begin{aligned} \mathbb{P}\{T_n > k\} &= \sum_{i=1}^{n-1} \mathbb{P}\{T_n > k \mid Y_0 = i\}\mathbb{P}\{Y_0 = i\} \\ &\leq \mathbb{P}\{T_n > k \mid Y_0 = 1\}, \end{aligned}$$

which completes the proof. ■

The bound established in Theorem 4 is all the more interesting as usual probabilistic inequalities fail to provide relevant results in this particular case. For example, Markov inequality leads for all real number $c \geq 1$ to

$$\mathbb{P}\{T_n \geq c\mathbb{E}(T_n)\} \leq \frac{1}{c},$$

and Bienaym-Tchebychev inequality leads for all real number $x > 0$ to

$$\mathbb{P}\{|T_n - \mathbb{E}(T_n)| \geq x\} \leq \frac{\pi^2 n^2}{12x^2}.$$

The author of [20] provides a bound, based on Chernoff inequality, for the tail probabilities of the sum of independent, but not necessarily identically distributed, geometric random variables. In the particular case of our protocol computing the maximum, this leads to the following result.

Theorem *5:* For all $n \geq 3$ and for all real number $c \geq 1$, we have

$$\mathbb{P}(T_n > c\mathbb{E}(T_n)) \leq \frac{1}{c}\left(1-\frac{2}{n}\right)^{(c-1-\ln c)(n-1)H_{n-1}}.$$

The right-hand side term is equal to 1 when $c = 1$.

Proof: We have already shown that

$$\mathbb{P}(T_n > c\mathbb{E}(T_n)) \leq \mathbb{P}(T_n > c\mathbb{E}(T_n) \mid Y_0 = 1\}.$$

The upper bound is then an application of Theorem 2.3 of [20], and it is clearly equal to 1 when $c = 1$. ■

Applying Theorem 4 at point $k = \lfloor c\mathbb{E}(T_n)\rfloor$, we obtain

$$\begin{aligned}\mathbb{P}(T_n > c\mathbb{E}(T_n)) &\leq \left(1+\frac{2\lfloor c\mathbb{E}(T_n)\rfloor(n-2)^2}{n}\right)\\ &\quad\times\left(1-\frac{2}{n}\right)^{\lfloor c\mathbb{E}(T_n)\rfloor-1}\\ &\leq \left(1+\frac{2c\mathbb{E}(T_n)(n-2)^2}{n}\right)\\ &\quad\times\left(1-\frac{2}{n}\right)^{c\mathbb{E}(T_n)-2}.\end{aligned}$$

From now on we denote this bound by $f(c,n)$ and in the same way, we denote by $g(c,n)$ the bound of $\mathbb{P}(T_n > c\mathbb{E}(T_n))$ derived from Theorem 5. We then have, for $n \geq 3$ and $c \geq 1$,

$$\begin{aligned}f(c,n) &= \left(1+\frac{2c(n-1)H_{n-1}(n-2)^2}{n}\right)\\ &\quad\times\left(1-\frac{2}{n}\right)^{c(n-1)H_{n-1}-2}\\ g(c,n) &= \frac{1}{c}\left(1-\frac{2}{n}\right)^{(c-1-\ln(c))(n-1)H_{n-1}}.\end{aligned}$$

We also introduce the notation

$$e(c,n) = \mathbb{P}(T_n > c\mathbb{E}(T_n)).$$

Theorem *6:* For every $n \geq 3$, there exists a unique $c^* \geq 1$ such that $f(c^*,n) = g(c^*,n)$ and we have

$$\begin{cases} f(c,n) > g(c,n) & \text{for all } 1 \leq c < c^* \\ f(c,n) < g(c,n) & \text{for all } c > c^*. \end{cases} \tag{6}$$

Furthermore,

$$\lim_{c\longrightarrow\infty}\frac{f(c,n)}{g(c,n)} = 0.$$

Proof: See [24]

■

The graphs on Figures 1, 2 and 3 illustrate the behavior of the bounds $f(c,n)$ and $g(c,n)$, depending on c and for different values of n, compared to the real distribution of T_n at point $c\mathbb{E}(T_n)$, i.e. to $e(c,n) = \mathbb{P}\{T_n > \mathbb{E}(T_n)\}$. The bound $f(c,n)$ that we provided in Theorem 4 clearly shows better accuracy than the Chernoff bound $g(c,n)$ provided in [20] above the threshold c^* introduced in Theorem 6. Furthermore, this threshold seems to decrease to 1 as n tends to infinity, as can be seen on Figure 4.

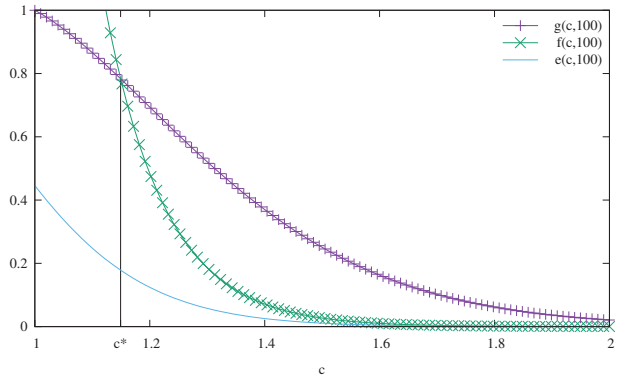


Fig. 1. Bounds $f(c,n)$ and $g(c,n)$ beside the real value of $\mathbb{P}(T_n > c\mathbb{E}(T_n)) = e(c,n)$ for $n = 100$, as functions of c. In this case, we have $c^* = 1.14641$.

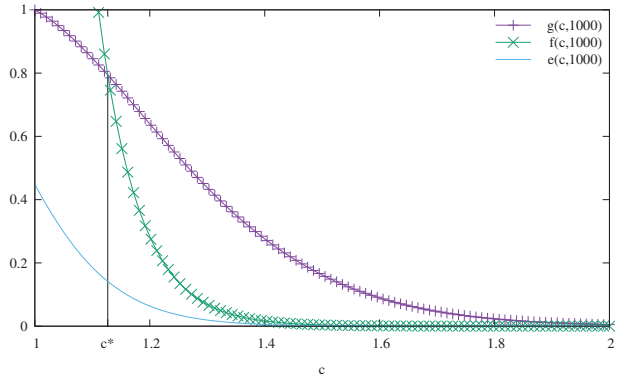


Fig. 2. Bounds $f(c,n)$ and $g(c,n)$ beside the real value of $\mathbb{P}(T_n > c\mathbb{E}(T_n)) = e(c,n)$ for $n = 1000$, as functions of c. In this case, we have $c^* = 1.12673$.

V. Asymptotic analysis of the distribution of T_n

We analyze in this section the behavior of the complementary distribution of T_n at point $c\mathbb{E}(T_n)$ when n tends to

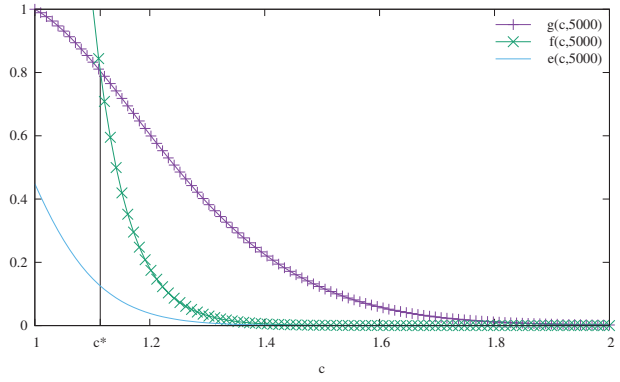


Fig. 3. Bounds $f(c,n)$ and $g(c,n)$ beside the real value of $\mathbb{P}(T_n > c\mathbb{E}(T_n)) = e(c,n)$ for $n = 5000$, as functions of c. In this case, we have $c^* = 1.11385$.

n	10	10^2	10^3	10^4	10^5	10^6	10^7
c^*	1.09	1.15	1.13	1.11	1.10	1.09	1.08

Fig. 4. Approximate values of c^* for different network sizes n.

infinity, depending on the value of c.

We prove in the following corollary that the bounds $f(c,n)$ and $g(c,n)$, obtained from Theorem 4 and Theorem 5 respectively with $k = c\mathbb{E}(T_n)$, both tend to 0 when n goes to infinity.

Corollary *7:* For all real number $c > 1$, we have

$$\lim_{n\longrightarrow\infty} f(c,n) = 0 \text{ and } \lim_{n\longrightarrow\infty} g(c,n) = 0.$$

Proof: For all $x \in [0,1)$, we have $\ln(1-x) \leq -x$. Applying this property to the bound $f(c,n)$ leads to

$$\begin{aligned} f(c,n) &\leq \left(1 + \frac{2c(n-1)H_{n-1}(n-2)^2}{n}\right) \\ &\quad \times e^{-2(c(n-1)H_{n-1}-2)/n} \\ &\leq \left(1 + 2c(n-2)^2 H_{n-1}\right) e^{-2(c(n-1)H_{n-1}-2)/n}. \end{aligned}$$

Since $\ln(n) \leq H_{n-1} \leq 1 + \ln(n-1)$, we get

$$\begin{aligned} f(c,n) &\leq \left(1 + 2c(n-2)^2(1+\ln(n-1))\right) \\ &\quad \times e^{-2(c(n-1)\ln(n)-2)/n} \\ &= \left(1 + 2c(n-2)^2(1+\ln(n-1))\right) e^{-2c\ln(n)} \\ &\quad \times e^{2(c\ln(n)+2)/n}. \end{aligned}$$

For $x \geq 0$, the function $u(x) = e^{2(c\ln(x)+2)/x}$ satisfies $u(x) \leq \exp\left(2c/e^{(c-2)/c}\right)$, so we obtain

$$f(c,n) \leq \frac{1 + 2c(n-2)^2(1+\ln(n-1))}{n^{2c}} \exp\left(2c/e^{(c-2)/c}\right).$$

The fact that $c > 1$ implies that this last term tends to 0 when $n \longrightarrow \infty$. Concerning the bound $g(c,n)$, we have

$$\begin{aligned} g(c,n) &= \frac{1}{c}\left(1 - \frac{2}{n}\right)^{(c-1-\ln(c))(n-1)H_{n-1}} \\ &= \frac{1}{c} e^{(c-1-\ln(c))(n-1)H_{n-1}\ln(1-2/n)} \\ &\leq \frac{1}{c} e^{-2(c-1-\ln(c))(n-1)H_{n-1}/n}, \end{aligned}$$

which tends to 0 when n tends to infinity, since $c - 1 - \ln(c)$ is positive for $c > 1$. ■

Theorem *8:* For all real $c \geq 0$, we have

$$\lim_{n\to+\infty} \mathbb{P}\{T_n > c\mathbb{E}(T_n)\} = \begin{cases} 0 & \text{if } c > 1 \\ \\ 1 & \text{if } c < 1. \end{cases}$$

Proof: From Corollary 7, both bounds $f(c,n)$ and $g(c,n)$ of $\mathbb{P}\{T_n > c\mathbb{E}(T_n)\}$ tend to 0 when n tends to infinity, so using either $f(c,n)$ or $g(c,n)$ we deduce that

$$\lim_{n\longrightarrow\infty} \mathbb{P}\{T_n > c\mathbb{E}(T_n)\} = 0 \text{ for all } c > 1.$$

In the case where $c < 1$, Theorem 3.1 of [20] leads to

$$\begin{aligned} \mathbb{P}\{T_n > c\mathbb{E}(T_n)\} &\geq 1 - e^{-2(n-1)H_{n-1}(c-1-\ln(c))/n} \\ &\geq 1 - e^{-2(n-1)\ln(n)(c-1-\ln(c))/n}. \end{aligned}$$

Since $c - 1 - \ln(c) > 0$ for all $c \in [0,1)$, the right-hand side term of this inequality tends to 1 when $n \longrightarrow \infty$. Thus, $\lim_{n\longrightarrow\infty} \mathbb{P}\{T_n > c\mathbb{E}(T_n)\} = 1$ when $c < 1$. ■

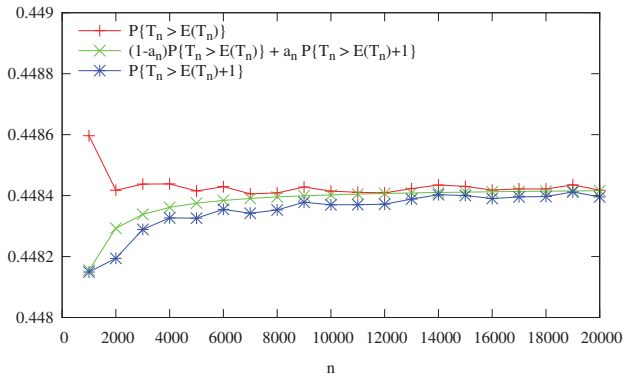


Fig. 5. $\mathbb{P}\{T_n > \mathbb{E}(T_n)\}$ as a function of n and its smoothing obtained with $a_n = \mathbb{E}(T_n) - \lfloor \mathbb{E}(T_n) \rfloor$.

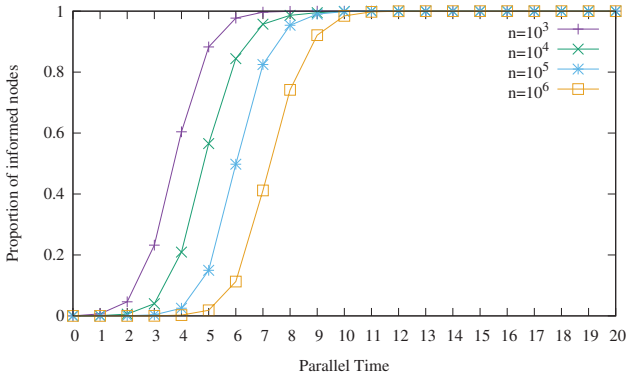


Fig. 6. Simulation results for the proportion of informed nodes as a function of parallel time

The results established previously don't allow us to figure out neither the existence of $\lim_{n\longrightarrow\infty} \mathbb{P}\{T_n > c\mathbb{E}(T_n)\}$ when $c = 1$, nor its value. However, numerical results give us a glimpse of its limiting behavior.

In Figure 5, we show the probability $\mathbb{P}\{T_n > \mathbb{E}(T_n)\}$ for different values of n. The oscillations of this probability with n are due to the fact T_n is a discrete random variable and $\mathbb{E}(T_n)$ is not an integer. That is why we propose in this figure a smoothing of this probability using the sequence

$$s_n = (1-a_n)\mathbb{P}\{T_n > \mathbb{E}(T_n)\} + a_n\mathbb{P}\{T_n > \mathbb{E}(T_n)+1\},$$

where a_n is the fractional part of $\mathbb{E}(T_n)$, that is $a_n = \mathbb{E}(T_n) - \lfloor \mathbb{E}(T_n) \rfloor$. Since $a_n \in [0,1]$, we have

$$\mathbb{P}\{T_n > \mathbb{E}(T_n)+1\} \leq s_n \leq \mathbb{P}\{T_n > \mathbb{E}(T_n)\},$$

that is why we also show in this figure the probability $\mathbb{P}\{T_n > \mathbb{E}(T_n)+1\}$. We also checked that the sequence (s_n) is increasing until $n = 20000$. This figure suggests the following result proposed as a conjecture.

Conjecture : $\lim_{n\longrightarrow\infty} \mathbb{P}\{T_n > \mathbb{E}(T_n)\}$ exists and ≈ 0.4484.

Figure 6 shows the results obtained by simulation concerning the proportion of nodes informed by rumor as a function of the parallel time. Recall that the parallel time refers to as the successive number of steps each node executes [5]. Initially, a

single node is informed of the rumor. This figure illustrates our analysis. For instance, with probability almost 1 one thousand nodes (resp. one million nodes) learn the rumor after no more than 7 (resp 11) interactions for each of them. The complexity in space (number of memory bits) is in $O(1)$.

VI. Conclusion

In this paper we have provided a thorough analysis of the rumor spreading time in the population protocol model. Providing such a precise analysis is a step towards the design of more complex functionality achieved by combining simple population protocols [23], [3]. Indeed, an important feature of population protocols is that they do not halt. Nodes can never know whether their computation is completed and thus nodes forever interact with their neighbors while their outputs stabilize to the desired value (e.g. the maximal value of any node of the network). By precisely characterizing, for each protocol of interest, with any high probability, the number of interactions each node must execute to converge to the desired value, each node can on its own, decide the time from which the current protocol has stabilized and start the parallel of sequential executions of the next ones.

References

[1] Huseyin Acan, Andrea Collevecchio, Abbas Mehrabian, and Wormald Nick. On the push&pull protocol for rumour spreading. In *Proceedings of the ACM Symposium on Principles of Distributed Systems (PODC)*, 2015.

[2] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[3] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(2):183–199, 2008.

[4] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[5] James Aspnes and Eric Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science, Distributed Computing Column*, 93:98–117, 2007.

[6] Noam Berger, Christian Borgs, Jennifer T. Chayes, and Amin Saberi. On the spread of viruses on the internet. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.

[7] Marin Bertier, Yann Busnel, and Anne-Marie Kermarrec. On gossip and populations. In *Proceedings of the International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2009.

[8] Keren Censor-Hillel, Bernhard Haeupler, Jonathan Kelner, and Petar Maymounkov. Global computation in a poorly connected world: Fast rumor spreading with no dependence on conductance. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, 2012.

[9] Flavio Chierichetti, Silvio Lattanzi, and Alessandro Panconesi. Rumor spreading in social networks. *Theoretical Computer Science*, 412(24):2602–2610, 2011.

[10] Andrea Clementi, Pierluigi Crescenzi, Carola Doerr, Pierre Fraigniaud, Francesco Pasquale, and Riccardo Silvestri. Rumor spreading in random evolving graphs. *Random structures and Algorithms*, 48(2):290–312, 2015.

[11] Sebastian Daum, Fabian Kuhn, and Yannic Maus. Rumor spreading with bounded indegree. In *Proceedings of the International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2016.

[12] Alan Demers, Mark Gealy, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dand Swinehart, and Doug Terry. Epidemic algorithms for replicated datbase maintenance. In *Proceedings of the ACM Syposium on Principles of Distributed Systems (PODC)*, 1987.

[13] Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures and Algorithms*, 1(4):447–460, 1990.

[14] Nicolaos Fountoulakis and Konstantinos Panagiotou. Rumor spreading on random regular graphs and expanders. *Random Structures and Algorithms*, 43(2):201–220, 2013.

[15] Nicolaos Fountoulakis, Konstantinos Panagiotou, and Thomas Sauerwald. Ultra-fast rumor spreading in social networks. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, 2012.

[16] Alan Frieze and Geoffrey Grimmet. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 85.

[17] George Giakkoupis. Tight bounds for rumor spreading in graphs of a given conductance. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2011.

[18] George Giakkoupis. Tight bounds for rumor spreading with vertex expansion. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014.

[19] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource discovery in distributed networks. In *Proceedings of the ACM Syposium on Principles of Distributed Systems (PODC)*, 1999.

[20] Svante Janson. Tail bounds for sums of geometric and exponential variables. Technical report. `http://www2.math.uu.se/~svante/papers/sjN14.pdf`

[21] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2000.

[22] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.

[23] Othon Michail and Paul Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 81:1–10, 2015.

[24] Yves Mocquard, Bruno Sericola, Samantha Robert, and Emmanuelle Anceaume. Analysis of the Propagation Time of a Rumour in Large-scale Distributed Systems. Technical report, 2016. `https://hal.archives-ouvertes.fr/hal-01354815`

[25] Konstantinos Panagiotou, Xavier Perez-Gimenez, Thomas Sauerwald, and Hé Sun. Randomized rumor spreading: the effect of the network topology. *Combinatorics, Probability and Computing*, 24(2):457–479, 2015.

[26] Konstantinos Panagiotou and Leo Speidel. Asynchronous rumor spreading on random graphs. *Algorithmica*, 2016.

[27] Bruno Sericola. *Markov Chains. Theory, Algorithms and Applications*. Applied stochastic methods series. WILEY, 2013.

[28] Boyd Stephen, Ghosh Arpita, Prabhakar Balaji, and Shah Devavrat. Randomized gossip algorithms. *IEEE/ACM Transactions on Networking*, 14:2508–2530, 2006.

Cost Sensitive Moving Target Consensus

Sisi Duan
Oak Ridge National Laboratory
Email: duans@ornl.gov

Yun Li
University of California, Davis
Email: yunli@ucdavis.edu

Karl Levitt
University of California, Davis
Email: levitt@cs.ucdavis.edu

***Abstract*—Consensus is a fundamental approach to implementing fault-tolerant services through replication. It is well known that there exists a tradeoff between the cost and the resilience. For instance, Crash Fault Tolerant (CFT) protocols have a low cost but can only handle crash failures while Byzantine Fault Tolerant (BFT) protocols handle arbitrary failures but have a higher cost. Hybrid protocols enjoy the benefits of both high performance without failures and high resiliency under failures by switching among different subprotocols. However, it is challenging to determine which subprotocols should be used. We propose a moving target approach to switch among protocols according to the existing system and network vulnerability. At the core of our approach is a formalized cost model that evaluates the vulnerability and performance of consensus protocols based on real-time Intrusion Detection System (IDS) signals. Based on the evaluation results, we demonstrate that a safe, cheap, and unpredictable protocol is always used and a high IDS error rate can be tolerated.**

***Index Terms*—Consensus, state machine replication, crash fault tolerance, Byzantine fault tolerance, moving target defense.**

I. Introduction

Consensus is a generic technique that implements fault-tolerant services through replication. It is critical to achieve both reliability and availability in various online services and cloud computing applications, including Google's Chubby [6], Amazon Web Services [1], and VMware's vSphere [2, 3]. Depending on the types of failures we aim to tolerate, various protocols are designed with different security guarantees and performance characteristics. It is in general known that there exists a tradeoff between the resiliency and the cost of the consensus protocols. Namely, a low cost protocol with low redundancy and high performance can only handle limited types of failures. For instance, Crash Fault Tolerant (CFT) protocols are less redundant but can only tolerate crash failures. In comparison, Byzantine Fault Tolerant (BFT) protocols handle arbitrary failures but are very expensive, which may be an overkill to use most of the time.

Several approaches have been proposed to switch among different protocols depending on an estimation of failures in the system [16, 20, 23, 37], which enjoy the benefits of both high performance in the normal cases and high resiliency under failures. However, there still exist some issues when handling a wide range of failures. First, most approaches employ two subprotocols. Therefore, there is usually an obvious performance degradation even when it is not necessary to use a high cost protocol. Second, if we employ more than two subprotocols to not suffer from large performance degradation, it is hard to determine which one should be used according to existing system and network vulnerability. Third, the switching of protocols is usually deterministic and predictable, which makes the system more vulnerable since the attackers have enough time to collect protocol information, prepare, and complete an attack.

978-1-5090-3216-7/16/31.00

In this paper, we propose a cost sensitive approach motivated by Moving Target Defense (MTD) to build a resilient consensus model that handles various types of failures according to system and network vulnerability. At the core of our idea is a formalized cost model that evaluates the vulnerability and performance of leader-based state machine replicated fault-tolerant consensus protocols. We use *damage cost* to represent the vulnerability of the protocols and *operational cost* to evaluate the running cost. The input of the cost model is a set of probabilities from IDS, each of which represents the certainty of a replica being normal, crash, or Byzantine. We view the IDS as an oracle that provides the probabilities periodically. The underlying idea is that if the IDS can provide a rough reference about which replica(s) might be faulty without actively participating the protocols, we will be able to use a protocol that causes the lowest damage to the system while achieving high performance.

Due to the use of our approach, a protocol that is safe, cheap, and unpredictable is always used. According to the cost model, we select a cluster of protocols with low cost according to existing system vulnerability. By switching protocols randomly among the cluster, an unpredictable protocol is always used. In addition, the formalized cost model can also be viewed as a theoretical model to analyze the consensus protocols. We illustrate our cost model with 8 consensus protocols and our evaluation results show that the selection of the protocols naturally follows the properties of the protocols. In addition, we handle crashing IDS through the use of a fail-safe protocol and we also show in the evaluation that our approach tolerates a high error rate for the IDS.

The contributions of the paper are summarized as follows. (1) We propose a formalized cost model to evaluate the vulnerability and performance of consensus protocols based on real-time IDS signals, which can also be viewed as a theoretical analysis model (§IV); (2) We illustrate our cost model using 8 different consensus protocols (§V); (3) Based on the cost model, we present a moving target consensus approach to select a cluster of safe, cheap, and unpredictable protocols (§VI); (4) Our evaluation results show that a cluster of both safe and fast protocols is always selected. In addition,

we tolerate a high IDS error rate and also handle the case where the IDS crashes (§VII).

II. Related Work

Most consensus protocols proceed in rounds [3, 7, 9, 13, 16, 23, 24, 35]. A number of approaches rely on (small) trusted components to prevent equivocation and handle Byzantine failures using fewer than $3f + 1$ replicas [8, 12, 20]. For instance, ByzID [12] relies on specification-based IDS [22] to passively monitor the consistency of the messages. We also rely on a trusted IDS that may fail by crashing. In contrast, we employ the IDS to evaluate the vulnerability of the replicas from the network and view it as an oracle.

A number of previous efforts have been made to evaluate distributed algorithms [21, 34]. We use similar measures to evaluate operational cost. In addition, we also include damage cost to evaluate the vulnerability of the system.

The cost factors and cost model have previously been proposed and their definitions are usually subjective to the specific problems [29]. Lee et al. [25] discuss cost factors related to intrusion detection: damage cost, response cost, and operational cost. They assign values empirically to cost factors based on the IDS results and improve the model by reducing some of the cost factors. We use similar terms but with different definitions for consensus protocols.

MTD has been applied to various of areas such as cyber security [18] and mobile wireless networks [15, 33] using techniques such as randomization [19]. In order to take advantage of cost-sensitive model as well as randomization, we do not always choose the minimum-cost solutions [25, 32] but take into concerns of both vulnerability and running cost. Turtle consensus [28] uses a MTD approach that switches between CFT protocol in each round to handle DoS failures but the switching of protocols is deterministic. In comparison, our paper considers a wide range of failures.

III. Preliminaries

In this section, we introduce the system model, our IDS model, and the background of consensus protocols.

A. System Model

We consider a distributed system with n replicas $\mathcal{P} = \{p_0, p_1, \cdots p_{n-1}\}$. Each replica can be viewed as a state machine following some protocol, where a protocol specifies the communication between replicas. We distinguish the types $\mathcal{T}$ of correctness for each replica: correct, crash, or Byzantine. A correct node faithfully follows the corresponding protocol. Faulty replicas may fail by crashing (stop executing the protocol) or be Byzantine (behave arbitrarily). The Byzantine failures we aim to handle are mainly caused by adversary attacks from the network. We assume fair-loss links, where if a message is sent infinitely often by a correct replica, a correct receiver will receive the message infinitely often. Liveness is ensured under partial synchrony [14]: synchrony holds after some unknown global stabilization time.

B. IDS Model for Moving Target Defense

An Intrusion Detection System (IDS) monitors the correctness of each replica p_i (near) real-time, which can fail by crashing. It monitors the host and network devices and detects events that could indicate an ongoing attack [11, 22, 27]. We view the IDS as an oracle that outputs a set of signals (N, C, A) with three probabilities $P_{i,N}$, $P_{i,C}$, and $P_{i,A}$. The three signals represent replica p_i being Normal (correct), Crashed, or under Attack (Byzantine). The three probabilities represent the confidence of each signal, which are refreshed periodically. The value of $P_{i,A}$ is set to 0 initially or after recovery. The IDS also evaluates the cost using our cost model and notifies the replicas the protocol to run. There are several ways to deploy the IDS. For example, we can deploy a network-based IDS over a LAN with a passive architecture. In a passive architecture, it monitors a copy of actual network traffic while no traffic passes through the sensor [30].

C. Consensus Protocols

Consensus protocols tolerate a certain number of failures through the use of redundant replicas. Correctness includes *safety* and *liveness*. Safety guarantees that all the correct replicas decide on the same value and liveness guarantees that all the correct replicas eventually decide. In order to handle f failures, different protocols may vary significantly regarding the minimum number of replicas. We specify three types of protocols: $\boldsymbol{C}$**-1** represents CFT protocols that require at least $f + 1$ replicas where safety may not be guaranteed, $\boldsymbol{C}$**-2** denotes CFT protocols that require at least $2f + 1$ replicas, and $\boldsymbol{B}$**-1** represents BFT protocols that require at least $3f + 1$ replicas. Without loss of generality, for each protocol that requires at least n replicas, we assume there are n replicas. When each protocol is run, safety of the protocol is guaranteed only when the number of corresponding $\mathcal{T}$ failures does not exceed f. Although replicas may be temporarily inconsistent, they can be consistent after switching to another protocol in our approach.

We consider leader-based consensus protocols that operate in *rounds*, where in each normal round a client request is received by replicas, assigned with a sequence number by the leader, agreed by the replicas, executed, and eventually the result is received by the client. The leader is also called the *primary*, which initializes each round. We assume by default the primary is p_0. Other nodes are called *backups* or *backup nodes*. Each protocol consists of several *phases*, where in each phase each replica receives and authenticates certain number of messages and sends messages to some replica(s).

We consider two classes of protocols: *broadcast-based* and *chain-based* [13, 35]. Among broadcast-based protocols, we further distinguish *primary-backup* approaches [5, 10, 17], where primary can communicate with backups but backups do not communicate with each other. In comparison, in regular broadcast-based approaches [7, 20, 23, 24], replicas are fully connected. In contrast, chain-based protocols organize replicas into a logical chain and the head is considered the primary.

Most protocols have *view change* scheme where a backup node takes over when existing primary is faulty. Some protocols have *reconfiguration* scheme to replace some faulty replicas. Protocols may use MACs or digital signatures for authentication. Unless otherwise mentioned, we assume MACs are used. Every protocol has a *checkpoint* scheme, where a replica periodically generates a snapshot of its state, signs a checkpoint message, and sends to other replicas. After the checkpoint becomes stable, previous messages are discarded.

IV. Cost Model

In this section, we formalize the measurement of the costs based on notations in TABLE I. Cost evaluation plays a very important role in our approach to select protocols. Therefore, to precisely evaluate cost factors and select appropriate protocols, we aim at cost metrics that follow these principles: 1) Cost metrics should be measured consistently [26]. 2) Source data should be cheap to gather in terms of time or money. Based on our formalized cost model, the cost can be easily calculated according to the IDS signals. 3) Cost metrics should be evaluated to a value with an associated unit of measures that characterize the value. We define *damage cost* as the number of lost or delayed requests, and *operational cost* as the time of running a normal round of the protocols. The cost factors considered here are standard quantities that all consensus protocols can adopt to perform the cost analysis.

Notation	Meaning
$P_{i,N}$	The probability replica p_i is Normal/correct.
$P_{i,C}$	The probability replica p_i has Crashed.
$P_{i,A}$	The probability replica p_i is being Attacked/Byzantine.
t_0	Transmission time between two replicas.
t_c	Transmission time between client and a replica.
t_1	The time it takes for an IDS to report a crash or an attack.
T_m, T_d	The time it takes to verify or generate a MAC/digital signature.
T_V	The time it takes for replicas to move to a new view.
T_R	The time it takes for recovery/reconfiguration.
λ	The number of requests in each checkpoint.
δ	Number of incoming requests per second.
Δ	Number of incoming requests since last checkpoint.

TABLE I
Notations.

Damage Cost (C_D). The damage cost evaluates the vulnerability of the protocol. We measure it as the sum of expected number of requests that will be lost or delayed due to the faulty replicas, as shown in Equ. (1). The input is a set of probabilities $P_{i,N}, P_{i,C}$, and $P_{i,A}$ for $i = 0, \cdots, n-1$. The $R_{i,c}$ and $R_{i,A}$ are fixed values that denote the number of requests that may be lost or delayed due to the failure of the replica p_i being $\mathcal{T}$ type where $\mathcal{T} = C$ or A. When we consider the cost for each individual replica p_i, we assume p_i is faulty and the number of $\mathcal{T}$ faulty replicas in $\mathcal{P}$ does not exceed f.

$$C_D = \sum_{i=0}^{n-1} (R_{i,C} P_{i,C} + R_{i,A} P_{i,A}) \tag{1}$$

The values for $R_{i,C}$ and $R_{i,A}$ depend on the identity of replica p_i and the protocol, as summarized below.

The primary crashes. For protocols that have view changes and $T_V < t_1$, the incoming requests during T_V time are lost since nodes cannot process any other requests, i.e., $R_{0,C} = T_V \delta$. Otherwise, all the requests since the failure of the primary will be lost, i.e., $R_{0,C} = t_1 \delta$. For simplicity, we only include the case where $T_V < t_1$.

The primary is Byzantine. We assume that the last checkpoint is stable for all the replicas. For CFT protocols with arbitrary failures, all the requests since the last checkpoint are considered lost since there is no guarantee of the safety of the protocol. Therefore, $R_{0,A} = \Delta$. In comparison, the BFT approaches handle this case through view changes. Therefore, the cost is the same with the previous case, i.e., $R_{0,A} = T_V \delta$.

A backup node crashes. All the broadcast-based protocols naturally handle the crash of backup nodes, i.e., there is no cost for this case. However, chain-based protocols suffer from backup failures. For instance, Chain [35] reconfigures faulty replicas. Therefore, requests during reconfiguration are delayed, i.e., $R_{i,C} = T_R \delta$.

A backup node is Byzantine. The correctness of the protocols is closely related to the primary. In C-1 protocols, the requests since last checkpoint will be lost, i.e., $R_{i,A} = \Delta$. This is because the case is indistinguishable from the case where the primary is Byzantine. In contrast, in a broadcast-based C-2 protocol, correct replicas are still consistent if the primary is correct. This is because the primary always sends consistent messages to the replicas. On the other hand, most broadcast-based BFT protocols handle backup failures. An exception happens to protocols like Zyzzyva [23] since it employs two subprotocols. Also, similar to the previous case, chain-based protocols also suffer from backup failures.

Operational Cost (C_O). The operational cost evaluates the cost to run the protocols. We present two types, latency ($C_{O,L}$) and throughput ($C_{O,T}$), and both are evaluated in terms of time according to similar metrics of previous work [34]. The smaller the latency $C_{O,L}$, the smaller the throughput $C_{O,T}$, and the larger the actual throughput will be. In addition, if the checkpoint is not required frequently, the cost of the checkpoint is not included in the operational cost.

Latency $C_{O,L}$. The latency cost is measured as the time of a consensus round starting from the leader receiving the request to the end of the round of agreement. It includes the transmission time and the time for authentication. $C_{O,L}$ is computed according to the following equation.

$$C_{O,L} = \sum_{j=1}^{\#phases} (n_{j,c} T_m + t_0 + n_{j,1} T_m + n_{j,2} T_m) + \varepsilon \tag{2}$$

In the above equation, $n_{j,c}$ is the number of MACs the replica p_j needs to verify and generate for the client, $n_{j,1}$ is the minimum number of MACs p_j needs to authenticate in each phase, and $n_{j,2}$ is the number of MACs each replica needs to generate in each phase. The cost is measured for the normal case when there is no message congestion. Notice that, although in broadcast-based protocols, all the replicas need to run the same steps, they run concurrently. Therefore, we measure the cost for each phase as the cost of a single

replica if all the replicas execute the same step. The total cost will be the sum of cost for each phase. We also include a variable ε, which includes the cost caused by switching to the new protocol, e.g. physical cost of starting a new replica. For simplicity, we do not include it in the examples in §V.

Throughput $C_{O,T}$. $C_{O,T}$ evaluates the time for authenticating and generating MACs or digital signatures of the bottleneck node. This is due to the fact that the bottleneck replica can continue processing new messages before a consensus round completes. Therefore, the transmission time is not included.

$$C_{O,T} = \sum_{j=1}^{\#phases} (n_{j,c}T_m + n_{j,1}T_m + n_{j,2}T_m) \tag{3}$$

V. Consensus Cost

We introduce the costs of 8 protocols to illustrate our cost model, with both CFT protocols and BFT protocols. Specifically, we survey the cost of two C-1 protocols, Remus and Semi-Active, which can guarantee safety only when all the replicas are correct. We also include two C-2 protocols, Paxos and Chain, which are the state-of-the-art CFT protocols. Finally, we present four BFT protocols, Aliph-Chain, BChain, PBFT, and Zyzzyva, which have different performance characteristics and are perfect for case study. In this section, we first briefly introduce the protocols and then show their cost using the notations in TABLE I.

Remus [9]: A primary-backup C-1 approach where the backups periodically obtain checkpoints from the primary to maintain the latest state. It can also be viewed as a semi-passive approach [10]. In the presence of failures, the requests since last one or two checkpoints will be lost depending on the time IDS detects the failure. Namely, if $t_1\delta$ is greater than λ, a node has already been faulty the last stable checkpoint. Therefore, two checkpoints will be lost. Otherwise, only one checkpoint will be lost.

$$C_D = \begin{cases} 2\lambda P_{0,C} + \Delta P_{0,A} + \sum_{i=1}^{f} \Delta P_{i,A} & \text{if } t_1\delta > \lambda \\ \lambda P_{0,C} + \Delta P_{0,A} + \sum_{i=1}^{f} \Delta P_{i,A} & \text{otherwise} \end{cases} \tag{4}$$

$$C_{O,L} = C_{O,T} = \frac{2\lambda T_m + T_d}{\lambda} \tag{5}$$

Semi-Active [3]: A primary-backup C-1 approach where the primary notifies the backups each incoming request so that all the replicas execute them directly. Since backup nodes receive all the requests from the primary, only those requests during view changes will be lost, i.e., $T_V\delta$.

$$C_D = T_V\delta P_{0,C} + \Delta P_{0,A} + \sum_{i=1}^{f} \Delta P_{i,A} \tag{6}$$

$$C_{O,L} = C_{O,T} = (2+f)T_m \tag{7}$$

Paxos [24]: A broadcast-based C-2 approach with two phases: the leader first notifies the backups of the incoming request; each replica sends a message to all other replicas. If a replica collects at least $f+1$ matching messages, it executes the request and sends a reply to the client.

$$C_D = T_V\delta P_{0,C} + \Delta P_{0,A} \tag{8}$$

$$C_{O,L} = (5f+3)T_m + 2t_0 \tag{9}$$

$$C_{O,T} = (3f+2)T_m \tag{10}$$

Chain [35]: A chain-based CFT approach. The head receives a request from a client and then sends along the chain towards the tail and the tail replies to the client. When a replica crashes, a non-faulty master node reconfigures the chain. When a node fails, all the request during reconfiguration will be lost.

$$C_D = T_R\delta P_{0,C} + \Delta P_{0,A} + \sum_{i=1}^{2f} T_R\delta(P_{i,C} + P_{i,A}) \tag{11}$$

$$C_{O,L} = 2(2f+1)T_m + 2ft_0 \tag{12}$$

$$C_{O,T} = 2T_m \tag{13}$$

Aliph-Chain [16]: A chain-based BFT approach. Each replica needs to verify MACs from at most $f+1$ previous replicas and also append MACs for up to $f+1$ subsequent replicas or the clients. The client accepts the reply message when it receives a message from the tail with $f+1$ valid MACs. In the equations, function $F(i)$ represents the latency up to replica p_i.

$$C_D = t_1\delta(P_{0,C} + P_{0,A}) + \sum_{i=1}^{3f}(t_1 + F(i))\delta(P_{i,C} + P_{i,A}) \tag{14}$$

$$C_{O,L} = \sum_{i=0}^{f-1}(f+i+2)T_m + \sum_{i=f}^{2f-1}(2f+2)T_m + \sum_{i=2f}^{3f}(4f-i+2)T_m + 3ft_0 \tag{15}$$

$$C_{O,T} = (2f+2)T_m \tag{16}$$

BChain [13]: A chain-based BFT approach. Being different from Chain and Aliph-Chain, only the first $2f+1$ replicas form a chain and the last f replicas serve as backups which are reconfigured periodically and the message is sent from the head to the $(2f+1)^{th}$ replica. It uses similar authentication scheme with Aliph-Chain. All the first $2f+1$ replicas notify the rest f replicas the execution order so that they are also up-to-date. When failures occur, the chain is reordered by the head with at most f rounds of reconfigurations.

$$C_D = T_V\delta(P_{0,C} + P_{0,A}) + \sum_{i=1}^{2f} T_R\delta P_{i,C} + 3fT_R\delta \sum_{i=1}^{2f} P_{i,A} \tag{17}$$

$$C_{O,L} = \sum_{i=0}^{f-1}(2f+2i+3)T_m + \sum_{i=f}^{2f}(6f-2i+3)T_m + 4ft_0 \tag{18}$$

$$C_{O,T} = (4f+3)T_m \tag{19}$$

PBFT [7]: A leader-based BFT approach with three phases: in the first phase the leader notifies the replicas of the incoming request; replicas exchange their messages until each correct replica collects at least $2f+1$ matching messages in the second and third phase. Replicas then reply to the clients.

$$C_D = T_V\delta(P_{0,C} + P_{0,A}) \tag{20}$$

$$C_{O,L} = (13f+3)T_m + 3t_0 \tag{21}$$

$$C_{O,T} = (10f+2)T_m \tag{22}$$

Zyzzyva [23]: A leader-based BFT approach where clients participate. The leader first notifies the replicas and the replicas directly send a reply to the client. If the client receives matching replies from all the replicas, it accepts the message. If it receives at least $2f+1$ matching messages, it sends them to all the replicas. The replicas then commit the request.

$$C_D = T_V\delta(P_{0,C} + P_{0,A}) + \sum_{i=1}^{3f}\left(\frac{t_1}{(25)(a)} - \frac{t_1}{(25)(b)}\right)(P_{i,C} + P_{i,A}) \tag{23}$$

$$C_{O,L} = \begin{cases} (6f+3)T_m + t_0 & \text{if } 3f+1 \text{ matching} \\ (10f+5)T_m + t_0 + 2t_c & \text{if } 2f+1 \text{ matching} \end{cases} \tag{24}$$

$$C_{O,T} = \begin{cases} (6f+2)T_m & \text{if } 3f+1 \text{ matching} \quad (a) \\ (8f+3)T_m & \text{if } 2f+1 \text{ matching} \quad (b) \end{cases} \tag{25}$$

VI. A Moving Target Consensus Approach

In this section, we first briefly overview our approach and introduce the procedures for switching protocols. Then we show our moving target algorithm for selecting protocols in details. Finally, we show the lower bound for the IDS values and discuss the case when IDS crashes.

Overview of the Protocol. We illustrate our approach in Fig. 1. It contains three components: the IDS, a set of available consensus protocols, and a set of replicas. The IDS monitors the correctness of the replicas and periodically evaluates the costs of the protocols based on the *Moving Target Algorithm*. It selects a protocol and sends configuration messages to the replicas. Namely, by default a cluster of protocols is selected and a random one is used periodically on a set of replicas $\mathcal{P}$. If the damage cost of existing protocol is higher than a threshold, the IDS selects a new cluster of protocols. The IDS can also select a set of new replicas $\mathcal{P}'$ according to the correctness of the replicas. After receiving the configuration message from the IDS, the replicas switch to the new protocol following the procedures in *Moving Target Consensus*.

The underlying idea is that given the IDS indication of failures of some replicas, running the same protocol may cause a large number of lost or delayed requests. If the damage is higher than expected, we should select a set of protocols that causes lower damage while still achieving good performance. The cost model provides the flexibility of selecting the right protocols according to both network and system vulnerability and user requirements.

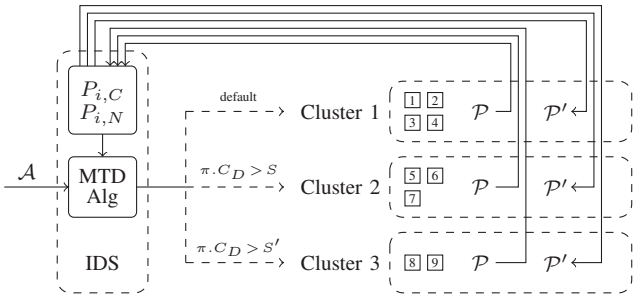


Fig. 1. Moving Target Consensus

Moving Target Consensus. We assume existing protocol π is run on a set of replicas $\mathcal{P}$. Replicas periodically generate checkpoints and authenticate them using digital signatures. In addition to the regular checkpoint steps of the protocols, if π is a C-1 protocol, we let the replicas also forward their checkpoint messages to the IDS.

The IDS updates the costs for all the protocols and sends a configuration message to the replicas periodically. A configuration message includes a protocol π', a set of replicas $\mathcal{P}'$, and the id of a default primary. A protocol π' is randomly selected 1) periodically in the same cluster of π, or 2) when the damage cost of π is higher than the threshold S. In the latter case, a new cluster is selected and a random one is used, as we will discuss in Algorithm 1. All the replicas in both $\mathcal{P}$ and $\mathcal{P}'$ also forward the configuration message to each other to guarantee that the configuration is learned.

If $\mathcal{P}'$ is a subset of $\mathcal{P}$, the leader in $\mathcal{P}'$ *initializes protocol* π' (more details later) and replicas start executing the protocol. Otherwise, replicas need to first *obtain the last stable checkpoint* and then *the new primary initializes* π'.

There are two cases for replicas to obtain the last stable checkpoint. If π is a C-1 protocol, the replicas in $\mathcal{P}'$ obtain a stable checkpoint directly from the IDS. In all other cases, replicas in $\mathcal{P}'$ need to obtain checkpoints from $\mathcal{P}$. Namely, after receiving a configuration message from the IDS, replicas in both $\mathcal{P}'$ and $\mathcal{P}$ send checkpoints to each other. In a checkpoint, with a sequence number greater than the last stable checkpoint, a replica includes all the committed requests in $\mathcal{O}$ and all the accepted but uncommitted requests in $\mathcal{U}$. For C-1 protocols and Aliph-Chain, all the requests are included in $\mathcal{U}$. For C-2 protocols, if a replica receives matching message from $2f+1$ replicas during protocol π, the request is included in $\mathcal{O}$ and other requests are included in $\mathcal{U}$. For B-1 protocols besides Aliph-Chain, each replica includes the committed requests according to the protocols, e.g., a valid *ack* in BChain, etc. If the new primary receives matching checkpoints from at least $f+1$ replicas, it starts π'.

In order to initialize π', the primary selects the last stable checkpoint and uses the state and sequence number l. The primary then determines L where $l+L$ is the largest sequence number found in $\mathcal{O}$ and $\mathcal{U}$. For each sequence number, the new primary selects a request M if at least $f+1$ replicas include M in $\mathcal{O}$ or at least $2f+1$ replicas include M in $\mathcal{U}$ (or $f+1$ for C-1 protocols). It then sends a message to all the replicas in $\mathcal{P}'$. The message includes the last stable checkpoint and a set of selected requests. After receiving the message, replicas process the requests according to π'.

We show in Theorem 1 the switching of protocols is both safe and live. We include the proofs for all the theorems in the Appendix.

Theorem 1. *Let protocol π on a set of replicas $\mathcal{P}$ be switched to protocol π' on a set of replicas $\mathcal{P}'$. If π' tolerates failures with type $\mathcal{T}$ and there are fewer than f $\mathcal{T}$ failures in both π and π', the switching of protocols is both safe and live.*

The Moving Target Algorithm. The underlying idea of out algorithm for selecting protocols is that based on the IDS signals, we evaluate the cost of the existing protocol. If the existing protocol is considered vulnerable regarding a threshold S, we select a new cluster of protocols that is safe and cheap. As shown in Algorithm 1, $\mathcal{A}$ represents all the available protocols we can use, which initially includes all the protocols. The function $top(x, \mathcal{B}.y)$ selects the x^{th} largest value according to the y value in set $\mathcal{B}$. We set up the threshold S to be the $\sigma|\mathcal{A}|^{th}$ largest of the damage cost for protocols in $\mathcal{A}$ where $\sigma \in (0,1)$. When the damage cost of existing protocol is higher than S, indicating that existing protocol may cause larger damage than expected, we start selecting a new cluster. We first filter all the protocols with higher damage cost from $\mathcal{A}$. Then we select protocols with operational cost smaller than the $\theta|\mathcal{A}|^{th}$ protocol according to the operational cost, where $\theta \in (0,1)$. Finally, we do an optional step among protocols in $\mathcal{R}$ to further filter protocols with outstanding

damage cost. Namely, we set up another threshold Λ for damage cost and filter the protocols with damage cost higher than $h + \Lambda$ where h represents the lowest damage cost for protocols in $\mathcal{R}$.

Algorithm 1 The Moving Target Algorithm

```
S ← top(σ|A|, A.C_D)
if π.C_D > S then                {Damage cost of existing protocol is high}
    A ← A.C_D < S                {Remove protocols with high damage cost}
    O ← top(θ|A|, A.C_{O,L})
    R ← A.C_{O,L} < O            {Select protocols with low operational cost}
    h ← top(|R|, R.C_D)
    C ← R.C_D < h + Λ            {Remove protocols with outstanding cost}
```

Notice that we use three parameters: σ, θ, and Λ. σ is used for threshold S in order to determine whether the damage cost of existing protocol is higher than a portion of protocols in $\mathcal{A}$. Similarly, θ is a threshold that is used to select a set of protocols in $\mathcal{A}$ with the lowest operational cost. It is important to select protocols with the similar performance given the damage cost is lower than S. Lastly, Λ is an optional threshold that is used to further choose protocols with low damage cost based on the previous selection. The values of σ and θ can be set up according to the requirement. However, the value of Λ is important to guarantee that we select the right protocols. As shown in Theorem 2, it is also related to the P values from IDS.

Theorem 2. *Let Ω be the damage cost caused by backups for BFT protocols and Λ be the threshold for selecting a cluster. In order for the approach to be safe, the following requirement for IDS holds, where $min(\Omega)$ represents the BFT protocol with minimum damage cost caused by all the backups.*

$$P_{0,A} > \frac{\Lambda + min(\Omega)}{\Delta + T_V \delta} \tag{26}$$

Dealing with Crashing IDS. The IDS generates configuration messages periodically. In order to handle the case where IDS crashes, each replica starts a timer after receiving a configuration message and waits for the next configuration message. If the replica does not receive any configuration message before its timer expires, it sends a $[cids]$ message to other replicas. If a replica receives more than $f + 1$ $[cids]$ messages, it also sends a $[cids]$ messages to other replicas. All the replicas then learn that the IDS has crashed. Then replicas switch to a default fail-safe protocol, in our case PBFT. This is due to the fact that PBFT, in general, has the lowest damage cost among all the protocols we use. This guarantees that the protocol is still safe when IDS crashes. Notice that C-1 protocols require only $f + 1$ replicas. In this case, the failure of IDS can only be detected if all the replicas are correct.

MTD Entropy. Based on Shannon's information entropy [31], MTD entropy is formalized to evaluate the randomness and effectiveness of the MTD model [36]. Specifically, the greater the entropy of the configuration of an MTD system, the more effective the approach is to prevent future attacks. We show the entropy of our approach in Theorem 3.

Theorem 3. *Let $\mathcal{A} = \{\pi_1, \pi_2, \cdots, \pi_m\}$ represent the m protocols we can use. $H(\mathcal{A})$ represents the MTD entropy, which can be denoted as:*

$$H(\mathcal{A}) = H(\pi_1, \pi_2, \cdots, \pi_m) = \sum_{i=1}^{m} p(\pi_i) log(p(\pi_i)) \tag{27}$$

$p(\pi_i)$ represents the possibility π_i is selected:

$$p(\pi_i) = \frac{1}{\sigma|A|} \tag{28}$$

Given that Λ is large enough, after selecting a cluster, the probability of each protocol being used is:

$$p(\pi_i) = \frac{1}{(1-\theta)(1-\sigma)|A|} \tag{29}$$

Based on this theorem, if the switching of protocols is deterministic, the entropy is 0 since the probability of each protocol is 1. In comparison, in our example, $|A| = 8$ and let $\sigma = \Lambda = 0.2$, the entropy for our approach is 8.68 initially and 96.51 after switching. If we simply switch among all the 8 protocols, the entropy is 192.00. We conclude that due to the unpredictability of our approach, we can also prevent further attacks using the randomization method. If there are more protocols in the same cluster, the effectiveness can be further increased.

VII. Evaluation

In this section, we show the evaluation of the effectiveness of our cost model in selecting protocols and the IDS error rate our approach can handle.

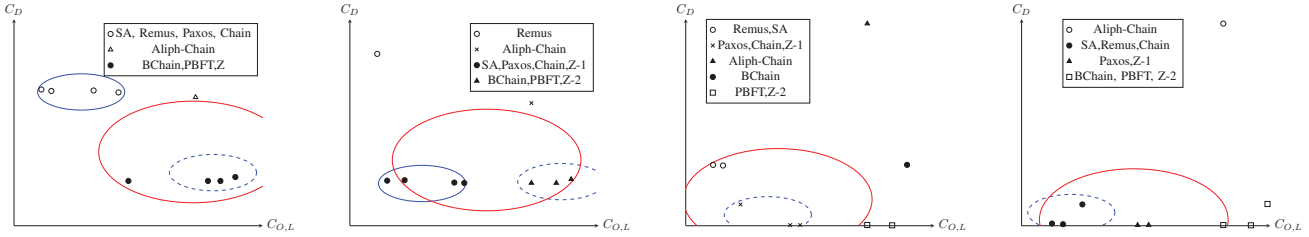
Setting	λ	T_m	T_d	T_R/T_V	$t_1/t_0/t_c$	δ	Δ	P
1	10	0.5	1.0	0.5	1.0	10	15	0.001
2	10	0.5	1.0	0.5	5.0	10	60	0.01
3	10	0.5	1.0	0.1	1.0	10	20	0.01
4	10	0.5	1.0	0.1	10.0	10	105	0.1

TABLE II
Experiment settings. T_m, T_d, T_R, T_V, t_1,t_0, and t_c are measured in ms. P is the default value of the replicas unless specified.

Implementation and Settings. The implementation of the protocols is based on Castro et al.'s implementation of PBFT. We evaluate throughput under failures based on our cost model using $0/0$ benchmark, where the clients issue 0kB request and receive 0kB replies. We test the throughput to demonstrate the effectiveness of our cost model. Experiments are carried out on DeterLab [4], utilizing a cluster of up to 20 identical machines connected through a 100 Mbps switched LAN. Each machine is equipped with a 3 GHz Xeon processor and 2 GB of RAM.

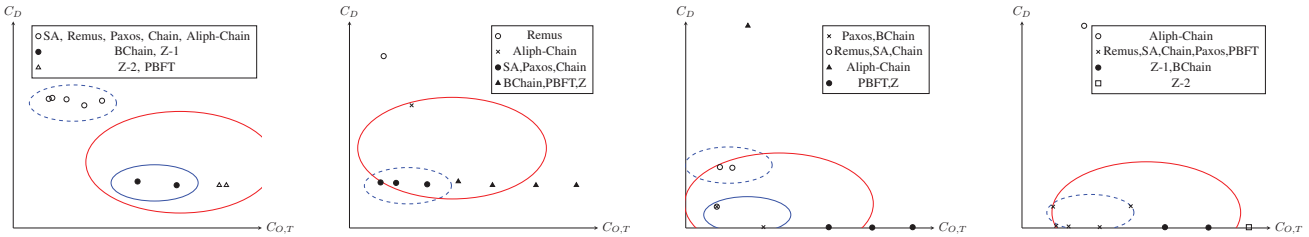
We use several parameters in our cost model. Among them, the P values are the output of the IDS. The values of δ, λ, and Δ are all fixed or preset. In comparison, the values of t_0, t_1, T_m, T_V, and T_R can be obtained through testing. Although the values can be different for different protocols or even for different rounds of each protocol, we can still test them and use average values to measure the cost. For instance, we measure t_0 as the half of the average round trip transmission time between any two correct nodes.

Additionally, we use IDS as an oracle in our cost model. We assign different values to assess our cost model. We evaluate our cost model using 4 settings, as shown in TABLE II, where P represents the default values unless specified.



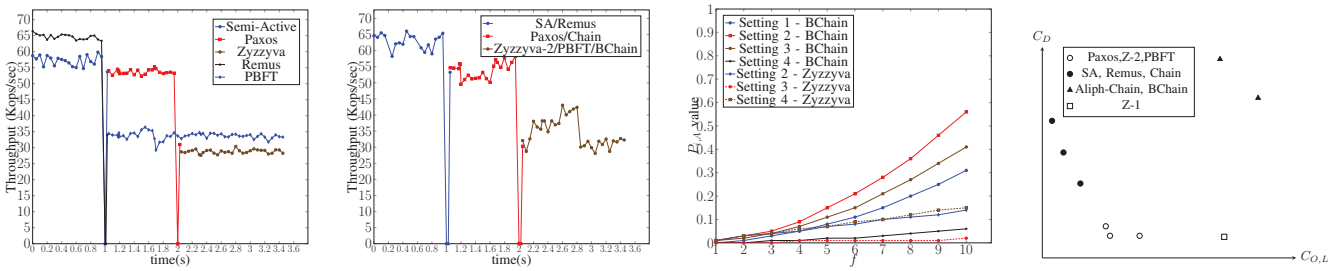
(a) $P_{0,A} = 0.9$. (b) $P_{0,C} = 0.9$. (c) $P_{1,A} = 0.9$. (d) $P_{1,C} = 0.9$.

Fig. 2. Damage cost (C_D) vs. Operational Cost - Latency ($C_{O,L}$) under setting 1 and $f = 1$. SA, Z, Z-1, Z-2 represent Semi-Active, Zyzzyva, Zyzzyva with normal run, and Zyzzyva when at least one backup node fails, respectively.



(a) $P_{0,A} = 0.9$. (b) $P_{0,C} = 0.9$. (c) $P_{1,A} = 0.9$. (d) $P_{1,C} = 0.9$.

Fig. 3. Damage cost (C_D) vs. Operational Cost -Throughput ($C_{O,T}$) under setting 1 and $f = 1$.



(a) Throughput under setting 1 with $f = 2$ and 10 clients. Protocols are not switched periodically in the same cluster.

(b) Throughput under setting 1 with $f = 2$ and 10 clients. Protocols are switched in the same cluster every 0.2 ms.

(c) Minimum requirement for $P_{0,A}$ between Paxos and BChain or Zyzzyva under different settings.

(d) C_D vs. $C_{O,L}$. Setting 1 with $f = 2$ and $P_{0,A} = P_{1,A} = P_{2,A} = P_{0,C} = P_{1,C} = P_{2,C} = 0..9$.

Fig. 4. Evaluation of the cost model and the protocol.

Selection of Protocols. We first evaluate the effectiveness of our cost model in selecting the "right" cluster of protocols, as shown in Fig. 2 and Fig. 3 under setting 1 and $f = 1$. In each experiment, the IDS reports a relatively high P value for one replica, either crash or Byzantine. Based on the figures, we notice that the protocols naturally fall into clusters. For the case where the primary is Byzantine, protocols fall into two clusters. As observed from Fig. 2(a) and Fig. 3(a), the damage cost is high for the CFT protocols and much lower for the BFT protocols. This observation correctly reflects the nature of the protocols. On the other hand, in the case where the primary crashes, the damage cost remains low since most protocols have view changes, as shown in Fig. 2(b) and Fig. 3(b). We observe similar results from the cases where a backup is faulty. Being different from the case where the primary is Byzantine, only Remus and Semi-Active (SA) have very high damage cost compared to other protocols. This can be explained by the fact that Remus and SA require only $f + 1$ replicas and the protocols are no longer safe. For other CFT protocols like Paxos, since the primary is correct, correct replicas are still consistent.

Parameter θ. We evaluate the protocols to determine an empirical value for θ, which is used to select protocols with low operational cost. We use ellipses to show the selection of the protocols in the figure. In practice, the threshold values represent ranges of cost values. As observed in Fig. 2 and Fig. 3, the threshold value can largely impact the selection of protocols. For instance, if we use a tight value, as illustrated in the small ellipses, protocols in general fall into the same category (either CFT or BFT). The only exception we notice is the case where a backup node fails. In this case, CFT protocols and BFT protocols fall into the same cluster, as shown in Fig. 3(b) and Fig. 3(d). However, the protocols are still safe since the primary is correct. The downside is the possibility that very few number of protocols are selected and the selection of protocol becomes predictable. In comparison, more protocols will be selected if we use a larger threshold. However, it is possible that "wrong" protocols will be included. In most cases for the protocols we illustrate, 2 or 3 is an appropriate number, which indicates that $\theta = [0.25, 0.375]$.

Throughput. We assess the throughput under failures. We use 10 concurrent clients and let $f = 2$ and $\sigma = \theta = 0.375$ based on setting 1. We inject a crash failure at time $t = 1s$ and a "Byzantine" failure at $t = 2s$ where the IDS reports a high probability within 0.2 ms. As illustrated in Fig. 4(a), we first do not include periodic switching among protocols and show two typical cases. In the first case, SA is run in the beginning, Paxos is used after a crash failure is injected, and Zyzzyva is selected after Byzantine failures. In the second case, Remus is first run and PBFT is used after failures, where the performance degrades suddenly. We then show in Fig. 4(b) with the same setting but protocols in the same cluster are switched every 0.2 ms. It can be observed that if protocols are switched with a tight bound on operational cost, the performance is in general consistent, where the switching of protocols generates some overhead.

Threshold Λ and IDS Error Rate. The value Λ is used to select the protocol with certain damage cost in $\mathcal{C}$ and we have shown the theoretical bound. In order to determine an appropriate Λ value, we show the minimum requirement for $P_{0,A}$ so that the damage cost of any CFT protocol is lower than the highest of the BFT protocols. This is considered the worst case where the CFT protocol might fall in the same cluster with BFT protocols. We evaluate the costs for all the settings by changing the $P_{i,A}$ values. In each setting, we compare the damage cost of Paxos with that of BChain and Zyzzyva. This is because, in general, Paxos has the lowest damage cost among CFT protocols while BChain and Zyzzyva have the highest damage cost among BFT protocols. Notice that Aliph-Chain itself may have high damage cost, but our approach filters the protocols with outstanding cost. As shown in Fig. 4(c), there exist some settings where the IDS must report a high $P_{0,A}$ value, especially when f is large. In most cases, the IDS does not need to report a P with value higher than 0.5. Based on our observation, we can handle a high IDS error rate so as for the approach to be safe.

Limitations. Our cost model has several limitations. First, it cannot be used to evaluate the case where the number of faulty nodes exceeds f. As shown in Fig. 4(d), we use setting 1 and $f = 2$. $P_{i,C}$ and $P_{i,A}$ for replica 0, 1, and 2 are 0.9. It can be observed that the damage costs for the protocols such as Paxos and PBFT are still low. This is because the cost is measured by assuming that fewer than f faulty replicas are present. Second, as we have shown previously, a high IDS error rate can be tolerated. However, performance can be degraded due to inaccurate IDS results, i.e., when IDS reports an attack while the replicas do not fail. Third, the protocols with high damage cost are removed when the IDS reports more failures. However, we do not provide a scheme to add protocols into the set. This problem can be resolved by periodically recovering the replicas and adding protocols to $\mathcal{A}$.

VIII. Conclusion

In this paper, we present a moving target consensus approach. At the core of our approach is a cost model that can be used to evaluate the damage cost and the operational cost for leader-based consensus protocols that operate in rounds. Based on real-time Intrusion Detection System signals about each replica being correct, crash, or Byzantine, the damage cost evaluates the vulnerability of the protocols while the operational cost evaluates the performance of the protocols. Our approach enables the use of a safe, fast, and unpredictable protocol according to existing system vulnerability. In addition, the cost model can also be viewed as a theoretical model to analyze the characteristics of the consensus protocols.

IX. Acknowledgments

Sisi Duan was sponsored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the Department of Energy. Yun Li and Karl Levitt were sponsored in part by the Army Research Laboratory under Cooperative Agreement Number W911NF-13-2-0045(ARL Cyber Security CRA).

References

[1] *Amazon Web Services (AWS)*. https://aws.amazon.com,.
[2] White paper: VMware high availability concepts, implementation, and best practices. Technical report, VMware, 2007.
[3] White paper: Protecting mission-critical workloads with VMware fault tolerance. Technical report, VMware, 2009.
[4] T. Benzel. The science of cyber security experimentation: the deter project. In *ACSAC*, pages 137–148, 2011.
[5] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. *Distributed Systems*. ACM Press/Addison-Wesley, 1993.
[6] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.
[7] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186, 1999.
[8] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *SOSP*, pages 189–204, 2007.
[9] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *NSDI*, pages 161–174, 2008.
[10] X. Défago and S. André. Semi-passive replication and lazy consensus. *Parallel and Distributed Computing*, 64:1380–1398, 2004.
[11] D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, 1987.
[12] S. Duan, K. N. Levitt, H. Meling, S. Peisert, and H. Zhang. ByzID: Byzantine fault tolerance from intrusion detection. In *SRDS*, pages 253–264, 2014.
[13] S. Duan, H. Meling, S. Peisert, and H. Zhang. BChain: Byzantine replication with high throughput and embedded reconfiguration. In *OPODIS*, pages 91–106, 2014.
[14] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of ACM*, 32(2):288–323, 1988.
[15] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 41–47. ACM, 2002.
[16] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The next 700 bft protocols. *ACM Transactions on Computer Systems*, 32(4):12:1–12:45, 2015.
[17] R. Guerraoui and A. Schiper. Software-based replication for fault tolerance. *Journal of Computer*, 30(4):68–74, 1997.
[18] W. House. Trustworthy cyberspace: Strategic plan for the federal cyber security research and development program. *Report of the National Science and Technology Council, Executive Office of the President*, 2011.
[19] S. Jajodia, A. K. Ghosh, V. Subrahmanian, V. Swarup, C. Wang, and X. S. Wang. Moving target defense ii. *Application of game Theory and Adversarial Modeling. Series: Advances in Information Security*, 100:203, 2013.
[20] R. Kapitza, J. Behl, C. Cachine, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel. CheapBFT: Resource-efficient Byzantine fault tolerance. In *EuroSys*, pages 295–308, 2012.

[21] I. Keidar. Challenges in evaluating distributed algorithms. In *Future directions in distributed computing*, pages 40–44. Springer, 2003.
[22] C. Ko, M. Ruschitzka, and K. N. Levitt. Execution monitoring of security-critical programs in distributed systems: a specification-based approach. In *Security and Privacy*, pages 175–187, 1997.
[23] R. Kolta, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. *ACM Transactions on Computer Systems*, 27(4):7:1–7:39, 2009.
[24] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.
[25] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1, 2):5–22, 2002.
[26] E. B. Lennon, M. Swanson, J. Sabato, J. Hash, and L. Graffo. It security metrics. *ITL Bulletin, National Institute of Standards and Technology*, 2003.
[27] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *Security and Privacy*, pages 59–66, 1988.
[28] S. Nikolaou and R. van Renesse. Turtle consensus: Moving target defense for consensus. In *Middleware*, pages 185–196, 2015.
[29] N. Poolsappasit, R. Dewri, and I. Ray. Dynamic security risk management using bayesian attack graphs. *Dependable and Secure Computing, IEEE Transactions on*, 9(1):61–74, 2012.
[30] K. Scarfone and P. Mell. Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800(2007):94, 2007.
[31] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.
[32] N. Stakhanova, S. Basu, and J. Wong. A cost-sensitive model for preemptive intrusion response systems. In *AINA*, volume 7, pages 428–435, 2007.
[33] P. Traynor, H. Choi, G. Cao, S. Zhu, and T. La Porta. Establishing pair-wise keys in heterogeneous sensor networks. In *INFOCOM*, 2006.
[34] P. Urbán, X. Défago, and A. Schiper. Contention-aware metrics for distributed algorithms: Comparison of atomic broadcast algorithms. In *Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on*, pages 582–589. IEEE, 2000.
[35] R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *OSDI*, pages 91–104, 2004.
[36] R. Zhang, S. A. DeLoach, and X. Ou. Towards a theory of moving target defense. In *MTD*, pages 31–40, 2014.
[37] P. Zieliński. Low-latency atomic broadcast in the presence of contention. In *DISC*, pages 505–519, 2006.

APPENDIX

A. Proof of Theorem 1

Proof. We first prove the following lemmas and then show the correctness of the theorem.

Lemma 4. *For all the BFT protocols, if a correct replica includes a request M in $\mathcal{O}$ with the same sequence number N, at least $2f + 1$ replicas accept M with N.*

Proof of Lemma 4: The lemma simply follows the correctness of the protocols and we ignore the details here. ■

Lemma 5. *If π' is run on the same set or a subset of replicas of π, i.e., $\mathcal{P}' \subseteq \mathcal{P}$, the switching to protocol π' is both safe and live.*

Proof of Lemma 5: If π' is run on the same set or subset of replicas $\mathcal{P}$, all the replicas maintain the same state from π. The lemma can then be proved by showing that 1) The new primary can select a set of requests based on the execution history for the switching to be live, and 2) If a correct replica has committed a request, the request will be selected by the new primary for the switching to be safe. Notice that if the primary is not correct, view changes will occur until a correct primary is selected. The correctness of view change is proved according to protocol π' and we only consider the case where a correct primary ensures the safety during the switching of protocols.

If both π and π' are CFT protocols, all the committed and uncommitted requests must be consistent since there are only crash failures. Therefore, the primary can order the requests based on the execution history and the switching of protocols is both safe and live.

Otherwise, if π is a BFT protocol, the primary is able to proceed since if fewer than $2f + 1$ replicas have committed a request, it selects null. It is also not possible where two sets of $f + 1$ replicas both include M in the $\mathcal{O}$. This can be proved by contradiction where in each set there is at least one correct replica, e.g., p_1 committed M and p_2 committed M'. In both cases, according to Lemma 4, at least $2f + 1$ replicas that have already accepted M or M'. Therefore, at least one correct replica has accepted both M and M', a contradiction. Therefore, the switching of protocols is live.

We then prove that if a correct replica has committed a request then the primary will select it. This can also be proved by contradiction assuming a correct replica p_i has committed a request M with sequence number N but the primary selects M'. If the primary assigns M' with N, there are at least $f+1$ replicas that include M' in the $\mathcal{O}$ set, among which there is at least one correct replica. Based on Lemma 4, it indicates that in π at least $2f + 1$ replicas has accepted M'. However, since p_i has committed the request M, at least $2f+1$ replicas has accepted M. Therefore, there must be at least one correct replica that has accepted both M and M', a contradiction. Therefore, the switching of protocol is safe and the lemma follows. ■

Lemma 6. *A correct latest checkpoint can be collected based on the replicas in $\mathcal{P}$ running π.*

Proof of Lemma 6: If we use new replica(s) for protocol π', each new replica obtains checkpoints from $\mathcal{P}$ and the new primary selects one with at least $f + 1$ signatures and orders the requests with a sequence number greater than the latest stable checkpoint. If π is a C-1 protocol, the replicas must send their signed checkpoint by the primary to the IDS so that IDS can transfer the checkpoint to the new replicas. Since we assume IDS is benign and can only fail by crashing, the new replicas will receive matching checkpoints and the correctness follows. Otherwise, if π is a B-1 protocol, the correctness simply follows the checkpoint scheme for BFT protocols and we ignore the details here. Lastly, if π is a C-2 protocol and there are Byzantine failures, it is possible that correct replicas have inconsistent states and checkpoints. However, the new primary is able to find a stable checkpoint if there exists a checkpoint with at least $f+1$ signatures and it is not possible that there exist two inconsistent checkpoints since there are $2f+1$ replicas. Since checkpoints from replicas can be verified by any replicas due to the use of digital signatures, all the new replicas will accept the checkpoint by the new primary. ■

Lemma 7. *If π' is run on a larger number of replicas than π, i.e., $|\mathcal{P}| < |\mathcal{P}'|$, the switching to protocol π' is both safe and live.*

Proof of Lemma 7: As we show in Lemma 6, if we use new replicas, the primary is able to select a stable checkpoint. Therefore, during the switching of protocols there are three cases for ordering requests with sequence number greater than last stable checkpoint: 1) π is a C-1 protocol and π' is a C-2 protocol, 2) π is a C-1 protocol and π' is a B-1 protocol, and 3) π is a C-2 protocol and π' is a B-1 protocol. The first case is trivial due to the fact that all the replicas are benign and the uncommitted requests are consistent. Therefore, the primary will be able to select a request for each sequence number. We then show the correctness the other two cases.

In the second case, if π is a C-1 protocol like Remus, the new primary selects null request for all the sequence numbers from l to $l+L$ since backups keep their states consistent from the checkpoints. Otherwise, replicas will include their executed requests in $\mathcal{U}$ instead of $\mathcal{O}$. The new primary only selects requests for each sequence number if uncommitted requests are matching for all the replicas. In this case, at least one correct replica has accepted the request. Therefore, the primary can select requests easily and the requests must be accepted by correct replicas. The correctness therefore follows.

In the third case, $\mathcal{O}$ includes requests where the replica collects $2f+1$ matching messages in π for C-2 protocols according to our consensus model. We first show safety that any committed requests by a correct replica will be included by the new primary. We prove by contradiction by assuming a correct replica p_i commits a request M with sequence number N and the new primary includes M' during the switching of protocols. According to our approach, if p_i includes M in $\mathcal{O}$, p_i receives matching messages for M with N from all the $2f+1$ replicas, among which at least $f+1$ of them are correct. Similarly, if the new primary in π' selects M' for N, it finds that at least $f+1$ replicas include M' for N in $\mathcal{O}$ or at least $2f+1$ replicas include M' in $\mathcal{U}$. If at least $f+1$ replicas include M' in $\mathcal{O}$, at least one correct replica includes M' for N and the correct replica receives $2f+1$ matching messages with M', among which at least $f+1$ replicas are correct. If at least $2f+1$ replicas include M' in $\mathcal{U}$, it is straightforward that at least $f+1$ correct replicas accept M'. Since there are only $2f+1$ replicas in π, there exists at least one correct replica that accepts both M and M' for N, a contradiction. Therefore, the protocol is safe.

We only need to prove liveness for the third case where the primary will be able to select a request for each sequence number. We show that it is not possible where there exists M and M' with the same sequence number, at least $f+1$ replicas include in $\mathcal{O}$ or $2f+1$ replicas include in $\mathcal{U}$. It is trivial that if $2f+1$ replicas include a request in $\mathcal{U}$, all the replicas accept the request. If at least $f+1$ replicas include a request in $\mathcal{O}$, at least one of them is correct. The correct replica must have received matching messages from $2f+1$ replicas in protocol π. Therefore, it is not possible that there exists M and M' with the same sequence number. The switching to protocol π' is live and the correctness of the lemma follows. ■

We now show the correctness of the theorem. During the switching of protocols, since we use $f+1$ replicas for C-1 protocols, $2f+1$ for C-2 protocols, or $3f+1$ for B-1 replicas, there are in total three cases: 1) the new protocol runs on the same number of replicas, 2) the new protocol runs on more replicas, and 3) the new protocol runs on fewer replicas. We have already show in Lemma 5 the first two cases are safe and live if $\mathcal{P}' \subseteq \mathcal{P}$. We also include the case where if new replicas are used in π', all the replicas will be able to use a consistent checkpoint and state in Lemma 6. Notice that if there are new replicas in $\mathcal{P}'$, replicas must be able to obtain consistent checkpoint from $\mathcal{P}$ since if π' runs on the same number or smaller number of replicas, the type of failures π' tolerates is weaker than or the same with π. Therefore, all the new replicas in $\mathcal{P}'$ can obtain consistent state. Finally, we also show in Lemma 7 the last case is safe and live. The correctness of the theorem then follows. □

B. Proof of Theorem 2

Proof. In order for our approach to be safe, we always need to guarantee that in the worst case when the primary is Byzantine, the damage cost of any CFT protocol is high enough so that it does not fall into the same cluster with other BFT protocol. Therefore, when the CFT protocol that has the lowest damage cost is greater than any BFT protocol, no CFT protocols will fall into the same cluster with BFT protocols. This requires a value of $P_{0,A}$ that is high enough regarding the threshold for selecting a cluster. We notice that Paxos has the lowest damage cost among the CFT protocols we illustrate, which has damage cost as shown in Equ. (8) and all the BFT protocols with view changes have the same pattern as follows.

$$C_D = T_V \delta(P_{0,C} + P_{0,A}) + \Omega \tag{30}$$

Ω represents the expected damage cost from the failures of backup nodes. Therefore, the following equation follows according to the selection of $\mathcal{C}$ in Algorithm 1.

$$T_V \delta P_{0,C} + \Delta P_{0,A} > T_V \delta(P_{0,C} + P_{0,A}) + min(\Omega) + \Lambda \tag{31}$$

In Equ. (31), $min(\Omega)$ represents the minimum damage cost by backups among all the BFT protocols. We then have the following:

$$(\Delta - T_V \delta) P_{0,A} > min(\Omega) + \Lambda \tag{32}$$

Therefore, Theorem 2 follows. □

C. Proof of Theorem 3

Proof. Since the threshold S is set to $\sigma|\mathcal{A}|$, in the beginning there are $\sigma|\mathcal{A}|$ replicas and the probability follows.

According to Algorithm 1, we first filter the protocols with damage cost greater than $\sigma|\mathcal{A}|$ protocols and there are $|\mathcal{A}'| = (1-\sigma)|\mathcal{A}|$ protocols. Next, we filter protocols with operational cost greater than $\theta|\mathcal{A}'|$ protocols. Therefore, there are $(1-\theta)(1-\sigma)|\mathcal{A}|$ protocols in $\mathcal{R}$. If we assume a large enough Λ value, set $\mathcal{C}$ has $(1-\theta)(1-\sigma)|\mathcal{A}|$ protocols and we switch among them, the theorem then follows. □

Peripheral Authentication for Autonomous Vehicles

Shlomi Dolev and Nisha Panwar
Ben-Gurion University of the Negev, Israel. {dolev,panwar}@cs.bgu.ac.il

***Abstract*—We propose a peripheral authentication scheme for autonomous vehicles. A mutual authentication protocol is required to secure every peripheral device access to a vehicle. Specifically, we present a vehicle to peripheral device authentication scheme. In addition, our three way handshake scheme for vehicle to keyfob authentication scheme based on generalized peripheral authentication scheme has been proposed. The vehicle to keyfob authentication scheme is adapted and improved with an additional attribute verification of the keyfob holder. Conventionally, vehicle to keyfob authentication is realized through a challenge-response verification protocol. An authentic coupling between the vehicle identity and the keyfob avoids any illegal access to the vehicle. However, these authentication messages can be relayed by an active adversary, thereby, can amplify the actual distance between the authentic vehicle and the keyfob. Eventually, through this malicious relaying an adversary can possibly get access to the vehicle, without any effort to generate or decode the crypto credentials. Our solution is a two party, three way handshake scheme with proactive and reactive commitment verification. Conceptually, our solution is different than the distance bounding protocols that requires multiple rounds of round trip delay measurement.**

Keywords*—*Authentication, access control, event data recorders.

I. Introduction

Currently, vehicles are customized to be a secure mobile information system [1]. The rapidly moving vehicles are compliant with the Dedicated Short Range Communication (DSRC) IEEE 1609 [2] based on Wireless Access in Vehicular Environment (WAVE) 802.11p [3]. However, another crucial aspect is to authorize the access to a vehicle via peripheral device connections. In general, vehicle internal networks are supposed to provide a secure identifying gateway to these external devices. However, our approach verifies any transient peripheral device integration via three way handshake that promises a secure authentication. In addition, derive all subsequent messages with the initial round authentication associated to initial prover.

We propose a secure mutual pairing between the vehicle and peripheral device [4]. The solution avoid any unauthorized access and, thereby, a consequent privilege to start the engine of a parked vehicle. Our motivation is to strengthen an access control over a static/parked vehicle such that an owner must be authenticated based on pre-defined Challenge-Response Pairs (CRP), shared commitments, actively measured dynamics, and attribution of human/owners characteristics.

Peripheral authentication. A secure digital periphery of the vehicle is achieved via secure authentication with respect to paired devices. Specifically, any temporary peripheral device connection with the vehicle must be authenticated for the extended functional security of the vehicle (ISO 2626 vehicle functional security standard). These peripheral devices as a keyfob, USB stick, cell phone, and, ipod provide extended services to the vehicle. Evidently, this is a potential exposure for the external threats to break-in an otherwise secure vehicle periphery. Our motivation is to secure the peripheral device integration, specifically, remote vehicle access via keyfob. In particular, the problem is beyond the effort to place a secure firewall for filtering any external threats due to a range of relay and impersonation attacks. A secure remote access is most crucial among other peripheral device connections because vehicle access via keyfob has a wider horizon to attack. Therefore, it is important to identify and authenticate the correct keyfob (continuously approaching towards the parked vehicle) via active locomotion pattern of the keyfob (more appropriately keyfob holder).

978-1-5090-3216-7/16/$31.00

Problem statement. The problem is to avoid an unauthorized remote vehicle access via fabricated Radio Frequency Identification (RFID) enabled keyfob. Also to provide an anthropomorphic link to the bonding between the vehicle and peripheral device such as RFID enabled keyfob. Conventionally, keyless entry systems provide an autonomous * sensing. Such that the parked vehicle keeps on sensing the presence of authentic keyfob in the proximity, e.g. via regular beacon solicitation. The authentic keyfob must be present in the proximity and respond back to these soliciting beacons from the parked vehicle. However, the absence of the authentic keyfob within the sensed region can be amplified with another RFID enabled keyfob. The malicious keyfob would create an illusion of the shorter distance by amplifying and relaying the signals between both parties. In addition, these RFID signals are vulnerable to other more sophisticated attacks [6] in an adaptive adversary model. In particular, an adversary recover a exhaustive number of CRP transcripts and based on that knowledge might fabricate a duplicate keyfob.

Design requirements. An authentication protocol construction must incorporate the verification of a pre-shared secret, an active response and a specific anthropomorphic feature, e.g., personalized locomotion pattern of keyfob holder. In particular, our design involves following factors and synergize it into a multi-dimensionally secure access control scheme. Essentially, design requirements can be summarized as:

Reciprocal authentication: A primary requirement is to provide a mutual authentication between vehicle and peripheral device, i.e., keyfob. In general, the vehicle to keyfob pairing is visualized from vehicle's perspective and keyfob as a responder. However, the vehicle as an initiator is more vulnerable to attack exposure as compared to the other way around. In our scheme, the keyfob is initiator and vehicle is responder to validate a specific service access grantee. i.e., authentic keyfob. The vehicle acts as a responder to authenticate the initiator and to reciprocate the secret challenge

*Note that the system settings are defined within the scope of autonomous vehicles, i.e., availability of IEEE 1609.2 [2], IEEE 802.11p [3], and, Black Box IEEE 1616 [5]

with vehicle identity towards initiator.

Identification based on pre-determined state: In our solution, initial pairing is secured using an internal state record of the vehicle which is pro-actively synchronized with the internal state record of the keyfob. The initial pairing must witness a matching internal state (inside the vehicle and authentic keyfob) as a part of pre-determined knowledge verification.

Reactive verification: CRP based reactive verification avoid misbinding attacks and satisfy at least a non-injective authentication property. The pro-active commitment (based on internal vehicle state) must be coupled with a reactive commitment verification. This handshake ordering would avoid an attack scenario in which the adversary might respond with any random response for an authentic challenge. Thus the vehicle must be able to verify the validity of response, i.e. response should be in correspondence with the current challenge.

Anthropomorphic features: The personification of any unique behavior or attribute of the keyfob owner can be verified during handshake. In particular, we propose to verify the locomotion with respect to keyfob holder. Also this locomotion pattern would become somewhat obvious and distinguishable over a period of time, i.e., any locomotion information collected over multiple authentication phases (between a specific vehicle and paired keyfob) would result into personification of this locomotion pattern (with respect to a specific keyfob holder). The human attribution of the owner's gait and active verification of the corresponding locomotion pattern is sort of customized way of authentication. In particular, this customization provides more intuitive authentication to same vehicle and keyfob over different sessions.

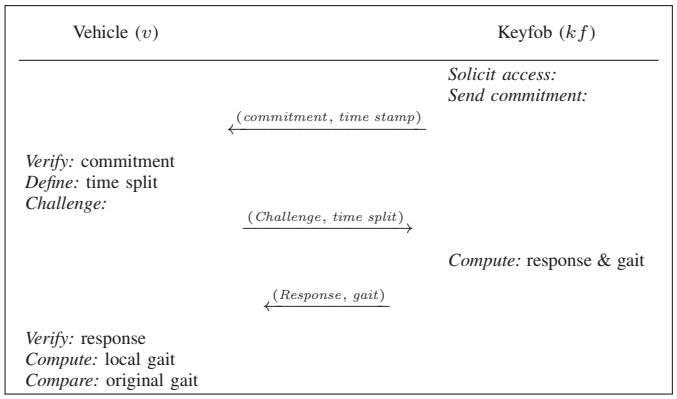


Fig. 1: The proposed approach.

Our contribution. We provide a solution (see Figure 1) with:

(*i*) *Knowledge based initial pairing:* A most recent internal state of any vehicle is known only to the keyfob in current use. Therefore, every time a vehicle changes its internal state, the corresponding commitment data is also changed. In particular, only a key that was used during the last drive would be in possession of the correct internal state of the vehicle.

(*ii*) *Commitment based authentication:* A reactively changing commitment data between the vehicle and authentic keyfob provides spontaneous identity verification. It provides a spontaneous verification of the peer party in communication during the last rounds of handshake.

(*iii*) *Personified localization:* An instant gait verification is done with respect to a keyfob holder approaching towards paired vehicle. Evidently, a direct communication between the paired vehicle and corresponding keyfob would yield the same gait observation, i.e., similar distance covered in similar time window; as opposed to an attack scenario in which the adversary relay the wireless radio messages. In particular, no additional hardware integration is required within the internal vehicle network. In particular, gait observation at the static vehicle would differ with the original gait observation at the keyfob holder, if an active relaying is involved during the wireless radio communication.

Events	**Recorded data IEEE 1616 [5]**
Location	(x,y) coordinates
Time stamps	paired with (x,y)
Deceleration	velocity decrease ratio
Acceleration	velocity increase ratio
Yawing	steering angle
Seat position	passenger posture
Airbag deployment	activation time
Unusual events	breaking above threshold

Table I: Event triggered data.

IEEE 1616 Event Data Recorders (EDRs). EDRs are used to maintain event primitives in a log. The event observation and recording process is defined by the standard IEEE 1616a. The event primitives such as acceleration, deceleration, steering angle/movement, velocity, and seat position, accounts for the driving decisions taken during the crash incident (see Table I). These essential event factors contribute to improve the consequent safety events [5] in future. Subsequently, forensic investigation against a crash event extracts and links these event records of the vehicle. However, the privileges regarding the access over this critical/confidential information of any vehicle depends on the state law. It may further be of utmost importance to forensic team investigation. In past, consumers were not aware that an Event Data Recorder (EDR) is integrated inside the vehicle. However, the current state laws, i.e. Black Box Privacy Protection act 2013 [7] made the EDR ownership clear to vehicle owner and that it cannot be accessed without the consent from the vehicle owner. (1) the presence and location of an EDR (also termed as a black box), (2) refining the critical event information and storage format (3) usage and claims to acquire the recorded internal state data for legal proceedings with owner's consent. We emphasize that the crucial event record stored on a volatile memory inside the Electronic Control Units (ECUs) can be used to authorize the original vehicle owner to the vehicle.

Definition 1 *EDR mobility pattern: A recent few seconds of the vehicle dynamics during the last itinerary available from a non-volatile EDR storage defines the vehicle mobility pattern. A static vehicle and the corresponding keyfob share this mobility trace as a common internal state of the vehicle.*

Previous work. Remote keyless entry via transponder integrated with the ignition key or using immobilizer, has been used actively [8]. A solution based on a distance bounding protocol and a verifiable multi-lateration scheme has been considered in [8]. An immobilizer is used to avoid the vehicle movement even if an unauthorized access is gained. The immobilizer based physical lock is useful to secure the physical periphery but not the digital periphery. In [9] a

coalition attack scenario has been solved, however, it differ in terms of access control countermeasures required for a parked vehicle scenario. Therefore, our solution eliminates the need of specialized Physically Unclonable Function (PUF) integration to the vehicle and the keyfob. Also a gait based authentication scheme is given in [10], [11] for identifying that the same user is operating over two devices simultaneously.

II. Adversary model

We present three different (but related) relay and impersonation attack scenarios. These attack scenarios are applicable to various systems based on service access verification. Specially, the services that are accessible over a wireless radio channel within a closer proximity. Therefore, the vehicle to peripheral device, e.g., keyfob, access scenarios are equally vulnerable to these scenarios.

Distance fraud. There are various services that are meant to verify presence of user in locality before granting access to the resources. According to the distance fraud an authentic prover claims to be at certain distance (thereby, legal to access the services) while actually being far away from the claimed distance with respect to the verifier. The adversary pretends to impersonate the original sender as a man-in-the-middle attack. For example, suppose that a vehicle and keyfob are situated apart (might be in the range of each other). An adversary might just amplify and relay these signals from the authentic keyfob pretending to acquire an access. In this case, an adversary would receive at leat an early access, in case the original keyfob has actually requested for vehicle access. Therefore, both the proactive and reactive secret verification (with additional locomotion verification for the keyfob authentication) are crucial to the proposed authentication scheme. The authentication protocols that satisfy only *aliveness* property are usually prone to distance fraud. Therefore, the protocol must satisfy, at least, a *weak agreement* property to avoid distance fraud attacks.

Mafia fraud. This attack is another more sophisticated form of distance attack. In this attack scenario two adversaries collude and get illegitimate access to the secure services, while the original sender has no intention to request an access or to reveal cryptographic secret. According to the mafia fraud an adversary tries to utilize a separate channel and an accomplice to extract and relay the credentials from an authentic prover (not actively interested at all, in any service access). Therefore, adversaries collude and relay the secure access code to break in the verifier (meant to provide the service access to authentic secret holder). The authentication protocols that satisfy only, *aliveness* and *weak agreement* properties are usually prone to mafia fraud. Therefore, the protocol must satisfy, at least, a *non-injective agreement* property to avoid mafia fraud attacks.

Terrorist fraud. According to the terrorist fraud an authentic prover assists with the adversary (by handing over secret component) to impersonate in front of the verifier. The attack scenario is practically feasible even with the biometric authentication because the original secret holder can authorize himself, and, let the service be accessible to other who does not possess secret biometric credentials. A secure protocol design requires more sophisticated identity verification method such as a ticket granting authority (issuing tickets for service access). It must be noticed that we are solution is not resistant to this type of impersonation attacks. In general, authentication protocols that satisfy the most concrete form of authentication, i.e., *injective agreement* property might still be prone to these attacks. A most resembling example is when any vehicle (that is secured under insurance policy of the owner) is stolen by a thief, if only, had the owner *subliminally* assisted to thief.

III. Proposed scheme

We propose an authentication scheme (Figure 2) for a remote vehicle access control. This access control can be perceived as an authorization check for services within the periphery of a vehicle. For example, we propose an initial pairing based on the event log replication. The event log is overwritten with every new event occurrence and keeps recording only the recent few seconds of vehicle dynamics. Concurrently, vehicle event log can be replicated over the peripheral device. Therefore, a peripheral device in possession of most recent event log is authorized as previous occupant.

Pre-processing phase: A pre-processing phase and the subsequent usage of cryptographic primitives is given as below:

Setup phase (Key generation): The manufacturing authority initializes $Init(Auth)$ the security module with a secret symmetric key K (for both vehicle and keyfob) before even handing it over to the consumer.

Registration phase (Symmetric key with user identity): The next phase is to provide a vehicle to keyfob binding at the time of handing it over to a specific consumer u. Accordingly, the registration phase is required to associate a pre-initialized symmetric key K in setup phase $Init(Auth)$ with the keyfob and the user, i.e., (K, kf, v, u) binding user u with vehicle v, keyfob kf and symmetric key K. It must be noticed that the user identity u is crucial for the initial binding such as creating an administration account. Initially, event records are null and does not provide a linkage between any keyfob holder, as in the past and in the current.

Query phase (Attempt to attack): In the query phase, adversary $\mathcal{A}$ utilizes the knowledge of symmetric key K (from the n number of transcripts t_n extracted during n sessions in past) and perform the following sequence of message exchange:

- $\mathcal{P}$ sends (K, kf, v, u) requesting for access permission.
- $\mathcal{A}$ retrieves (K, kf, v, u) and relay (K, kf_{adv}, v, u).
- $\mathcal{V}$ verifies (K, kf_{adv}, v, u) before granting access permission, i.e., $Check \leftarrow (K, kf_{adv}, v, u)$.

First, we present a simple authentication approach for peripheral devices, e.g., cell phone, USB stick, ipod, laptop, and other bluetooth devices.

Peripheral authentication scheme. The digital periphery (meaning physical as well as wireless signal periphery) of a vehicle must utilize reactive and proactive commitment verification towards an access grantee. In general, a vehicle is supposed to receive a request for safe pairing and a subsequent access to various internal vehicle modules, e.g., Anti-lock Braking System (ABS), Powertrain Control Module (PCM), Engine Control Unit (ECU), Transmission Control Unit (TCU), Tire Pressure Monitoring (TPM), Active Control Module (ACM), Relay Control Module (RCM), Heat Ventilation and Air Condition System (HVAC). Evidently, the security of these modules is related to the secure pairing with peripheral devices. However, the secure pairing is even more crucial when a peripheral device (e.g., keyfob) requests a remote access to the vehicle.

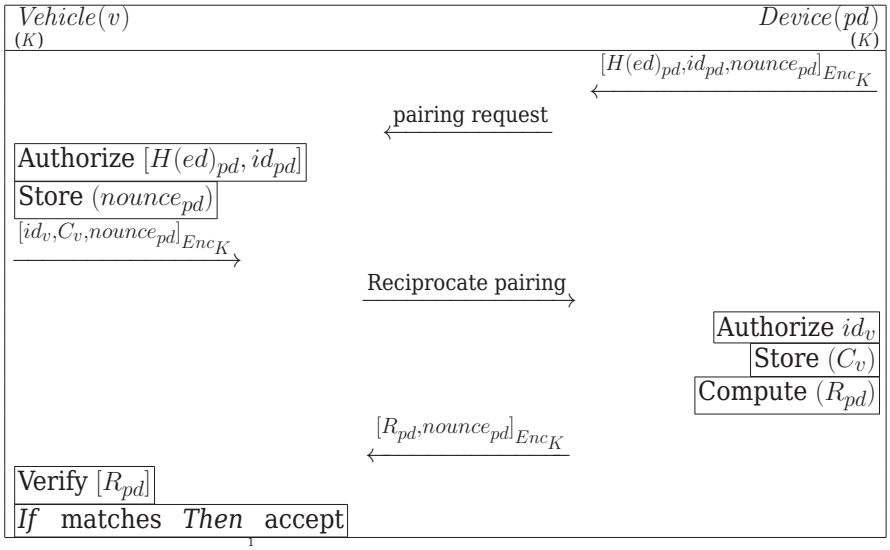


Fig. 2: Peripheral authentication scheme.

As detailed in Figure 2, a vehicle receive the pairing request from a peripheral device. The request begins with encrypted identity information from a specific peripheral device pd. For example, the shared internal state ed, identity pd_{id}, and sequence number $nounce$. The event information is securely hashed in an abstract form. The vehicle and device share a symmetric key K (as detailed in setup and registration phase) and all handshake messages are encrypted with K. Next, vehicle verify the pairing request as: (*i*) Is the requesting device has similar internal state as $H(ed)_v = H(ed)_{pd}$ (i.e., the device has been used during most recent drive in past)? (*ii*) Is the requesting device has an authentic identity as pd_{id}? (*iii*) Is the requesting device has a unique nounce as $nounce_{pd}$? In the second step, after the successful verification of initial commitment, vehicle would send a challenge C_v with vehicle identity id_v and $nounce_{pd}$. The device authenticates the reciprocated values, i.e., id_v and paired $nounce_{pd}$. Consequently, the device would produce and send the corresponding response R_{pd} with $nounce_{pd}$. The vehicle verifies the reactive response and grants the access, if authentication succeeds.

Next we present an authentication game to define the adversary advantage over the proposed scheme $Auth$.

Definition 2 *The security game for the proposed authentication scheme $Auth$, adversary $\mathcal{A}$, prover $\mathcal{P}$ and an authentic verifier $\mathcal{V}$ is given as below:*

$\mathcal{A}$ wins the game if $[Check = 1]$. The probabilistic advantage of adversary, $Adv(\mathcal{A})$, for winning the game is

$$Adv(\mathcal{A}) = Pr[Check = 1]$$

Specifically, vehicle to keyfob authentication scheme $Auth$ is secure if the $Adv(\mathcal{A})$ is negligible. In addition, $Pr[Check = 1]$ is maximum during the initial rounds of pairing when event log is almost null. Therefore, a user identity is used for initial pairing as long as the event data is not populated enough.

IV. Future work

We aim to provide a three way handshake based vehicle to keyfob authentication scheme that incorporates authentication based on human attribution. In addition, the potential usage of one time password schemes, e.g., HMAC based One Time Password (HOTP) and Time based One Time Password (TOTP). As a part of open problems, our solution based on commitment verification and machine learning is not designed to cope with replay attacks. Furthermore, the solution does not cope with adversary that might have tuned the relaying delay, mimicking the change in distance over time in a malicious fashion.

References

[1] K. Dellios, D. Papanikas and D. Polemi. Information Security Compliance over Intelligent Transport Systems: Is IT Possible?. *IEEE Security Privacy*, 13(3), pp 9-15, 2015.

[2] Dedicated Short Range Communications (DSRC) Concept of Operations and ISO Layer Implementation Summary available at URL: http://www.its.dot.gov/factsheets/dsrc_factsheet.htm

[3] R. Uzcategui and G. Acosta-Marum. Wave: A tutorial. *Communications Magazine, IEEE*, 47(5):126–133, 2009.

[4] A. Kumar, N. Saxena, G. Tsudik, and E. Uzun. A comparative study of secure device pairing methods. In *Pervasive and Mobile Computing*, 5(6), pp 734-749, 2009.

[5] IEEE Standard for Motor Vehicle Event Data Recorders (MVEDRs). *IEEE Std 1616-2004*, pp 1-163, 2005.

[6] W. Aerts, E. Biham, D. De Moiti, E. De Mulder, O. Dunkelman, S. Indesteege, N. Keller, B. Preneel, G. Vandenbosch, and I. Verbauwhede. A Practical Attack on KeeLoq. *Journal of Cryptology*, 25(1), pp 136-157, 2012.

[7] Black Box Privacy Protection Act available at URL: https://www.govtrack.us/congress/bills/113/hr2414

[8] C. Patsakis, K. Dellios, and M. Bouroche. Towards a distributed secure in-vehicle communication architecture for modern vehicles. *Computers and Security*, 2014.

[9] S. Dolev, Ł. Krzywiecki, N. Panwar, and M. Segal. Optical puf for vehicles non-forwardable authentication. *Computer Communications*, 93, pp 52-67, 2016.

[10] R. Mayrhofer and H. Gellersen. Shake Well Before Use: Intuitive and Secure Pairing of Mobile Devices. In *IEEE Transactions on Mobile Computing*, 8(6), pp 792-806, 2009.

[11] T.D. Gray, S. Valiyani, and V. Polotski. Gait-based authentication system, WO Patent App. PCT/CA2010/001,002, 2011.

Efficient Transmission Strategy Selection Algorithm for M2M Communications: An Evolutionary Game Approach

Safa Hamdoun*, Abderrezak Rachedi*, Hamidou Tembine†, Yacine Ghamri-Doudane‡

*Paris-Est University † New York University ‡University of La Rochelle

{hamdoun, rachedi}@u-pem.fr {tembine}@nyu.edu {yacine.ghamri}@univ-lr.fr

***Abstract*—*Device-to-device* (D2D) communications, one of the major component of the evolving 5G networks, is showing promising advantages on supporting *machine-to-machine* (M2M) communications. In this paper, we consider the design of efficient transmission strategy selection algorithm for M2M communications underlaying cellular networks. First, a group of *machine-type-devices* (MTDs) is matched with a particular *user equipment* (UE). MTDs belonging to the same group can access the same spectrum within its matched UE while the latter *quality of service* (QoS) is maintained. Next, we propose an efficient evolutionary game based transmission strategy selection algorithm for M2M communications using D2D mode. Specifically, MTDs switch opportunistically from a non-cooperative strategy to a cooperative strategy. Initially, we consider a non-cooperative scenario due to the selfish behavior of devices. In case the latter QoS is not satisfied, MTDs switch to a cooperative game. In a cooperative game, we propose two alternative power control schemes: a fixed mixed-strategy power control scheme where each MTD willing to play cooperatively selects the power strategy from a discrete level of powers and an adaptive mixed-strategy power control scheme. The latter technique enables to set efficiently the discrete power levels using a fuzzy logic and a *proportional-integral-derivative* (PID) controllers aiming to assure the desired QoS of UEs while maximizing the efficiency of M2M communications. Simulation results show that the evolutionary game based transmission strategy selection algorithm avoids significant degradation of traditional *human-to-human* (H2H) services in terms of throughput and fairness compared to a single non-cooperative game strategy. Besides, the adaptive mixed-strategy power control scheme outperforms the fixed mixed-strategy power control scheme by saving the battery life of MTDs while guaranteeing the latter QoS.**

***Index Terms*—M2M communications, D2D communications, QoS, PID controller, Fuzzy logic, evolutionary game.**

I. Introduction

Machine-to-machine (M2M) communications is a new paradigm that refers to the autonomous communications involving a myriad of machines, that interact among themselves without or with limited human intervention. The ubiquitous connection of devices favors the emergence of a vast range of intelligent M2M applications ranging from e-health, smart grids, smart homes as well as intelligent transportation systems, enabling partially the internet of things (IoT) [1], [2].

Cellular wireless technologies have been considered a potential candidate to support M2M communications for its ubiquitous coverage, good support of user mobility as well as high data rates. Consequently, the third generation partnership project (3GPP) has standardized M2M as *machine-type-communication* (MTC) in long term evolution and its advancements (LTE-A). 3GPP has been focusing in release 10 and beyond on air interface improvements to counter the potential problems posed by MTC on their cellular networks optimally designed for *human-to-human* (H2H) communications [3].

Contrarily to traditional H2H applications, M2M systems are characterized by a massive number of deployed devices with specific features such as: time-tolerance, small data transmission, extra low power consumption and centralized data collection. Hence, the associated signaling load and complexity of traditional LTE schedulers being designed to carry high data rates for broadband applications make existing LTE uplink schedulers prohibitive to cater to M2M requirements. Besides, the network efficiency and scalability can be drastically affected by the large number of active M2M devices, which can lead to the access and core network congestion due to the signaling overhead.

Along with the MTC new paradigm, 3GPP has introduced a new technology called *device-to-device* (D2D) communications [4]. D2D refers to the direct communication between devices without traversing the base station (BS) in cellular networks. Eventhough the spectrum efficiency is the major advantage of D2D communications, designing new resource allocation methods to mitigate the co-channel interference remains the key issue to solve.

In this paper, we study the resource sharing problem for M2M communications using D2D mode, where multiple D2D pairs can use the same sub-channel. We design an efficient evolutionary game based transmission strategy selection algorithm for M2M communications underlaying cellular networks. Specifically, *machine-type-devices* (MTDs) switch opportunistically from a non-cooperative strategy to a cooperative strategy, where the key issue is to control the interference introduced to H2H users while saving the battery life of M2M devices. The major contributions of this paper can be summarized as follows:

- We design an efficient evolutionary game based transmission strategy selection algorithm. We define a preference order for the transmission strategy of M2M devices where

978-1-5090-3216-7/16$31.00

we consider initially a non-cooperative scenario assuming that M2M devices are selfish as they are in the practical scenarios. Then, M2M devices whose QoS is not satisfied switch to a cooperative scenario.
- We propose two alternative power control schemes in the cooperative scenario; a fixed and an adaptive mixed- strategy power control schemes. While the former consider a fixed discrete strategy space, the latter approach enables to set properly the power levels using a fuzzy logic and a proportional-integral- derivative (PID) controllers in order to assure the desired QoS of H2H users while maximizing the efficiency of M2M communications.
- We assess the impact of the proposed transmission strategy selection algorithm on H2H services in terms of throughput and fairness in an H2H/M2M coexistence scenario and evaluate the M2M transmit power consumption.

The remainder of this paper is organized as follows. In Section II, we present some existing approaches for M2M scheduling. In Section III, an overview of two major controllers in the control field, namely PID and fuzzy logic is introduced. In Section IV, we describe the specific technical details of the scenario under evaluation. In Section V, the evolutionary game based transmission strategy selection algorithm for M2M communications is formulated. Two novel mixed strategy power control schemes are developed in Section VI. The performance evaluation of our proposal is drawn in section VII followed by the conclusion in Section VIII.

II. Related works

Different from the traditional human-centric communications where most of the research efforts is in the downlink, MTC applications are usually uplink-centric. Consequently, uplink (UL) scheduling for M2M communications in LTE networks and beyond has become particularly a challenging issue to solve. Numerous works that have focused on uplink scheduling aim to support traditional H2H communications [5], [6], [7]. Specifically, LTE is designed to carry high data rates for broadband application. However, the small amount of data to forward for MTDs, in addition to the huge number of devices, render the associated signaling load with existing UL schedulers prohibitive to cater to M2M scenarios. Therefore, changing existing approaches as well as designing new protocols in order to offload the signaling overhead becomes crucial.

In [8], authors have proposed two fully dynamic M2M scheduling algorithms for LTE uplink based on a delay tolerance objective and channel conditions. While higher priority has been given to UEs, the remaining resources have been assigned to MTDs. In the first scheduler, radio resources have been ranked based on the channel quality. Meanwhile, in the second scheduler, MTDs have been ranked first based on their delay tolerance, then the least delay tolerant machines have been assigned the best resources in terms of channel quality. In [9], authors have developed a mixed scheduler for H2H and M2M communications. Here, the devices have been classified into two queues. H2H users and some delay-sensitive MTDs have been included in the high-priority queue, while the remaining MTDs have been grouped in the low-priority queue. Then different scheduling algorithms have been applied for each queue. The major drawback of these algorithms consists of the starvation problem for M2M devices in case of a heavy H2H traffic scenario. Moreover, the quite good performance achieved comes at the expense of a huge signaling load because of the centralized scheme consisting of sending reports and receiving allocation decisions from the *evolved NodeB* (eNB) individually per MTD.

In [10], a distributed channel sharing algorithm based on a game theoretic approach has been designed for massive access management in an H2H/M2M coexistence scenario with the aim of maximising the sum weighted data-rate of all devices. A group of MTDs has been matched with a particular UE such that MTDs in each group can access the sub-channel of their matched UE by using TDMA scheme and send their data. However, they do no considered the UE QoS guarantees.

In [11], authors have formulated the resource sharing problem between M2M and H2H communications as an interference aware bipartite graph, then have proposed a two phase radio resource allocation approach. In the first phase, H2H users have been assigned radio resource using conventional H2H schedulers. In the second phase, authors proposed an M2M radio resource sharing algorithm. A power control scheme for the concurrently transmitting M2M nodes using D2D technology has been developed to mitigate the H2H service degradation following a probability that is set based on a Proportional-integral-derivative (PID) controller. The main drawback is the associated signaling load due to the centralized approach.

In this paper, we investigate the resource sharing problem for M2M communications using D2D mode. Unlike most of the existing work which consider that a sub-channel is reused by no more than one D2D pair, we consider a multi-resource sharing case where a sub-channel can be reused by multiple D2D pairs. We establish a preference order for the transmission strategy of M2M devices. Hence, we consider initially a non-cooperative scenario due to the selfishness of M2M devices. Then, M2M devices whose QoS is not satisfied switch to a cooperative scenario. We also consider the cost of information exchange. To the best of the author's knowledge, our work is pioneer in dealing with efficient mode selection strategy using an evolutionary game approach with the aim to assure the desired QoS of H2H users while maximizing the efficiency of M2M devices.

III. Overview of Fuzzy Logic and PID controllers

The PID and fuzzy logic are two controllers that are widely used in the control field in order to solve many problems in various applications areas such as controlling the congestion problem, leveling the security services in wireless sensor networks and managing the radio resources [12], [13].

A. PID Controllers

The PID controller is the most common feedback controller that stands for proportional-integral-derivative. The basic concept of a feedback controller is to keep a measured process variable close to a desired value despite of the variation of the process dynamics. Fig. 1 illustrates the PID feedback control system. The error signal $e(k)$ is the input of the PID controller that represents the difference between the measured process variable Q_k and a reference value Q_{ref}. The PID controller in turns gives as output the control variable u_k. A PID controller has three types of control actions:

- **Proportional to the error (*P* part)**: The P part is proportional to the current error. It reacts immediately to the sensed error.
- **Proportional to the integral of the error (*I* part)**: The integral controller integrates the history of the error.
- **Proportional to the derivative of the error (*D* part)**: The derivative controller tries to predict the immediate future and makes corrections based on the estimated error.

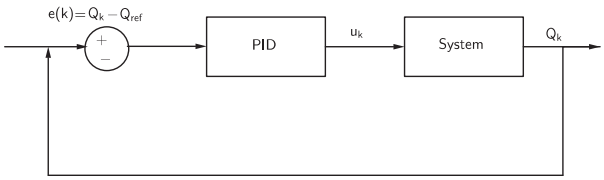


Fig. 1: The PID-controller based system

These three parameters can be used separately or in a combination [14].

B. Fuzzy Logic

Fuzzy logic is another mathematical tool in the control field that exploits a linguistic model of the process to be controlled [15]. The fuzzy theory deals with imprecision and is easier to prototype compared to the PID controller. Indeed, the latter applies a mathematical tool to generate a specific output given a quantitative and precise data value. Therefore, fuzzy logic is more appropriate when a mathematical model of the process cannot be defined or it is too complex to be evaluated in real time. The fuzzy logic decision making process is composed of three consecutive steps as illustrated in Fig. 2:

- **Fuzzification:** The input variables are fuzzified using predefined *membership functions* (MBFs). MBFs are a set of fuzzy regions that define the control variables in the fuzzy model. Unique names known as labels are given to these regions, within the domain of the variable. The MBFs: $X \rightarrow \{0,1\}$ assigns every control variable, x $\in$ X numbers from 0 and 1 unlike in the binary logic where only a value from two-element set $\{0, 1\}$ is assigned. That's why it is called fuzzification.
- **Fuzzy inference system:** Fuzzy numbers or input variables are fed into a predefined fuzzy control rules which tie the input values to the output model properties and are written with a IF-THEN clauses syntax.
- **Defuzzification:** The output of the fuzzy set is converted into a crisp value. The defuzzification can be performed using several methods. The most popular method is the centroid, where the center of area of the fuzzy set is determined and the value at which this occurs is used as the defuzzified output.

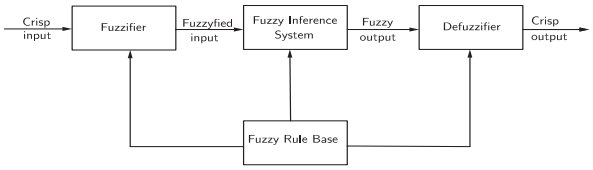


Fig. 2: A fuzzy system

We are motivated to use both controllers for determining the power level strategies of MTDs to efficiently avoid the interference situations in an H2H/M2M coexistence scenario.

IV. System Model and Assumptions

We consider the uplink scenario of a single cellular network where K channels have been allocated in an orthogonal manner to N traditional cellular users. At the same time, M MTD pairs attempt to share the uplink radio resources using D2D communications. Each MTD pair M_j with $j = 1, 2, ..., M$ is compromised of a MTD transmitter, $M_{j,t}$, and a MTD receiver, $M_{j,r}$, as shown in Fig. 3 and has a rate requirement of $R^{min}_{M_j}$. Each cellular link U_i with $i = 1, 2, ..., N$ is a connection between a UE and the eNB (i.e., the base station) and has a rate requirement of $R^{min}_{U_i}$. We use $\mathbb{N} = \{1, 2, ..., N\}$ to denote the set of indices of cellular links, $\mathbb{M} = \{1, 2, ..., M\}$ to denote the set of indices of M2M links and $\mathbb{K} = \{1, 2, ..., K\}$ to denote the set of indices of radio resources.

In LTE networks, radio resources are distributed in the time-frequency domain every transmission time interval (TTI) which consists of one subframe and has a duration of one ms. In the frequency domain, the available bandwidth is divided into a number of sub-channels. Each sub-channel has a bandwidth of 180 Khz and along with 7 symbols in the time domain constitutes a resource block (RB). The latter is the minimum unit of the resource allocation process [16].

A set of orthogonal sub-channels are allocated to each UE U_i with $i = 1, 2, ..., N$ using conventional schedulers (such as proportional fairness (PF) or round robin (RR)) optimally designed for H2H users. We assume that each sub-channel k is allocated to only one UE, implying that there will be no interference observed from the UEs at the eNB. The sub-channels allocated to the UEs are fixed for each transmission frame. During each transmission frame, a set of MTDs is matched to each UE based on cellular rate requirements and the channel information collected by the eNB from the cellular and M2M links, thus forming a virtual cluster. The latter construction is updated each transmission frame. To construct virtual clusters for each cellular link represented by circles in Fig. 3, we require the definition below:

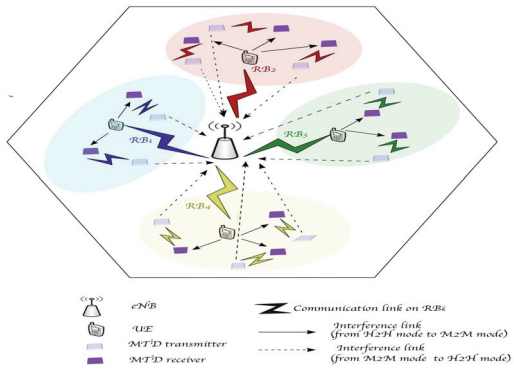


Fig. 3: System model under study: inter-MTD within D2D underlaying cellular network

Definition 1: The interference value of a cluster C_i on RB_k, denoted by $v_I^k(C_i)$, is defined as the sum of the interference introduced by L M2M pairs if sharing the radio resource RB_k with a UE, U_i. Thus, the interference value $v_I^k(C_i)$ can be given as:

$$v_I^k(C_i) = \sum_{j=1}^{L} P_j^k g_{M_{j,t},eNB}^k, j \in \mathbb{M} \quad (1)$$

where P_j is the MTD transmit power.

The basic idea of the virtual cluster construction is to match iteratively MTDs pairs to a UE where both access the same sub-channel taking into account the interference value of the cluster in order to guarantee that the UE's QoS is always satisfied. Thus, a virtual cluster is composed with L MTDs whose total introduced interference if sharing the RB_k with the UE U_i, is equal or below to a given UE interference threshold I_{th}^k, $v_I^k(C_i) \leq I_{th}^k$.

Our objective is to design an efficient evolutionary game based transmission strategy selection algorithm for M2M communications using D2D mode to satisfy the requirements of all nodes. Hence, to guarantee the QoS of the UE U_i, the achievable throughput for the UE should be larger than a threshold as defined as follows:

$$R_{U_i}^k \geq R_{U_i}^{min}, \forall i \in \mathbb{N} \quad (2)$$

The achievable throughput of UE, U_i with $i \in \mathbb{N}$, on RB_k can be expressed as

$$R_{U_i}^k = \beta.log_2(1 + \frac{P_i g_{U_i}^k}{\sum_{j \in C_i} P_j^k g_{M_{j,t},eNB}^k + \sigma^2}) \quad (3)$$

where P_i is the UE transmit power, g_{U_i} is the channel gain of the UE from U_i to eNB while $g_{M_{j,t},eNB}$ is the interference link from the MTD transmitter to the eNB. The first term in the denominator in Eq. 3 represents the interference from MTDs belonging to the same virtual cluster C_i, while the second term represents the variance of the thermal noise, denoted by σ^2 and modeled as an independent Gaussian distribution with zero mean.

To guarantee the QoS of the MTD pair M_j with $j \in \mathbb{M}$, the achievable throughput for the MTD should be larger than a threshold as defined as follows:

$$R_{M_j}^k \geq R_{M_j}^{min}, \forall j \in \mathbb{M} \quad (4)$$

We evaluate the throughput of MTD pair M_j, on RB_k as

$$R_{M_j}^k = \beta.log_2(1 + \frac{P_j^k g_{M_{j,t},M_{j,r}}^k}{P_i g_{U_i,M_{j,r}}^k + \sum_{j' \in C_i, j \neq j'} P_{j'}^k g_{M_{j',t},M_{j,r}}^k + \sigma^2}) \quad (5)$$

where $g_{M_{j,t},M_{j,r}}$ is the channel gain of the M2M communication from the MTD transmitter $M_{j,t}$ to the MTD receiver $M_{j,r}$, $g_{M_{j',t},M_{j,r}}$ is the interference link from the $M_{j',t}$ to $M_{j,r}$, and $g_{U_i,M_{j,r}}$ is the interference link from the UE to $M_{j,r}$. The first term in the denominator in Eq. 5 represents the interference from UE U_i to the $M_{j,r}$, while the second term represents the inter-cluster interference. This latter represents the interference generated from MTDs of the same virtual cluster when sharing the radio resource, RB_k.

V. Formulation of the Transmission Strategy Selection Algorithm based on Evolutionary Game

We consider a preference order for the transmission strategy of each M2M pair. Hence, for any given M2M pair, $Reuse_{NCG} \geq_j Reuse_{CG}$ implies that an MTD pair M_j with $j \in \mathbb{M}$ matched to a UE U_i in a virtual cluster C_i with $j \in \mathbb{N}$ strictly prefers a non-cooperative scenario due to the individual selfish behavior of MTD over a cooperative scenario. Here, NCG stands for non-cooperative game while CG stands for a cooperative game. Initially, we assume a non-cooperative scenario where MTDs are free to act according to their own interests without regard to the overall performance of the virtual cluster. In a non-cooperative game, we consider that MTD transmitters use the maximum transmit power aiming to gain their targeted QoS. If the obtained QoS of MTD is less than the required QoS, then MTDs belonging to the same virtual cluster C_i whose QoS is not satisfied form a coalition denoted by Co_i and switch in a fully distributed manner from a non-cooperative transmission strategy to a cooperative transmission strategy. Let's assume that L' is the number of MTD pairs in the formed coalition of each virtual cluster. Algorithm 1 implements the switch rule for the transmission strategy selection of each MTD pair. We also consider the cost of information exchange inside a coalition. Then, we propose two alternative power control algorithms based on cooperative game theory.

Algorithm 1 Transmission strategy selection algorithm

1: **for** each Virtual Cluster C_i, $i \in \mathbb{N}$ **do**
2: Each MTD link starts to play selfishly in a non-cooperative scenario using the maximum transmit power
3: **if** MTD's QoS requirement is not satisfied **then**
4: MTDs whose QoS is not satisfied form a coalition Co_i
5: **if** Cost of information exchange ($\hat{P}_j$ in Eq. 6) is less than a predetermined threshold ($\tilde{P}_j$) **then**
6: MTDs in the coalition switch to a cooperative game
7: **end if**
8: **end if**
9: **end for**

VI. Power Control in Cooperative Game

In this section, we present our proposed power control scheme for MTDs inside each coalition based on the cooperative game theory.

A. Cost of information exchange

We consider the cost of information exchange inside each coalition Co_i in terms of transmit power in order to model the data exchange penalty. Consequently the total power cost for a coalition is taken as the sum of the powers required by each MTD transmitter in a coalition, $M_{j,t} \in Co_i$, to communicate to the remaining MTD transmitters of the same coalition $M_{j',t} \in Co_i, j' \neq j$ and can be expressed as:

$$\hat{P}_j^k = \sum_{j' \in Co_i, j' \neq j} \frac{\nu_0 . \sigma^2}{(g^k_{M_{j,t}, M_{j',t}})^2} \tag{6}$$

where ν_0 is a target average SNR for information exchange, σ^2 is the noise variance and $g_{M_{j,t}, M_{j',t}}$ is the channel gain between MTDs transmitters, from $M_{j,t}$ to $M_{j',t}$. $\hat{P}_j$ should be less than a given threshold denoted by $\tilde{P}_j$.

B. Payoff function

Let's denote π the payoff function of different MTD transmitters $M_{j,t}$ in the same Coalition Co_i of each virtual cluster C_i. Here, we consider a joint throughput and power control game model. The payoff function of an $M_{j,t}$ is composed of an utility function and a cost function, where $\frac{R^k_{M_j}}{R^{min}_{M_j}}$ is the utility function that represents the user's satisfaction in terms of throughput and $\frac{P^k_j}{P^{max}_j}$ is the cost function that depends on the power consumption. α and γ are the positive weights constants of the throughput and the price of the transmission cost, respectively.

$$\pi_j(R^k_{M_j}, P^k_j) = \alpha . \frac{R^k_{M_j}}{R^{min}_{M_j}} - \gamma . \frac{P^k_j}{P^{max}_j} \tag{7}$$

C. Fixed mixed-strategy power control algorithm

The L'-player cooperative power control game is formulated as $G = \{L', P, \pi\}$, with $L' = \{1, 2, ..., L'\}$ as the player set and P is the strategy set. Each MTD $M_{j,t}$ selects a transmit power lever P_j such that $P_j \leq P^{max}_j$ where P^{max}_j is the maximum allowed transmit power of an MTD. π is the payoff function which characterizes the utility function and its cost function. The power level is usually quantized into discrete values in practice [17]. Therefore, the power level of M2M pair M_j is assumed to be chosen from a finite set P_j. At each time TTI, each MTD transmitter $M_{j,t}$ forwards its data with transmit power strategies from $P_j = \{P_j(1), P_j(2), ..., P_j(N')\}$ where N' is the total action sets. In a fixed mixed-strategy power control algorithm, we consider the case of a fixed power level where the strategy space P_j of each MTD has a minimum and maximum power constraints. Throughout this paper, we assume that the available power level for each MTD is with the same dimension. According to definition of the payoff function, we can obtain the payoff matrix when different $M_{j,t}$ choose multi-power levels.

- Solve Mixed Strategy Nash Equilibria

To solve the matrix power control game, we use the fundamental theorem of mixed strategy Nash equilibria [18]:

Definition 2: Let σ= (σ_1,σ_2,...$\sigma_{L'}$) be a mixed strategy profile for an L'-player game. For any player $j = 1, 2, ..., L'$ let σ_{-j} represent the mixed strategies used by all the players other than player j, Let S_j the finite set of pure strategies available to player j, and let $\pi_j(s, \sigma_{-j})$ $s \in S_j$ be the payoff to player j when playing s against σ_{-j}.

Then σ is a Nash equilibrium $\Leftrightarrow$ the following two conditions hold:

- If s, s' in S_j are two strategies that occur with positive probability in σ_j, then $\pi_j(s, \sigma_{-j})$= $\pi_j(s', \sigma_{-j})$
- If s, s' in S_j where s occurs with positive probability in σ_j, and s' occurs with zero probability in σ_j, then $\pi_j(s, \sigma_{-j}) > \pi_j(s', \sigma_{-j})$

D. Adaptive mixed-strategy power control algorithm

Rather than fixing the power levels in the strategy space of each MTD having a minimum and maximum power constraints, we use two novel power control schemes that efficiently adjust the MTD transmit power in order to not only assure the desired QoS of H2H users and maximize the spectrum efficiency of M2M communications but also to save the battery life of MTDs.

1) Strategy space set using the PID controller: The primary goal is to drive the actual UE throughput, $R^k_{U_i}$, obtained in an H2H/M2M case to converge to its corresponding required QoS in terms of throughput, $R^{min}_{U_i}$ [11]. Therefore, the PID controller takes as input the error signal ***e(k)*** that should be related to the MTD transmit power $P^k_j(k)$. Particularly, ***e(k)*** represents the gap between the current MTD transmit power and the maximum MTD transmit power. This latter is determined given the UE's interference threshold that assure its desired QoS, $R^{min}_{U_i}$. The PID controller takes as output the power control ratio, ***u(k)***, determined by a weighted sum as follows:

$$\begin{aligned} u(k) = u(k-1) + k_p\Big(1 + \frac{T}{T_i} + k_p \frac{T_d}{T}\Big)e(k) \\ - k_p\Big(1 + 2\frac{T_d}{T}\Big)e(k-1) + k_p \frac{T_d}{T} e(k-2) \end{aligned} \tag{8}$$

where T and $e(k)$ represents the sampling period and the error signal at the k^{th} sampling period, respectively. T_i and T_d are two parameters that depends on the proportional gain k_p, the integral gain k_i, and the derivative gain k_d. They are equal to (k_p/k_i) and (k_d/k_p), respectively. In each scheduling period (TTI), the new MTD transmit power is determined by multiplying the actual MTD transmit power and the power control ratio R_{pc} derived from the output of the PID system. The transmit power control ratio of the MTD transmitter $M_{j,t}$ is expressed as follows in dB domain:

$$R^k_{pc}(dB) = 10.log10 \frac{(P^k_j)^{new}}{(P^k_j)^{actual}} \tag{9}$$

$$= (P^k_j)^{new}(dBm) - (P^k_j)^{actual}(dBm) \tag{10}$$

Where $(P^k_j)^{new}$ is the new transmit power and $(P^k_j)^{actual}$ is the actual transmit power of $M_{j,t}$ on RB_k.

Thus, by applying the PID controller, the transmit power is limited to assure the desired QoS of H2H users and also to maximize the efficiency of M2M spectrum usage.

2) Strategy space set using the fuzzy logic: Unlike the proposed MTD transmit power control process using a PID controller where a precise output is generated using a mathematical model, fuzzy controllers approximate the mathematical solution and thus less computational complexity is required. Furthermore, the fuzzy logic can incorporate the human knowledge into a machine based decision. We design a fuzzy power controller that dynamically adjusts the transmit power of the specific MTD in order to assure the QoS of H2H users as well as to maximize the efficiency of the MTD spectrum usage. The desired UE throughput should be greater than a predetermined threshold (see Eq. 2). From Eq. (3), we notice that the MTD transmit power is influenced by the following two parameters called antecedents in the fuzzy logic.

Antecedent *x*1) : The UE's throughput level in an H2H scenario. Indeed, the UE's throughput given by Eq. (3) obtained in an exclusive H2H mode (no interference from M2M mode, $\sum_{j \in C_i} P_j^k g_{M_{j,t},eNB}^k = 0$) is a predominant parameter of the MTD transmit power and reflects the sensitivity to interference.

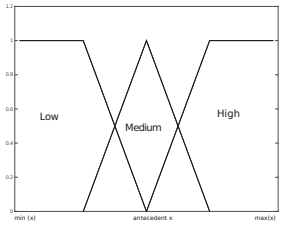


Fig. 4: The membership function for the antecedent

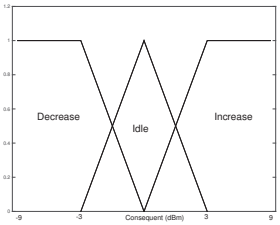


Fig. 5: The membership function for the consequent

Consequently, the antecedent $(x1)$ is the ratio of the throughput of a UE in an exclusive H2H scenario to the corresponding QoS. It is given by:

$$x1 = \frac{R_{U_i}^k}{R_{U_i}^{min}} \tag{11}$$

The linguistic variable x1 characterizes the transmission state of the H2H link with the term set: *low*, *medium* and *high*.

Antecedent *x*2) : The channel information of the interference link caused by an MTD to the reclaiming UE is also a predominant parameter of the MTD transmit power. Similarly to antecedent (x1), given the UE's interference constraint to maintain the QoS of H2H users, the ratio of the actual interference introduced by M2M mode to the interference threshold is used to evaluate the level of interference. We use the linguistic variable $x2$ that specifies *low*, *medium* and *high*.

$$x2 = \frac{P_j^k g_{M_{j,t},eNB}^k}{P_j^{max} g_{M_{j,t},eNB}^k} \tag{12}$$

Consequent: The consequent of this process, which is the MTD transmit power control ratio (R_{pc}^k), is defined in (9) and the new transmit power level is obtained by multiplying the actual transmit power and the power control ratio derived from the output of the fuzzy logic system together. The consequent is divided into three levels: *decrease*, *increase* and *idle*. Trapezoidal membership functions are used to represent the level of the antecedents $x1$ and $x2$ as well as the consequent as depicted in Fig. 4 and Fig. 5, respectively.

Fuzzy inference system We set up the fuzzy rules based on linguistic knowledge from a group of experts after defining the membership functions. The number of rules is $2^3 = 9$ rules since there are two antecedents and each antecedent has 3 fuzzy sub-sets. We establish the fuzzy control rules as shown in Table I. Finally, the defuzzification is performed using the most well known method: centroid.

TABLE I: Rule base

Antecedent x1	Antecedent x2	Consequent
low	low	increase
low	medium	increase
low	high	increase
medium	low	increase
medium	medium	idle
medium	high	decrease
high	low	increase
high	medium	increase
high	high	increase

After properly setting the transmit power levels of the discrete strategy space of MTD players in the cooperative scenario using PID and fuzzy logic controllers, we calculate the payoff matrix when different $M_{j,t}$ choose multi-power levels. Then, we solve the mixed strategy Nash equilibria of the obtained payoff matrix (see Definition 2).

VII. Performance evaluation

In order to evaluate the efficiency of the proposed Transmission Strategy Selection Algorithm (TSSA), we conduct the following simulations based on the 3GPP LTE system model [19]. The main parameters are summarized in Table II. We consider an isolated cell where traditional H2H and M2M communications coexist and can share the RBs for individual data transmission. The system bandwidth considered is $10MHz$. Therefore, 50 usable RBs are available per TTI. The channel model accounts for small scale Rayleigh fading and large scale path loss (log-normally distributed). The MTDs and UEs in the cell are distributed randomly each transmission frame. For simplicity and without loss of generality, we assume one RB is assigned to each H2H user per TTI. This assumption is due to the complexity introduced to uplink LTE scheduling algorithms because of the adjacency and power restrictions imposed by the Single Carrier-Frequency Division Multiple Access (SC-FDMA) and which is not the aim of our work. On the other side, one RB is assumed to be sufficient to fulfill the M2M throughput requirement. We assume two power level strategies for each player $\{P_1,P_2\}$, where $P_1 = 8dBm$ and $P_2 = 11dBm$ in the proposed fixed discrete strategy space. P_1 and P_2 are the mean of the obtained MTD transmit powers when using the PID and

fuzzy logic controllers, respectively in the adaptive discrete strategy space. We use the *round robin* (RR) scheduler for H2H communications. We also consider that 3 is the maximum size of the virtual cluster which means that up to 3 M2M pairs can share the sub-channel with a given H2H user. In other words, a maximum number of M2M pairs of $M = 3.N = 150$ can share the sub-channels with H2H users in each transmission frame. The simulation results are obtained through averaging 50 different realizations.

TABLE II: SIMULATION PARAMETERS

Parameter	Value
Cell radius	$500m$
UEs per cell	N = 50
Path loss model	UMi in [19]
P_i	$23dBm$
P_j^{max}	$14dBm$
$D2D\ range$	$150m$
Noise power spectrum density	$-174dBm/Hz$
Carrier frequency	$2.5Ghz$
Small scale fading	Rayleigh fading coefficient with zero mean and unit variance
Modulation	QAM
K_p, K_d, K_i	0.3
$R_{U_i}^{min}$	$64Kbps$
$R_{M_j}^{min}$	$\{5Kbps, 9.2Kbps, 15Kbps\}$
α	1
γ	0.7
ν_0	$10dB$
$\tilde{P_j}$	$0.01P_j$

The nomenclatures, H2H and H2H/M2M mentioned in all figure legends refer respectively to the exclusive H2H case and the H2H/M2M coexistence case.

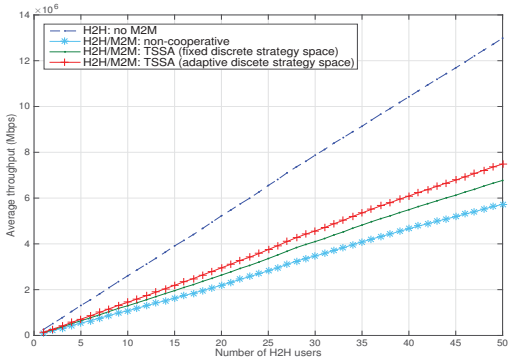


Fig. 6: H2H average throughput

Fig. 6 demonstrates the H2H throughput as function of H2H users in order to evaluate the impact of introducing M2M communications on H2H services. We compare the two proposed Transmission Strategy Selection Algorithm (TSSA) with a fixed discrete strategy space and adaptive discrete strategy space with a non cooperative approach. The latter consists of using the maximum MTD transmit power between M2M pairs sharing the same resource block. The worst performance in an H2H/M2M scenario is obtained when using the non cooperative strategy where the total H2H throughput for 50 UEs is decreased about 60 % compared to the throughput obtained in an exclusive H2H scenario. On the other hand, the H2H throughput is decreased about 50 % and about 40% when using TSSA with fixed discrete strategy space and TSSA with adaptive discrete strategy space, respectively. This can be explained by the fact that the adaptive strategy space adjusts properly the M2M transmit power in a way that the QoS of H2H users is assured contrarily to the fixed strategy space where high interference situations cannot be avoided. The degradation is justified by the multiple resource sharing even-thought the virtual clustering process assure the desired QoS of H2H users.Both qualitative fairness measure, using max-min's

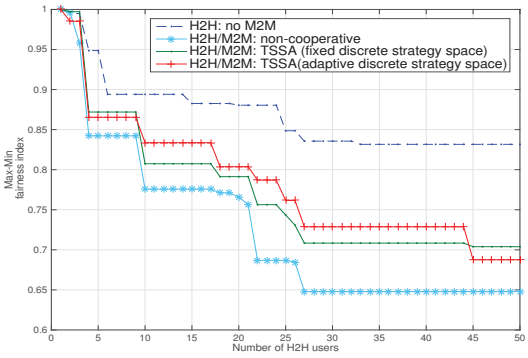


Fig. 7: Max-Min's Fairness index

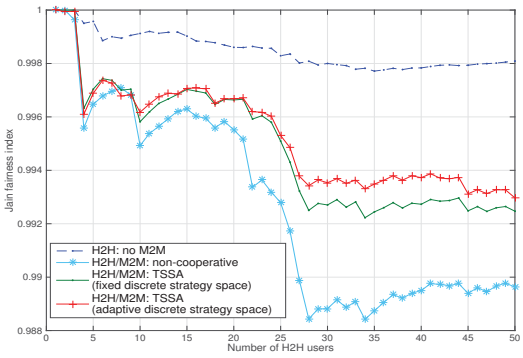


Fig. 8: Jain's Fairness index

fairness index, and quantitative fairness measure through jain's fairness index based on throughput are illustrated in Fig. 7 and Fig. 8 respectively to give us more information about how the fairness policy of H2H scheduler get affected due to the emergence of M2M communications. Both measures show that our proposed TTSA with a fixed discrete strategy space and adaptive discrete strategy space maintains the fairness level of the the existing H2H scheduler, namely RR which is quite interesting. It is clearly seen that the level of fairness of H2H users is more reduced when using a non-cooperative strategy.

Fig. 9 shows the total power consumption of MTD transmitters as a function of transmission time. Our proposed TTSA

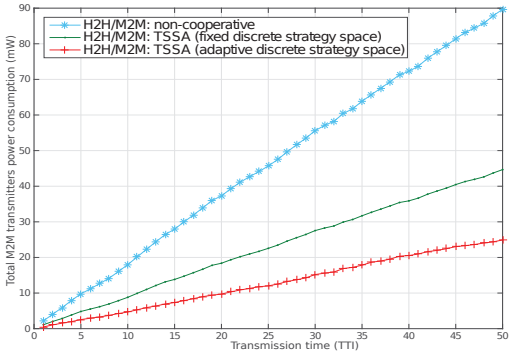


Fig. 9: Total power consumption of MTD transmitters

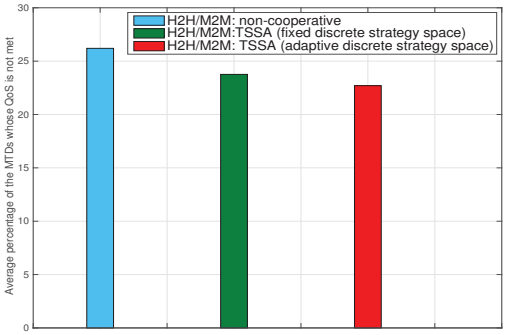


Fig. 10: Percentage of M2M pairs whose QoS is not satisfied

with adaptive discrete strategy space achieves significant reduction in terms of transmit power of about 45% and 70% compared to the TTSA with fixed discrete strategy space and non-cooperative approaches, respectively and hence improving notably the device battery life.

Fig. 10 illustrates the average percentage of MTDs whose QoS in terms of throughput is not met for 50 TTIs. The proposed TSSA algorithm with both alternatives slightly outperforms the non-cooperative strategy where MTDs are using the maximum transmit power which is quite promising. Indeed, this proves that the proposed TSSA algorithm guarantees the M2M communications while achieving a significant gain in terms of transmit power consumption as demonstrated in Fig. 9 as well as assuring the desired QoS of H2H users.

VIII. Conclusion

In this paper, we have studied the resource sharing problem for M2M communications using D2D mode, where multiple D2D pairs can use the same sub-channel. We have developed an efficient evolutionary game based transmission strategy selection algorithm for M2M communications underlaying cellular networks aiming to guarantee traditional H2H services while maximizing the spectrum efficiency of M2M links. Specifically, we have considered initially a non-cooperative scenario since M2M devices are selfish as they are in the practical scenarios. Then, M2M devices whose QoS is not satisfied switch to a cooperative scenario. We have proposed two alternative power control schemes in the cooperative scenario: a fixed and an adaptive mixed- strategy power control schemes. While the former consider a fixed discrete strategy space, the latter approach enables to set properly the power levels using a fuzzy logic and a PID controllers in order to save the battery life of M2M devices. Simulation results show that the evolutionary game based M2M transmission strategy selection algorithm avoids significant degradation of H2H services in terms of throughput and maintains the UE's level of fairness compared to the non-cooperative approach. In addition, the adaptive mixed-strategy power control scheme allows to save considerably the MTD battery life while maintaing the QoS of M2M links. As a future work, we intend to perform a comparative study when varying the dimension of the strategy space.

References

[1] M. Chen, "Towards smart city: M2m communications with software agent intelligence," *Multimedia Tools and Applications*, vol. 67, no. 1, pp. 167–178, 2013.

[2] M. Chen, J. Wan, and F. Li, "Machine-to-machine communications," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 6, no. 2, pp. 480–497, 2012.

[3] T. 3GPP, Technical Specification Group Radio Access Network, "study on ran improvements for machine-type communications (release 10)," 2010.

[4] A. Asadi, Q. Wang, and V. Mancuso, "A survey on device-to-device communication in cellular networks," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 4, pp. 1801–1819, 2014.

[5] N. Abu-Ali, A.-E. M. Taha, M. Salah, and H. Hassanein, "Uplink scheduling in lte and lte-advanced: Tutorial, survey and evaluation framework," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 3, pp. 1239–1265, 2014.

[6] R. Kwan and C. Leung, "A survey of scheduling and interference mitigation in lte," *Journal of Electrical and Computer Engineering*, vol. 2010, p. 1, 2010.

[7] E. Yaacoub and Z. Dawy, "A survey on uplink resource allocation in ofdma wireless networks," *Communications Surveys & Tutorials, IEEE*, vol. 14, no. 2, pp. 322–337, 2012.

[8] A. S. Lioumpas and A. Alexiou, "Uplink scheduling for machine-to-machine communications in lte-based cellular systems," in *GLOBECOM Workshops (GC Wkshps), 2011 IEEE*, 2011, pp. 353–357.

[9] S. Zhenqi, Y. Haifeng, C. Xuefen, and L. Hongxia, "Research on uplink scheduling algorithm of massive m2m and h2h services in lte," 2013.

[10] S. Bayat, Y. Li, Z. Han, M. Dohler, and B. Vucetic, "Distributed massive wireless access for cellular machine-to-machine communication," in *2014 IEEE ICC*, 2014, pp. 2767–2772.

[11] S. Hamdoun, A. Rachedi, and Y. Ghamri-Doudane, "A flexible m2m radio resource sharing scheme in lte networks within an m2m/h2h coexistence scenario," in *IEEE ICC*, 2016.

[12] M.-L. Ku, *Cognitive Radio and Interference Management: Technology and Strategy*. IGI Global, 2012.

[13] A. Ksentini, Y. Hadjadj-Aoul, and T. Taleb, "Cellular-based machine-to-machine: overload control," *Network, IEEE*, vol. 26, pp. 54–60, 2012.

[14] J. W. . S. Ltd, "System engineering in wireless communications," *Communications Surveys & Tutorials, IEEE*, 2009.

[15] E. Cox, "Fuzzy fundamentals," *Spectrum, IEEE*, vol. 29, no. 10, pp. 58–61, 1992.

[16] E. U. T. R. A. E.-U. P. C. 3GPP and r. G. P. P. G. Modulation" TR 36.211, "study on ran improvements for machine-type communications (release 10)," 2010.

[17] Y. Xing and R. Chandramouli, "Stochastic learning solution for distributed discrete power control game in wireless data networks," *IEEE/ACM Transactions on networking*, vol. 16, pp. 932–944, 2008.

[18] D. Fudenberg and J. Tirole, "Game theory," *The MIT Press, Cambridge, Massachusetts*, 1991.

[19] M. Series, "Guidelines for evaluation of radio interface technologies for imt-advanced," *Report ITU*, pp. 2135–1, 2009.

On The Privacy-Utility Tradeoff in Participatory Sensing Systems

Rim Ben Messaoud*†, Nouha Sghaier†, Mohamed Ali Moussa* and Yacine Ghamri-Doudane†
*LIGM Laboratory, Université Paris-Est, Champs sur Marne, France
†L3I Laboratory, University of La Rochelle, La Rochelle, France
{rim.ben_messaoud, nouha.sghaier, yacine.ghamri}@univ-lr.fr
mohamed-ali.moussa@u-pem.fr

***Abstract*—The ubiquity of sensors-equipped mobile devices has enabled citizens to contribute data via participatory sensing systems. This emergent paradigm comes with various applications to improve users' quality of life. However, the data collection process may compromise the participants' privacy when reporting data tagged or correlated with their sensitive information. Therefore, anonymization and location cloaking techniques have been designed to provide privacy protection, yet to some cost of data utility which is a major concern for queriers. Different from past works, we assess simultaneously the two competing goals of ensuring the queriers' required data utility and protecting the participants' privacy. First, we introduce a trust worthy entity to the participatory sensing traditional system. Also, we propose a general privacy-preserving mechanism that runs on this entity to release a distorted version of the sensed data in order to minimize the information leakage with its associated private information. We demonstrate how to identify a near-optimal solution to the privacy-utility tradeoff by maximizing a privacy score while considering a utility metric set by data queriers (service providers). Furthermore, we tackle the challenge of data with large size alphabets by investigating quantization techniques. Finally, we evaluate the proposed model on three different real datasets while varying the prior knowledge and the obfuscation type. The obtained results demonstrate that, for different applications, a limited distortion may ensure the participants' privacy while maintaining about 98% of the required data utility.**

***Keywords*—Participatory Sensing, Privacy-preserving, Data utility, Convex Optimization, Tradeoff.**

I. Introduction

The emergent Participatory Sensing (PS) paradigm [1] has powered people carrying sensors-equipped devices to share and collect data about specific phenomenon. In this context, mobile users, denoted as participants, are actively involved in sensing tasks as in the case of taking photos or recording audios/videos. Besides, the participatory applications may be allowed to report *opportunistically* other sensors' readings such as the accelerometer, GPS and gyroscope [2]. As a result, various potential applications have been proposed to address different issues affecting individuals like activity recognition and health care [2] or a community such as urban traffic monitoring [3] and environment management [4].

Participatory sensing comes also with significant advantages when compared to traditional sensing networks. First, it leverages the cost limitations of the static deployed sensors and gathers users' data from places not economically feasible before such as in traffic congestion control applications [5]. Moreover, it solves the coverage range issue and offers a much broader one given the participants' mobility. However, this promising paradigm raises new challenges [2] such as the energy consumption issue, the necessary rewarding mechanisms and mainly the privacy concerns which are the most significant barriers to users' participation in the sensing process.

978-1-5090-3216-7/16/$31.00

Indeed, reporting data in a community-scale task may incur private information leakage. Particularly, most of participatory applications require location and sensing time tagged data. Hence, an adversary can derive sensitive information (e.g. the participant's residence, income level, political affiliation, medical condition, etc.) by only observing the multiple reports in the system [6]. Besides, other applications do not access directly private information but can collect sensors' readings that may be correlated with. For instance, by sending the energy consumption reports of different smart home equipment, users may release *unintentionally* their household activities to a service provider. Thus, users are reluctant to participate to sensing campaigns unless with privacy protection guarantee.

A variety of privacy-preserving schemes have been proposed in the literature. The simpler is the pseudonymity [7] where participants share their sensors' readings associated with pseudonyms instead of their real identities. However, this does not guarantee necessarily privacy. Furthermore, encryption-based methods have been implemented as a user-centric model [8]. Hence, it comes with an important computation complexity which may drain users' devices batteries. Other methods deploy mainly generalization [9], [10] or obfuscation techniques [11]. The former method is based on reducing the accuracy of the reported data to "general" value common among a set of participants such as in spatial cloaking methods [12]. While the latter is based on perturbing data, by adding random noise or hiding samples of measurements of the original data, in order to protect participants' sensitive information. Yet, this impacts the usefulness of the collected data, a major criterion in sensing systems. In fact, the utility of a data source is related to its ability to reveal viable information, necessary to analysts or queriers to offer services that meet users' needs. Thus, any approach that gives priority only to the information privacy aspect while overlooking the resultant reduction in utility is not likely to be practically usable.

To address utility and privacy competing goals, we propose in this paper a privacy-preserving model which conserves the queriers' data utility requirements. Therefore, we introduce first a trusted entity in the participatory sensing system as described in Figure 1. This entity, denoted broker, could

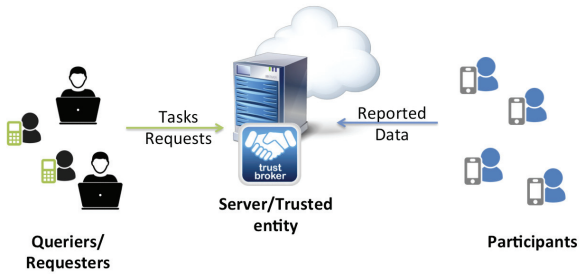


Fig. 1. The centralized architecture of PS system with trusted entity

be merged with the central server, i.e, it receives queriers' specific requests and collects data from participants. Also, we propose to run on the broker, a general obfuscation-based mechanism that aims to achieve a utility-privacy tradeoff. The basic idea is to obfuscate the reported data by participants in order to minimize the privacy leakage with their private information without degrading the final released data. Our main contributions can be summarized as follows:

- We present privacy and utility metrics to quantify the requirements of each part of the participatory sensing system; the requesters and the participants.
- We propose a utility-aware privacy-preserving mechanism that achieves a tradeoff between the queriers' requirements and the participants' privacy concerns.
- We validate our privacy-utility tradeoff mechanism on three different real datasets while varying the distribution of the collected data and the obfuscation type which highlights its generality and efficiency.

The rest of this paper is organized as follows. Section II gives an insight into the related work. Section III describes our system model. Section IV and V outline the mathematical formulation and the proposed algorithm. We evaluate the proposed privacy-utility tradeoff mechanism in Section VI. Conclusions and Future work are withdrawn in Section VIII.

II. Related Work

In this section we summarize the most relevant research work of the privacy in PS and privacy-utility tradeoff.

A. Privacy in Participatory Sensing

Privacy-preserving mechanisms vary based on their adopted techniques. We mainly distinguish four classes; cryptography, anonymization, data aggregation and obfuscation/perturbation. The former class relies on encryption method as in the case of PEPSI [8], a privacy-enhanced architecture for crowdsensing applications. In this work, authors aim to hide sensing reports from unauthorized entities. Thus, each participant has to obtain an encryption key to cipher his collected data and a decryption key to be known by the end-user (service querier) to decipher it. This process requires high computation and energy resources which makes it unsuitable for most PS applications. Moreover, the queriers can be curious to gather information about the participants and hence the collected data should be distorted before being reported to them in order to protect participants' sensitive information.

Another widely adopted privacy-aware technique in PS systems is the anonymization. This method aims to avoid association between the participants' identities and their private information [5] and has been implemented especially for location-privacy. In this context, authors in [9] utilized the *Tesselation* technique to generalize one user's location in a *Tile* containing k other users' locations to guarantee k-anonymity. This results in a so-called spatial cloaking to prevent participants' trajectories tracking. In the contrast, the anonymization can not be used in many crowdsensing applications that require accurate location tagged data.

Shi et al. [13] focused on privacy-preserving data aggregation in PS systems and proposed a distributed-scheme called PriSense. The basic idea is to not rely on a central entity to provide protection to participants. However, the participants tend to distribute their collected data among their neighbors. Upon receiving a request from the aggregation server, each participant returns his data and the remaining data of his neighbors. This reduces the probability to successfully attribute each sensor reading to its corresponding mobile user. Nevertheless, this scheme ignores the inter-participants threat. In fact, participants may be curious to collect information about each-other. Particularly, in competitive sensing campaigns, disclosing the bid of other users may help the participant to select an adequate price and get the task [14].

Researchers adopted also data perturbation/obfuscation techniques which consist of perturbing the data by adding noise or by hiding some of its features [12]. Poolview [11] is a novel data perturbation mechanism that generates a noise model with similar characteristics to the phenomenon measured by the PS application. Then, this model is distributed to all participants so they can generate the noise locally and modify the configuration parameters regularly in order to enhance their privacy protection against historical attacks.

The main drawback of the above described methods is that the data utility is slightly studied. Though, different privacy protection mechanisms incur an important data degradation and information loss, essentially, when modifying the attributes of the reported measurement as in the case of perturbation.

B. Privacy-Utility tradeoff

In order to address the privacy-utility tradeoff, the research community tends to apply information-theoretic tools [15–18]. Particularly, using rate-distortion theory, authors in [16] have developed a utility-privacy tradeoff region for databases. Their model is based on Shannon entropy which may be inadequate for some applications where individual anonymity guarantees are required. Du Pin Calmon and Fawaz [17] have introduced a general framework for privacy against statistical inference and formulated a convex program to find privacy-preserving mappings for minimizing the information leakage from a user's data with utility constraints. Both works address collective privacy in database systems while considering the data utility constraint. However, they present only theoretic metrics with no evaluation on real applications. Further, authors in [18] have extended the work of [17] with the aim of solving the privacy-utility issue in rating web applications.

Inspired by these works, we propose a general data utility-aware privacy-preserving mechanism in participatory sensing

systems that we describe its model in the following section.

III. System Model

In this section, we introduce the different entities involved in our privacy-utility aware PS system. We define also the attacker and the data privacy-preserving and processing models.

A. Participatory Sensing System Architecture

We consider, in this paper, the participatory sensing system of Figure 1, where two parties are communicating via the trusted entity as detailed below:

Participants: Set of smart-devices equipped users registered within the participatory sensing platform to contribute data either voluntarily or while receiving monetary incentives/services in return. Thus, they are more likely to report a good quality contribution in order to get interesting "rewards". However, they have privacy concerns and may require that a set of their contributed measurements should remain private. For instance, by reporting periodic temperature measurements along with collection points, participants may be exposing their trajectories to a service provider (weather service). Consequently, the latter can disclose their behavior and daily-life related information such as the frequented places.

Queriers/Requesters: Set of service providers or individual users looking for a specific data and requiring a predefined quality level under which the information may be perceived as *useless*. Hence, each querier sets a utility threshold value, based on a predefined quality evaluation metric that takes into account the specificity of the data collecting purpose. Then, he sends this value to the broker to be considered as the **minimum** accepted quality level for the collected and reported data.

Server/Broker: A central entity in the cloud that receives information requests from the queriers and assigns the corresponding tasks to the available participants based on their sensing preferences and/or current location. It is worth noting that the tasks assignment scheme adopted by the server is out of the scope of this paper. Also, for simplicity reasons, we consider that the server plays the role of the trusted entity (broker) which collects the participants' contributed data and receives the utility threshold values from queriers.

The broker's goal is to answer participants' privacy concerns by minimizing the probability of inferring their private information from publicly reported measurements while respecting the data utility threshold value set by the queriers.

B. Adversary Model

In this work, we consider both partial-internal and external adversaries [19]. That is, an attacker might be part or not of the participatory sensing system. First, we assume that the server is the only trusted entity who collects data from participants with no curiosity or intentions of comprising their privacy. Second, we consider that other participants and queriers are honest but curious. Thus, they honestly report their data or their utility metrics, but they may try to learn about other users' behaviors from their periodically released sensing data. Also, we consider that external adversaries may be eavesdropping as false queriers to the different collected measurements in order to disclose the participants' private information. Finally, we assume that the adversaries are passive. Hence, they can only read and observe data but they do not modify it.

C. Privacy-preserving Model

Let $M \in \mathcal{M}$ denotes a sensing measurement collected by a participant where $\mathcal{M}$ represents the set of all possible measurements. Indeed, M can be correlated with a participant private information denoted by $S \in \mathcal{S}$, where $\mathcal{S}$ denotes the set of all possible secrets detained by participants. The correlation between M and S can be expressed through the jointly random distribution $P_{M,S}(m, s)$, where $(m, s) \in \mathcal{M} \times \mathcal{S}$ are two realizations of the random variables M and S. This may compromise the participants' privacy since observing M may result in infering S. Therefore, a participant would rather send his collected data M to the broker which should generate a distorted version $D \in \mathcal{D}$ to be reported to the queriers.

Note that we consider the settings described in [17]. Accordingly, we opt for the next defined privacy mechanism:

Definition 1 (Privacy-mapping [17])**.** *A privacy-preserving model is a probabilistic mapping $f : \mathcal{M} \to \mathcal{D}$ characterized by the conditional probability $p_{D|M}(d|m)$ that minimizes the privacy risk to infer a private information S, jointly distributed with a public data M, from the released data D.*

Furthermore, we take into consideration the threshold value on the utility of the reported data D set by the requesters, that we denote by U_δ. In other terms, the broker must report data D with utility $U_D \geq U_\delta$. Therefore, we define the utility regression level as $U_M - U_D$ and the maximum utility regression level as follows:

Definition 2 (Maximum utility-regression level)**.** *The maximum utility regression level tolerated by a querier is the difference between the utility of the sensing data M, U_M, and the utility threshold level, U_δ; $\delta = U_M - U_\delta$.*

D. Data Processing Model

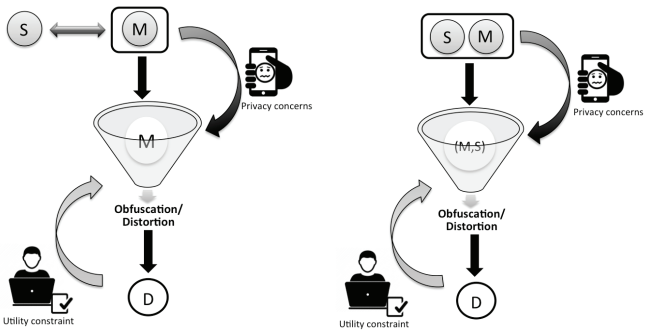
Without loss of generality, we will study separately throughout this paper two scenarios:

- First, we tackle the case of a measurement M correlated with a secret S. For instance, a turned-on room light, considered as a measurement M, indicates its occupancy and thus user's indoor position considered as a secret S. In this case, only the measurement M would be obfuscated.
- Second, we consider the case of a secret S jointly reported with the measurement M. For example, in location tagged sensing tasks, the location S and the measurement M (e.g Temperature) are sent together. Thus, both variables (M, S) should be distorted.

The aforementioned scenarios are illustrated in details in Figure 2. In the next section, we investigate both proposed scenarios and we propose a general mathematical formulation for the privacy-utility tradeoff that we target.

IV. Problem Formulation

In this section, we define adequate evaluation metrics for privacy leakage and data utility. Then, we formulate the corresponding optimization problem.



(a) S is correlated with M (b) S is reported with M

Fig. 2. The different scenarios of data reporting and obfuscation

A. Privacy Leakage Metric

In order to define a robust privacy metric, we should consider the adversary estimation model. According to [19], an adversary may perform a statistical inference on the different measurements reported by participants to make a first estimation of the secret S. Further, when inferring the released data D, the attacker estimates again the possible set of users' secrets. The two estimations come with costs. Therefore, after observing the collected data, the adversary obtains an average cost gain defined by authors in [17] as:

$$\Delta C = C_0 - C_D \tag{1}$$

where C_0 and C_D are the estimation costs with no prior knowledge and after observing the released data, respectively.

With respect to this model and while considering a log-loss cost function, the adversary average cost gain expression turns into the mutual information [17] between the secret S and the released data D expressed as:

$$\begin{aligned} I(S,D) &= H(S) - H(S|D) \\ &= \sum_{s\in\mathcal{S}}\sum_{d\in\mathcal{D}} p_{S,D}(s,d)\log\Big(\frac{p_{S,D}(s,d)}{p_S(s)p_D(d)}\Big) \end{aligned} \tag{2}$$

where $H(S)$ is the entropy of S defined as: $H(S) = -\sum_{s\in\mathcal{S}} p_S(s)\log(p_S(s))$.

The mutual information quantifies how much information is shared between two random variables [19]. In our case, we measure by $I(S,D)$ the quantity of information shared between S and D, which represents the amount of information leaked from a privacy mechanism. We opt for this function as our privacy leakage metric to be minimized. In other terms, we target to maximize the conditional entropy between the secret of a participant S and the distorted data D, $H(S|D)$.

B. Data Utility Metric

We aim to quantify the collected data quality as perceived by a querier. Thus, we reuse the "utility" theory introduced first in economy to evaluate a product or a service then extended to different fields. For example, utility functions have been widely used to evaluate networks or services and can be in different forms such as exponential, logarithmic or sigmoid. In this work, we opt for the utility function proposed by the authors in [20] given its general formulation flexible for different sensing data. We define for a measurement x in the interval $[x_\alpha, x_\beta]$:

$$U(x) = \begin{cases} 0 & x < x_\alpha \quad (3a) \\ \dfrac{(\frac{x-x_\alpha}{x_m-x_\alpha})^\zeta}{1+(\frac{x-x_\alpha}{x_m-x_\alpha})^\zeta} & x_\alpha \le x \le x_m \quad (3b) \\ 1 - \dfrac{(\frac{x_\beta-x}{x_\beta-x_m})^\gamma}{1+(\frac{x_\beta-x}{x_\beta-x_m})^\gamma} & x_m < x \le x_\beta \quad (3c) \\ 1 & x > x_\beta, \quad (3d) \end{cases}$$

where $\zeta \ge max\{\frac{2(x_m-x_\alpha)}{x_\beta-x_m}, 2\}$ and $\gamma = \frac{\zeta(x_\beta-x_m)}{x_m-x_\alpha}$ are the tuned steepness parameters, and x_m is the median of $[x_\alpha, x_\beta]$.

This function sets bounds to the possible values of any distorted version of a measurement M collected by a user and released by the broker as D. Hence, the lower accepted "utility" of D is $U(x_\alpha)$ which corresponds to the aforementioned threshold value set by the queriers, U_δ.

We consider $U(x)$ as the main metric to evaluate the utility of D throughout this work because of its general application. Nevertheless, any other less complex utility metric can be used such as the Euclidean distance in case of location-privacy or the Hamming distance in case of binary reported information.

The data utility metric is rather a constraint to be respected while looking for the privacy mapping that we model by Eq. (4). Thus, we set the maximum accepted utility regression level for a querier, introduced in Definition 2, as an upper bound of the average distance between the utilities of the original measurements M and their distorted versions D.

$$\mathbb{E}_{M,D}[d(U(M),U(D))] \le \delta \tag{4}$$

where $d(x,y)$ is a distance function between two variables x and y that depends on the obfuscation type.

In the following paragraph, we utilize the two defined privacy and utility metrics in order to formulate the corresponding optimization problem.

C. Optimization Problem

Our goal is to minimize the participants' privacy leakage expressed by Eq. (2) while respecting the utility constraint described by Eq. (4). To do so, the broker must achieve such tradeoff by solving the following optimization problem:

$$\begin{aligned} \underset{P_{D|M}}{\text{Minimize:}} \quad & I(S,D) \\ \text{subject to:} \quad & \mathbb{E}_{M,D}[d(U(M),U(D))] \le \delta \end{aligned} \tag{5}$$

In this paper, we adopt the settings of [17] to look for a privacy mapping that solves the problem (5) and we distinguish the formulation for the two scenarios of Figure 2.

1) ***S correlated with M:*** First, we consider the general model where a participant's private information S is correlated with a collected data M. Indeed, the three variables; S, M and D form a Markov chain as follows: $S \to M \to D$. Thus, the joint distribution of S and D can be expressed as function of $p_{S,M}$ and $p_{D|M}$ as follows:

$$p_{S,D}(s,d) = \sum_{m\in\mathcal{M}} p_{D|M}(d|m)p_{S,M}(s,m) \tag{6}$$

According to the formulation of (6), we rewrite the objective function of our optimization problem as a function of the conditional probability $p_{D|M}$ representing our privacy mapping and the joint probability $p_{S,M}$ representing the prior knowledge that the broker can statistically compute from the different historical reported data by participants.

$$\begin{aligned} I(S,D) = & \sum_{s,m,d} p_{D|M}(d|m)p_{S,M}(s,m)\times \\ & \log\Big(\frac{\sum_{m"} p_{D|M}(d|m")p_{M|S}(m"|s)}{\sum_{s',m'} p_{D|M}(d|m')p_{S,M}(s',m')}\Big) \\ & = g(p_{D|M}, p_{S,M}). \end{aligned} \tag{7}$$

Note that p_S, p_M and $p_{M|S}$ could be derived from the joint probability $p_{S,M}$. Moreover, the obtained expression $g(p_{D|M}, p_{S,M})$ responds to the form $ax\log(\frac{x}{z})$ which proves its convexity as stated in [17].

Also, we approximate the distance between utilities to any other distance depending on the obfuscation type; $\mathbb{E}_{M,D}[d(U(M),U(D))] \sim \mathbb{E}_{M,D}[d(M,D)]$. Then, we express the data utility constraint as a function of $p_{D|M}$ and $p_{S,M}$ by placing $p_M(m) = \sum_{s\in\mathcal{S}} p_{S,M}(s,m)$ in the expectancy expression.

$$\begin{aligned} \mathbb{E}_{M,D}[d(M,D)] &= \sum_{m,d} p_{M,D}(m,d)d(M,D) \\ &= \sum_{m,d} p_{D|M}(d|m)p_M(m)d(M,D) \\ &= \sum_{s,m,d} p_{D|M}(d,m)p_{S,M}(s,m)d(M,D) \\ &= h(p_{D|M}, p_{S,M}, d(M,D)) \end{aligned} \tag{8}$$

Finally, we reformulate the problem (5) in order to enhance its dependency on the probabilistic privacy mapping $p_{D|M}$ and the prior knowledge $p_{S,M}$. The final optimization problem formulation is expressed as:

$$\begin{aligned} \underset{P_{D|M}}{\text{Minimize:}} \quad & g(p_{D|M}, p_{S,M}) \\ \text{subject to:} \quad & h(p_{D|M}, p_{S,M}, d(M,D)) \leq \delta \end{aligned} \tag{9}$$

*2) **S reported with M**:* Problem (9) models the utility privacy tradeoff in participatory sensing for any reported measurement that may be correlated with a participant's secret. Without loss of generality, we investigate the case of a secret S reported along with a measurement M as illustrated in Figure 2(b). Hence, the prior knowledge is reduced to the marginal distribution of one measurement $\tilde{M} \sim (M,S)$. As a result, the formulation of our objective and constraint are simplified to the next two functions $I(\tilde{M},D) = g'(p_{D|\tilde{M}}, p_{\tilde{M}})$ and $\mathbb{E}_{\tilde{M},D}[d(U(\tilde{M}),U(D))] = h'(p_{D|\tilde{M}}, p_{\tilde{M}}, d(\tilde{M},D))$.

In the next section, we introduce the proposed algorithm to solve the privacy-utility tradeoff issue in participatory sensing systems for both scenarios.

V. Privacy-Utility Tradeoff Mechanism

Hereafter, we design our privacy-utility tradeoff mechanism and we tackle the different issues that may be encountered in participatory sensing systems such as the mapping of distorted data and large size data alphabets.

Algorithm 1 Privacy-Utility Tradeoff Algorithm

Require: Joint probability $p_{S,M}(s,m)$, Maximum utility regression level δ,

1: **Obfuscation type:** Generate the corresponding $\mathcal{D}$
2: Choose the distortion/utility metric U and compute the distance $d(U(M),U(D)) \sim d(M,D)$
3: Set the vector of variables $X = p_{D|M}$
4: **Interior-point method:**
Minimize : $g(X, p_{S,M})$
subject to:
1) $h(X, p_{S,M}, d(M,D)) \leq \delta$
2) $AX = b$
3) $\sum X = 1, \forall D \in \mathcal{D}$
4) $0 \leq X \leq 1, \forall D \in \mathcal{D}, M \in \mathcal{M}$
5: **Return** $p_{D|M}$

First, we note that problem (9) is convex and bears some approximation to modified rate-distortion problems [17]. Such problems are widely studied in information theory with the aim of computing the bounds of a specific distortion. However, few applications have been developed based on this model since they require to be solved using a dual minimization procedure analogous to the Arimoto-Balhut algorithm [21]. Nevertheless, by using the formulation of problem (9), we may rely on standard and efficient algorithms for solving the convex optimization such as the Interior-Point Method (IPM) or the Successive Quadratic Programming (SQP). Furthermore, we consider a unique algorithm for the two scenarios of Figure 2, since they differ mainly in the formulation and in the prior knowledge but not in the solving method.

Without loss of generality, we assume that the joint probability distribution between a private information S and a collected data M, $p_{S,M}$, is known by the broker by analyzing statistically the multiple sensing reports and thus can be the input of our mechanism. Besides, for simplicity reasons, we suppose that all the queriers for a same type of data have the same maximum utility regression level, δ. Moreover, we consider the fact that we are looking for a probabilistic privacy mapping $p_{D|M}$. Therefore, we add two extra constraints to problem (9) as follows: $\sum_{D\in\mathcal{D}} p_{D|M} = 1, \forall M \in \mathcal{M}$ and $0 \leq p_{D|M} \leq 1, \forall D \in \mathcal{D}, M \in \mathcal{M}$.

Finally, we consider two possible encountered challenges specific to the participatory sensing applications:

*a) **Mapping of distorted data**:* the set of the distorted data $\mathcal{D}$ generated by a broker might be larger/smaller than or equal to the set of measurements $\mathcal{M}$ collected by the participants depending on the type of obfuscation used. Hence, some mapping from M to D may be not possible, i.e, $p_{D|M} = 0$. In order to minimize the computation time, we set such probabilities to be null. To do so, we form a boolean matrix A with ones in the positions of potential zeros probabilities and zeros otherwise. Next, we add the following condition to our optimization problem: $AX = b$, where $X = p_{D|M}$ and b is a zeros vector with the size of X.

*b) **Data with large size alphabets**:* most of participatory sensing collected data is rather with large size alphabets such as the environment sensors' readings. Therefore, the size of the set of measurements $\mathcal{M}$ can be very important which results in a challenging estimation of the prior knowledge $p_{S,M}$. Furthermore, the set of distorted data $\mathcal{D}$ can be, depending on the selected obfuscation, as large as $\mathcal{M}$. Thus, the solving of the predefined optimization problem (9) could be complex and time consuming. Also, regarding the non-linear nature of the objective function $g(p_{D|M}, p_{S,M})$, we should minimize the size of data to be distorted in order to utilize the standard convex solvers. In order to deal with such an issue, we opt for the well-known quantization techniques. We differ the quantization method based on the type of the sensing application. For example, if the participatory sensing task targets collecting location-tagged data, we can consider clustering the sensing area into small cells. Though, we must highlight that the quantization step represents an additional distortion source which may yield to sub-optimal privacy-utility tradeoffs. Hence, we generate a new set of data alphabets that we denote by $\mathcal{Q}$ and we compute the corresponding prior knowledge $p_{S,Q}$ that represents the new input of our privacy-utility algorithm. The resulted privacy mapping is defined as $p_{D_q|Q}$ where $D_q \in \mathcal{N}$ is the set of distorted quantized-data. Finally, we map the obtained probabilities values to their corresponding original alphabets.

The steps of our privacy-utility tradeoff method are summarized in Algorithm 1. First, we compute the prior knowledge of the broker, $p_{S,M}$. Next, we select an obfuscation technique based on the data type (e.g exchanging position for location data, adding noise for audio/photo, etc.). Accordingly, we generate the set of possible distorted data $\mathcal{D}$ and compute the distance among utilities U_M and U_D. Finally, we run the IPM method to solve our optimization problem.

VI. Performance Evaluation

Hereafter, we present three different real sensing datasets to be considered as crowd sensed measurements. We run simulations on each different dataset and we evaluate the observed privacy and utility metrics while varying the prior knowledge distribution, the obfuscation type and the scenario.

A. Real Sensing Datasets

*1) **Occupancy Detection Data**:* This dataset was generated by authors in [22] with the aim of evaluating the accuracy of different classification Machine Learning algorithms. The experiment consists of detecting if a room is occupied or not based on environment sensors readings. The sensors collect the ambient temperature, the humidity, the light and the CO_2 level measurements. In this work, we consider that such readings are the measurements M reported via a crowdsensing application to a service provider in order to monitor smart-house connected objects. However, when reporting these measurements, a participant can expose his position which is considered as the secret S since it reveals his *indoor* lifestyle.

*2) **GPS Trajectory Data**:* This data has been collected from the GO_Track mobile application [23] and studied by authors in [24] in order to identify similar trajectories for carpooling purpose. The readings include rating of the traffic, weather and transportation besides collecting the visited geographical points, the vehicle speed and the total realized distance. Hence, we consider here a traffic rating crowdsensing application that reports periodically these measurements. And, we assume that the private information S is the participants' driving behavior which can be inferred from the reported speed, weather and traffic status values.

TABLE I. Occupancy data characteristics

Datasets	Data Distribution	
	0 (non occupied)	1 (occupied)
Dataset1 (closed door)	0.64	0.36
Dataset2 (open door)	0.79	0.21

*3) **Crowd Temperature Data**:* This dataset is contributed by Mohannad et *al*. [25], as collected outdoor temperature by taxis in Rome. Taxicabs are equipped with temperature sensors attached to their vehicles which report their readings along with the corresponding geographical position to a central server every 6 hours. In this paper, we suppose that such measurements are gathered via a participatory sensing application. That is, the participants' private information (location) is sent along with the measurements, which represents the scenario of Figure 2(b).

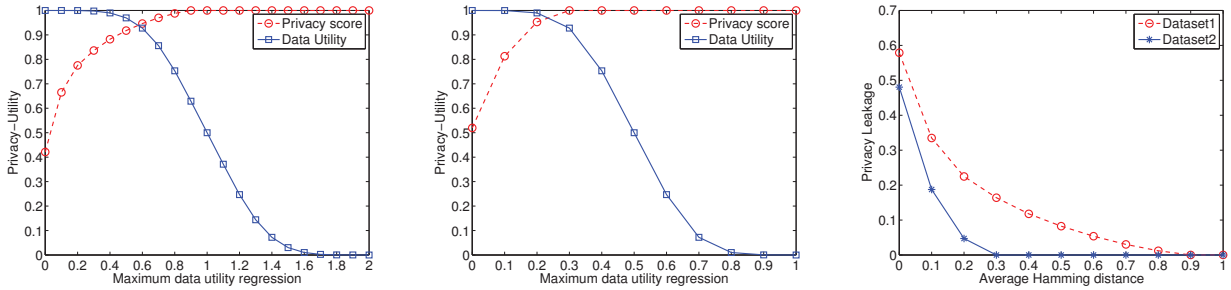
B. Evaluation Results

In the following, we run our privacy-utility tradeoff method, using the convex solver Interior Point Method of Matlab, on the above described dataset for various participatory sensing applications and scenarios. For different plots, we recall the utility metric U of Eq. (3) to measure the achieved data utility and the mutual information of Eq. (2) to quantify the privacy leakage. Also, we set an inversely proportional privacy score as $P_s = 1 - I(S, D)$.

*1) **Prior knowledge impact (scenario 1)**:* We consider the crowdsensing from smart-house connected objects applications using the occupancy detection data [22]. We utilize two datasets collected with room door open and closed as detailed in Table I. Let $S \in \{0, 1\}$ be the room occupancy status where $S = 1$ for an occupied room and 0 otherwise. Also, we denote by $M = \{T, H, L, CO_2\}$ and $D = \{\tilde{T}, \tilde{H}, \tilde{L}, \tilde{CO_2}\}$ the vector of measurements reported by a participant to the broker and by the broker to the queriers, respectively. Such readings are large size alphabets data. So, we apply a quantization step that clusters the values into significant intervals based on the identified thresholds of [22]. Since the number of features in M is important, we opt for the *exchange-distortion* obfuscation technique to generate a set of distorted data $\mathcal{D}$ of a same size as $\mathcal{M}$. That is, we perturb the measurements' vector M by exchanging its elements' values with others in the set $\mathcal{M}$.

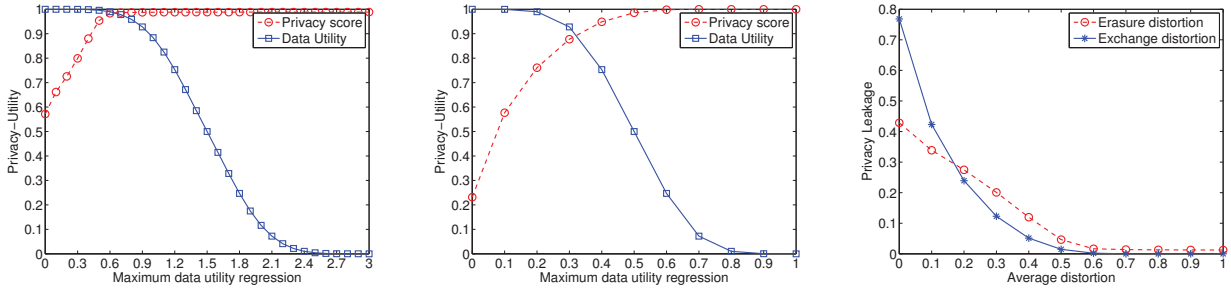
We plot in Figure 3 (a) and (b) the utility-privacy tradeoff. The utility metric is naturally a decreasing function of the data regression level δ which implies the amount of distortion applied to M. That is, the more the data is perturbed, the less useful is. Nevertheless, our method has reached a utility-privacy tradeoff by achieving at least 90% of privacy protection with less than 10% of data-utility loss for both datasets.

On the other hand, we plot in Figure 3 (c) the evolution of the privacy leakage, $I(S, D)$ as function of the average obfuscation level measured by the Hamming distance and we



(a) Privacy-Utility Tradeoff for Dataset1 (b) Privacy-Utility Tradeoff for Dataset2 (c) Privacy leakage-Dataset1 & Dataset2

Fig. 3. The Privacy and Utility metrics for the two datasets of Occupancy



(a) Privacy-Utility Tradeoff for erasure-distortion (b) Privacy-Utility Tradeoff for exchange-distortion (c) Privacy leakage exchange Vs distortion

Fig. 4. The Privacy and Utility metrics for the two distortion applied on GPS data set

compare it for the two datasets. Clearly, the privacy leakage is minimized when data perturbation level increases until we achieve a perfect privacy, $I(S,D)=0$. Also, we observe that, for Dataset1, the perfect privacy requires distorting at least one feature of the measurements vector M whereas an average of 30% distorted data was enough to reach the same privacy level for Dataset2.

This is due to the difference of the data distribution as detailed in Table I. Dataset1 has a balanced distribution of occupancy measurements which makes distinguishing the secret harder while Dataset2 consists of more non occupied-room reported readings. Indeed, this distribution presents the prior knowledge of the broker about the participants' data. Hence, the more balanced the joint distribution $p_{S,M}$ (by reporting different readings) is, the harder to compromise one's private information is.

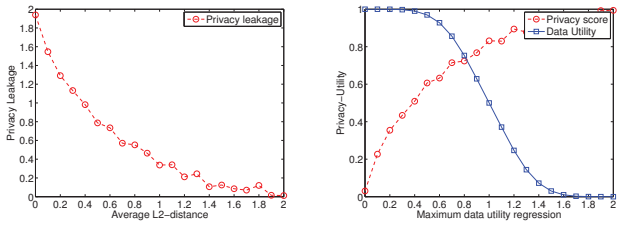
*2) **Obfuscation type impact (scenario 1)**:* We suppose studying here a traffic rating participatory sensing application. We consider as secret S the driving behavior of a participant and by $M=\{speed, R, W\}$ the reported values for the vehicle speed, the traffic and the weather rating, respectively. Similar to the occupancy data, we notice that the reported speed measurements are large size alphabets data and we quantized it by setting three different intervals; $([0,30],[30,50],[\geq 50])km.h^{-1}$. Besides, we vary the type of obfuscation to generate D and we plot the corresponding results in Figure 4.

First, we set the obfuscation technique as the *erasure-distortion*, i.e, we hide one or more features of the measurements vector M. The obfuscation level is quantified by the average number of erasures and varies from 0 to 3. Figure 4(a) shows the achieved privacy and utility values for different data-utility regression levels. We notice that hiding one feature among 60% of the collected data is enough to obtain the maximum privacy level achieved when hiding all data features. Moreover, our method realizes such privacy protection while maintaining more than 95% of data utility.

Further, we set the obfuscation type as the *exchange-distortion* measured by the Hamming distance. Similarly, we show in Figure 4(b) that, differently from the erasure-distortion simulation, the privacy-utility tradeoff is obtained for only 35% of data obfuscated while realizing an important quality of data.

Finally, we compare the impact of varying the obfuscation type on the traffic rating crowdsensing data in Figure 4(c). We observe that an average of 0.1 erasure-distortion data ensures lower privacy leakage values than the exchange-distortion. This confirms that hiding a feature of the measurements vector M has an interesting impact on the privacy risk whereas exchanging it with another value affects mainly the data joint distribution. Nevertheless, by exchanging more values (Average Hamming distance ≥ 0.2), we obtain better privacy levels. Note also that the minimum achieved privacy leakage level by erasure-distortion is $I(S,D)=0.0117$ while the exchange-distortion obfuscation realized a perfect privacy. This highlights the importance of the obfuscation type selection in order to ensure better privacy for similar data regression levels.

*3) **Obfuscating both variables (M,S) impact (scenario 2)**:* Hereafter, we investigate the scenario of Figure 2(b) for location tagged reported temperature measurements. Hence, we study the efficiency of our method while using the marginal distribution of $\tilde{M}=(M,S)$ instead of the joint one. To do



(a) Privacy leakage (b) Privacy-Utility Tradeoff

Fig. 5. The Privacy and Utility metrics for temperature crowdesensing app

so, we first cluster the sensing area into 9 sub-areas and we consider the *exchange-distortion* on the participants' locations. That is, the users may report close by sub-areas tags along with their sensor readings instead of their real positions. Figure 5 shows the realized privacy and utility values for different L2-distance (Euclidean) values.

We observe that the privacy leakage is decreasing slowly and reaches its minimum level for a l2-distance ≥ 2. On the other hand, we achieve a privacy-utility tradeoff by changing 80% of the location data to the cost of minimizing the utility of data to 70%. This slightly more important cost, compared to the other participatory sensing applications, is due to the fact that the location is distorted, however necessary for the accuracy of the reported temperature readings. Indeed, we achieve by our privacy-mechanism a better privacy level for participants, but lower data quality.

VII. Conclusions & Future Work

In this paper, we focus on the privacy-utility tradeoff in participatory sensing systems. This issue is encountered by a participant recruited to report publicly some data to a querier while protecting his private information and respecting the data utility requirements simultaneously. We present an adequate metric to quantify the privacy protection level of participants and consider a data utility threshold set by queriers. Also, we investigate the adversary model and propose to rely on a trusted entity aiming at minimizing the privacy leakage while respecting the data utility constraint. We evaluate our privacy-preserving utility-aware mechanism on different participatory sensing applications based on various real datasets. Results demonstrate the efficiency of our solution for different obfuscation types and data distribution.

For future work, we consider enhancing the performance of our scheme by merging the proposed probabilistic model with other privacy-preserving techniques such as l-diversity or differential privacy.

References

[1] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, "Participatory sensing," in *WSW: Mobile Device Centric Sensor Networks and Applications*, 2006, pp. 117–134.

[2] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, "A survey of mobile phone sensing," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 140–150, 2010.

[3] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *ACM conf. on Embedded Network Sensor Systems*, 2008, pp. 323–336.

[4] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda, "PEIR, the personal environmental impact report, as a platform for participatory sensing systems research," in *Mobile Systems, Applications and Services*, 2009, pp. 55–68.

[5] I. J. Vergara-Laurens, L. G. Jaimes, and M. A. Labrador, "Privacy-preserving mechanisms for crowdsensing: Survey and research challenges," *IEEE Internet of Things Journal*, 2016.

[6] M. Wernke, P. Skvortsov, F. Dürr, and K. Rothermel, "A classification of location privacy attacks and approaches," *Personal Ubiquitous Comput.*, vol. 18, pp. 163–175, 2014.

[7] K. Shilton, "Four billion little brothers?: Privacy, mobile phones, and ubiquitous data collection," *ACM Communications*, vol. 52, pp. 48–53, 2009.

[8] E. D. Cristofaro and C. Soriente, "Participatory privacy: Enabling privacy in participatory sensing," *IEEE Network*, vol. 27, 2013.

[9] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "AnonySense: Privacy-aware people-centric sensing," in *Mobile Systems, Applications and Services*, 2008, pp. 211–224.

[10] Y. Yao, L. T. Yang, and N. N. Xiong, "Anonymity-based privacy-preserving data reporting for participatory sensing," *IEEE Internet of Things Journal*, vol. 2, pp. 381–390, 2015.

[11] R. K. Ganti, N. Pham, Y.-E. Tsai, and T. F. Abdelzaher, "Poolview: Stream privacy for grassroots participatory sensing," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, 2008, pp. 281–294.

[12] D. Christin, A. Reinhardt, S. S. Kanhere, and M. Hollick, "A survey on privacy in mobile participatory sensing applications," *J. Syst. Softw.*, vol. 84, 2011.

[13] J. Shi, R. Zhang, Y. Liu, and Y. Zhang, "Prisense: Privacy-preserving data aggregation in people-centric urban sensing systems," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–9.

[14] H. Jin, L. Su, H. Xiao, and K. Nahrstedt, "Inception: Incentivizing privacy-preserving data aggregation for mobile crowd sensing systems," in *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2016, pp. 341–350.

[15] I. S. Reed, "Information theory and privacy in data banks," in *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, 1973, pp. 581–587.

[16] S. R. L. Sankar and H. V. Poor, "A theory of privacy and utility in databases," in *ArXiv e-prints, Feb. 2011*, 2011.

[17] F. du Pin Calmon and N. Fawaz, "Privacy against statistical inference," in *50th Annual Allerton Conference on Communication, Control, and Computing*, 2012, pp. 1401–1408.

[18] S. Salamatian, A. Zhang, F. du Pin Calmon, S. Bhamidipati, N. Fawaz, B. Kveton, P. Oliveira, and N. Taft, "Managing your private and public data: Bringing down inference attacks against your privacy," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, pp. 1240–1255, 2015.

[19] I.Wagner and D.Eckhoff, "Technical privacy metrics: a systematic survey," in *eprint arXiv:1512.00327*, 2015.

[20] Q.-T. Nguyen-Vuong, Y. Ghamri-Doudane, and N. Agoulmine, "On utility models for access network selection in wireless heterogeneous networks," in *IEEE Network Operations and Management Symposium, NOMS*, 2008, pp. 144–151.

[21] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley-Interscience, 2006.

[22] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and {CO2} measurements using statistical learning models," *Energy and Buildings*, vol. 112, pp. 28 – 39, 2016.

[23] "GO TRACK application," https://play.google.com/store/apps/gotrackfree, accessed: 2016-08-16.

[24] M. O. Cruz, H. Macedo, and A. Guimares, "Grouping similar trajectories for carpooling purposes," in *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, 2015, pp. 234–239.

[25] M. A. Alswailim, H. S. Hassanein, and M. Zulkernine, "CRAWDAD dataset queensu crowd temperature (v. 2015-11-20)," Downloaded from http://crawdad.org/queensu/crowd_temperature/20151120, nov 2015.

Decision-Theoretic Approach to Designing Cyber Resilient Systems

Vineet Mehta
MITRE Corporation

Paul D. Rowe
MITRE Corporation

Gene Lewis
Stanford University

Ashe Magalhaes
Stanford University

Mykel Kochenderfer
Stanford University

***Abstract*—The increasing number of persistent attacks on computing systems has inspired considerable research in cyber resilience solutions. Resilient system designers seek objective approaches to aid in the comparison and selection of effective solutions. Decision theoretic techniques such as Markov decision processes can be leveraged for such comparisons and design decisions. Markov decision processes facilitate examination of uncertainty in system dynamics, diversity of responses, and optimization for operational objectives. This paper proposes a system design approach based in decision theory to achieve effective cyber resilience solutions. The prototypical example of a system with network intrusion detection and host reconstitution is used to illustrate this approach and highlight difficulties designers face due to the non-trivial coupling that may arise between response mechanisms.**

I. Introduction

Recently, the field of cyber resilience has received growing attention as the cyber defense community has pivoted from the pursuit of absolute protection to the exploration of methods for damage mitigation and system recovery. Cyber resilience has been defined as "the ability of cyber systems and cyber-dependent missions to anticipate, continue to operate correctly in the face of, recover from, and evolve to better adapt to advanced cyber threats" [1]; this is a paradigm shift away from the traditional focus of preventing intruders from entering a system. Recent work in cyber resilience has subsequently focused on the study of reconstitution, or actions that allow for the recovery of infected systems back to a normal operating state. Cyber resilient design attempts to blend traditional security measures with proactive strategies and adaptive components to create a system that can both defend and recover from malicious attacks conducted by an adversary.

The design of cyber-resilient systems is a non-trivial task. Even simple system architectures that operate under strong assumptions about adversary behavior are prone to volatile and unpredictable system dynamics due to inherent uncertainty in system inputs and the effects of component interactions. This difficulty is further exacerbated by the number and variety of defensive and resilience solutions available, coupled with diverse organizational objectives, and shifting adversarial tactics. Systematic analysis under a range of operational conditions is necessary for understanding the performance sensitivity of candidate system designs; to perform such an analysis, a flexible framework is needed within which system designs can be quantitatively modeled, evaluated and compared. Such a framework would ease the designer's task of identifying cyber resilient architecture options.

We contribute to the development of this framework by proposing a decision-theoretic modeling paradigm. Specifically, we illustrate the advantages of a Markov Decision Process (MDP) formulation [2], [3] for comparing cyber resilience of a variety of system designs and configurations. Through the use of an illustrative example, we demonstrate how an MDP system model can be used to account for the variety of mission objectives, input uncertainties, and design options in a unified manner. Each MDP formulation entails an optimal policy (a mapping from the system state to a recommended action). We study the sensitivity of the utility of these policies to changes in the adversary-controlled parameters of a model. Our illustrative example suggests how more complex systems can be analyzed through the composition of component models. The facility for composition allows for a unified examination of design options from diverse resilience techniques.

Previous work in the formation of mathematical frameworks of cyber resilience [4], [5] has focused on systems modeled as interdependent service components connected over a graph. Broadly, these approaches represent the system health as a collection of component states, with the system transitioning between states in response to both external influences and actions taken on components. Though these works are important steps in model development, their focus has been on recommending the best actions that return a system to normalcy during operation. There appears to be an absence of frameworks which address system designs with a mixed set of cyber resiliency goals (anticipate, withstand, recover, and evolve) in a unified manner, with the goal of supporting both feature selection during system design and sensible action recommendation during system operation. Our approach suggests a unifying framework within which features contributing to different resilience goals can be compared against each other; a system designer can therefore decide, for example, whether it is better to implement a feature that makes it easier to *withstand* an attack or a feature that reduces the time to *recover* from an attack. Furthermore, the system operator is provided with guidance on how to best employ the resilience features selected by the system designer.

Approved for Public Release; Distribution Unlimited. Case Number 16-2931. This technical data was produced for the U. S. Government under Contract No. FA8702-16-C-0001, and is subject to the Rights in Technical Data-Noncommercial Items Clause DFARS 252.227-7013 (JUN 2013).

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

Concretely, the main goal of this paper is to illustrate the use of decision theory for a quantitative analysis of resilience as it relates to the security of computing systems. In support of this goal we provide:

- A formulation of systems analysis for cyber resilience within the context of a decision-theoretic framework.
- An application of this formulation to an example system with a specific set of resilience response mechanisms: network intrusion detection and host reconstitution.
- An illustration of design options analysis using this framework in comparing the resilience afforded by families of systems.

The paper is structured as follows. Section II describes previous work in support of cyber resilience. Section III describes the use of a POMDP framework for examining resilience for security of computing systems. Section IV introduces the formulation of an example system as an MDP model. Section V describes experiments aimed at comparing the resilience of different system configurations. Section VI compares results obtained from the calculation of optimal policies for a variety of system configurations, with different resilience features, and adversarial activity. The results illustrate how a designer can establish which resilience features are most effective. Section VII closes the paper with a discussion.

II. Related Work

Previous work in support of cyber resilience has focused on the development of a set of goals and a framework within which these goals may be pursued; examples include the Cyber Resiliency Engineering Framework (CREF) proposed by Bodeau et. al [1], [6] and the CERT Resilience Management Model [7]. Under these frameworks, generic cyber resilience goals are defined by a cyber system's ability to anticipate adversarial attacks, continue operation under distress, recover operation after sustaining an attack, and utilize previous encounters to better prepare for future attacks.

A. Mathematical Formulations of Cyber Resilience

Work in developing a mathematical formulation of modern cyber resilience can be traced to Ramuhalli *et. al* [4], who proposed a graph-based cyber resilience model for the study of reconstitution actions. They model their system as a connected network with nodes that host a set of dependent services; the system maintains a state related to its health, with the ability to transition between sets of fully operational, marginally operational, and compromised states in response to natural causes or the actions of an adversary. Reconstitution is treated as an optimization problem over the resulting graph, balancing the value of continued operation against the costs of reconstitution; resilience is achieved through complete reconstitution of lost services.

Choudhary *et. al* [5] expand this graph-based framework by modeling systems of interest as a collection of interdependent system components. Here each system component is modeled as a subgraph and assumed to provide a distinct service while being susceptible to a particular type of attack; the system as a whole is then modeled as the total set of connected sub-graphs. Resilience is achieved by selecting a subset of pre-defined mitigation actions. Choudhary *et. al* also provide an action recommendation engine that takes as input the current system state and provides as output an action recommendation. The recommendation engine compiles sufficient statistics related to performance and security measures of interest, and then solves an optimization problem to provide a sensible advisory. In contrast, our approach provides a recommendation engine in the form of a policy; advantages include the ability to examine policies without the need for a running system during design-time, and fast real-time action lookup for a given system state when in deployment.

B. Decision-Theoretic Approaches to Cyber Security

One well-explored approach to handling the inherent uncertainty in adversarial cyber attacks is to adopt a learning-based decision-theoretic model. These models are capable of leveraging data to derive optimal actions based on the state of the system. To the best of our knowledge, such models have not been employed as generalized frameworks in the design or orchestration of resilience for secure systems. Decision-theoretic approaches have been applied extensively to the adaptation of detection models in intrusion detection systems (IDS); examples include a multi-agent reinforcement learning approach considered by Servin and Kudenko [8], where agents cooperate to detect intrusions, and a partially observable Markov decision process (POMDP) formulation provided by Lane [9], where incoming partially labeled data is assumed to be generated by a normal user or an attacker. They have also been applied in the selection of suitable policies for guiding the operation of intrusion detection and response systems, as demonstrated in the analysis provided by Kriedl [10].

The mention of previous IDS related efforts here is motivated only by their connection to our own illustrative example of applying MDP models for design and operation of resilient secure systems. Our main aim, however, is to demonstrate the general utility of such models in providing a unified framework for design of resilient systems, and their subsequent operation. Since our illustrative example considers IDS models, we provide some details on the IDS attributes highlighted in our example. Despite the popularity of IDS systems, the inaccuracy of detections and large volume of false alarms (in some cases nearly 70% of all alerts) can limit effectiveness [11]. IDS systems also have difficulty keeping up with incoming network traffic which exceeds rates of a few megabits per second. Studies with Snort, an open source intrusion detection system, have reported dropped packets proportions of more than 70% [12] due to network packet throughput limitations.

In order to address this limitation, Bakhoum [13] disproved the common notion that a strong IDS must inspect every packet, showing instead that network security can be maintained while inspecting only a subset of incoming packets. However, such a system is still vulnerable to allowing hostile packets past the IDS and into the host, where an infection can

cause critical computing components to malfunction indefinitely until taken offline for maintenance. This result suggests that a controller taking actions over both the IDS and the system host is needed to achieve the cyber resilience goals of operation under distress and recovery after a sustained attack.

In order to highlight the use of decision theory for selection of features during design, we include a host capable of reconstitution as a separate component, connected to the IDS by a communications link. This allows us to illustrate the modular composition of a system from components and to examine non-intuitive responses that can arise from component interactions under a range of adversarial scenarios. We consider how different design choices augment the space of actions available for an operational system, and how the addition of such actions affects overall cyber resilience. We also illustrate the flexibility of decision theory based design for assessing resilience of alternate operational objectives through appropriate constructions of reward structures. This ability to assess alternate designs in the context of operational objectives allows for discriminating the effectiveness of resilience features. For instance, an IDS, which is often viewed as a valuable feature, might prove to be of little value for certain designs and operational scenarios.

III. MDP Framework for Cyber Resilience

When we talk of the resilience of a system we refer to the stability of that system's performance under shocks of various types. The performance is measured by some objective function f, and the shocks are represented as changes in the different inputs to f. There is no single right choice of objective function to measure a system's performance; in fact, there may be many such performance measures of interest for a given system. The system may similarly be more or less stable as different inputs vary, so we might be more interested in resilience to changes of particular inputs. In cyber resilience we are primarily interested in the stability of an objective function f under changes to inputs that are under the control or influence of a cyber adversary.

A. Decisions in Cyber Resilient System Design

An important question in system design is whether adding a given feature to a system will make it more resilient, and if so by how much. Similarly, a system designer may be constrained to add only one of two extra features, and so would like to know which one has a greater effect on the system's resilience. It is common for such features to provide system administrators with the ability to adaptively respond to adversary activity. An important aspect in increasing the resilience is to choose good policies for response. This suggests a connection between cyber resilience and decision theory.

To demonstrate this connection consider a standalone system that is connected to an external network on which an adversary may reside. The adversary can inject malicious messages into the stream of incoming messages; these malicious messages have some chance of infecting the system. The system incurs some cost for being infected. Common methods for making the system more resilient to incoming malicious messages include (a) employing an intrusion detection system (IDS) to filter out messages matching known malware signatures and (b) periodically resetting or reconstituting the system to a known good state. Both of these resilience mechanisms may introduce a new cost of their own. If message inspection cannot keep up with the rate of incoming messages, dropped packets will reduce the performance of the system. Similarly, resetting the system might entail taking it offline for some time, which itself may carry some cost.

These feature interactions increase the complexity of the system; by offering more actions to take in a given state (e.g. inspect a message vs. let it pass through uninspected), the variety of system behaviors depending on the policy chosen is correspondingly higher. Given the additional cost of some of the new actions, there is a chance that there are new policies available to deploy that perform worse than the original system. This suggests that techniques from decision theory can bring value to the problem of cyber resilience.

B. Markov Decision Process Formulation

A popular model used in decision theory is a (fully observable) Markov Decision Process (MDP) or its partially observable variant (POMDP) [2], [3]. In general a POMDP is described by the tuple $F_\Omega = (S, A, P_S, R, Z, P_Z, \Omega)$ where:

Parameter	
S	Set of states
A	Set of actions
P_S	Probability distribution over resulting states
R	Rewards associated with states and actions
Z	Set of observations
P_Z	Probability distribution over resulting observations
Ω	Set of system parameters

A POMDP model of the system described above can provide us with parameterized, quantitative dynamics. The set Ω can encode probabilistic parameters such as false positive and negative rates of the IDS, or the likelihood that a malicious message will infect the system. We can therefore investigate and compare the system's performance under changes to the underlying parameters, and consider variations associated with uncertainty of these parameters. Likewise, we can compare the system behavior with and without the two resilience mechanisms we described. Since such models admit solutions (i.e. policies that optimize the utility function) it is sensible to use the expected utility of the system under this optimal policy as a performance measure. That is, $\bar{V}_* = \mathbb{E}[V_*(s)]$ acts as a scalar metric for characterizing the performance of a system modeled by F_Ω. In this way the resilience of the system roughly corresponds to the stability of $\bar{V}_*$ under changes to the underlying parameters Ω.

IV. Exemplar Problem Formulation

In the sequel, we illustrate the practicality and power of our MDP cyber resilience framework via an examination of a prototypical system.

A. Model Overview

Our example model features an intrusion detection system protecting a host, as illustrated in Figure 1. The actions of the intrusion detection system and the host are guided by a controller. The action spaces over the IDS and host system is specified as $a_I \in A_I = \{\texttt{inspect}, \texttt{pass}\}$ and $a_H \in A_H = \{\texttt{wait}, \texttt{reset}\}$. Messages generated in the extranet

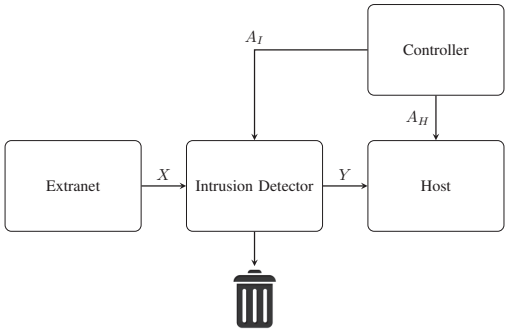


Fig. 1. Intrusion Detection and Response System

first pass through the intrusion detection system before arriving at the host system. A message's infection state is given by the random variable $x \in X = \{\texttt{benign}, \texttt{malicious}\}$. An incoming message is `malicious` with probability λ. The intrusion detection system can either pass the input message or drop it, guided by the controller actions $a_I \in A_I$. The random variable Y encodes the message output states as: $y \in Y = \{\texttt{benign}, \texttt{null}, \texttt{malicious}\}$. If a message successfully passes through the intrusion detection system it's outputs state y corresponds to that of the input x. Otherwise the output has state $y = \texttt{null}$. In the following we detail the internal operation of the intrusion detection system and the host, as influenced by the actions (a_I, a_H).

B. Intrusion Detection System Model

The state transition diagram in Figure 2 describes the operation of the intrusion detection system.

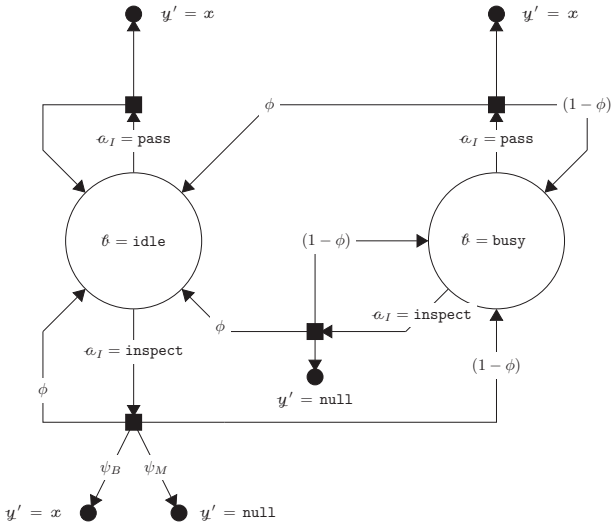


Fig. 2. Intrusion detector state transition diagram

The random variable B encodes this state as $b \in B = \{\texttt{busy}, \texttt{idle}\}$. The intrusion detection system is in the $b = \texttt{idle}$ state if it is not inspecting a message for malicious content, otherwise its state is $b = \texttt{busy}$. We employ the Markov assumption to model transitions between states. Therefore transition to a future state $B' = b'$ depends only on the current state $B = b$ and the action $A_I = a_I$. The intrusion detection system simply passes an input message without inspection if the current action is $a_I = \texttt{pass}$. Under this action, the output of the intrusion detection system at the next time step y' corresponds to the infection state of x. In this instance, the intrusion detection system returns to the `idle` state with certainty. If the intrusion detection system starts in the `busy` state, it transitions to the `idle` state with probability ϕ. The value of parameter ϕ governs the mean holding time in the `busy` state, and lower values of ϕ are interpreted as settings where the system performs deeper message inspection. When the system starts in the `idle` state, under the action $a_I = \texttt{inspect}$, it transitions to the `busy` state with probability $(1 - \phi)$. In our model we make the simplifying assumption that the intrusion detection system provides an output which reflects the outcome of its inspection in the next time step, independent of the busy time induced by `inspect` action.

We model the intrusion detection system as an imperfect inspection system that can produces false positive and false negative results. The false positive and false negative probabilities for malicious message detection are specified by the parameters β_{FP} and β_{FN} respectively. When the intrusion detection system inspects a message, the message is classified as `malicious` with probability $\psi_M = \lambda(1-\beta_{FN}) + (1-\lambda)\beta_{FP}$ and `benign` with probability $\psi_B = \lambda\beta_{FP} + (1-\lambda)(1-\beta_{FN})$. When the intrusion detection system is in the `idle` state and it classifies a message as `malicious`, that message is always dropped and $y = \texttt{null}$ for the next time step. While the intrusion detection system is in the `busy` state, the action to `inspect` also results in that message being discarded and $y = \texttt{null}$. This behavior models computational limits typical of intrusion detection systems when attempting to keep pace under high traffic loads.

C. Host System Model

Figure 3 illustrates operational dynamics of the host, which are also modeled by a Markov Decision Process. The host system is described by two primary state variables that characterize the operational and infection states. The host has two operational states $w \in W = \{\texttt{full}, \texttt{reset}\}$. In the `full` state, the host is capable of fully processing incoming messages. The `reset` state provides an opportunity for the host to either avoid processing potentially malicious messages or allow for repairs if the host was previously infected. The state variable $h \in H = \{\texttt{clean}, \texttt{infected}\}$ specifies whether the host system is infected or not.

The state of the host system is influenced by the input message state y and action a_H. If the action state is `wait` and incoming message is not `malicious`, the system remains in the $(\texttt{full}, \texttt{clean})$ state. We allow the host to have some

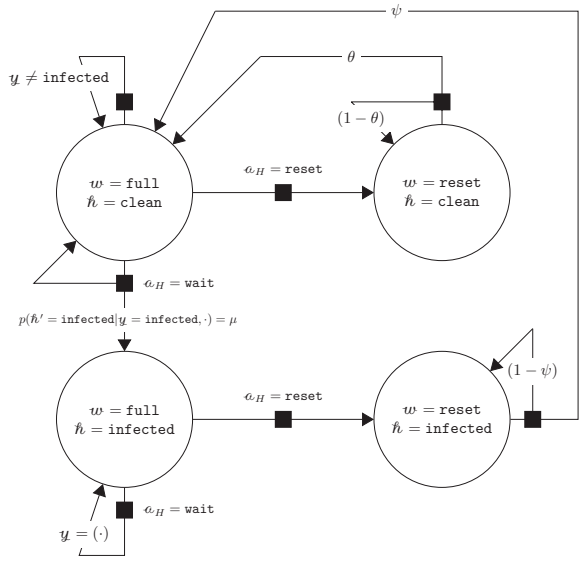


Fig. 3. Host state transition diagram

intrinsic resistance to `malicious` incoming messages. This ability is parameterized by μ: the probability of infection in the event of a `malicious` message. Incoming `malicious` messages cannot further infect a host, when the host state is (`full`, `infected`). The `reset` action forces a transition to $w =$ `reset` state. Our model allows different mean holding times in the `reset` state for the infection states `clean` and `infected`, through separate restoration probabilities θ and ψ.

D. Stochastic Model Dynamics

In order to examine the stochastic dynamics of our model as a whole we must consider the joint state of the system $S = (Y, X, W, H, B)$, under the influence of aggregate actions $A = \{A_I, A_H\}$. The transition probability from current state S to future state S' conditioned on action A is factored as:

$$P(S' \mid S, A) = P(Y' \mid X, B, A_I)P(X') \times P(W', H' \mid W, H, Y, A_H)P(B' \mid B, A_I)$$

The probability of a malicious incoming message is independent of other state variables, and given as a Bernoulli distribution with parameter λ:

$$P(X') = \text{Ber}(X' = x' \mid \lambda) = \lambda^{x'}(1-\lambda)^{1-x'}$$

Selected component probabilities are provided in Tables I and II. For brevity of presentation, we have only specified the non-zero probabilities in these tables. The probability function $P(Y' \mid X, B, A_I)$ is specified in Table I. This function models the output of our intrusion detection system at the next time step as dependent only on the input message state, the intrusion detection system busy state, and the controller action. The probability function $P(B'|B, A_I)$ is specified in Table II. This function models the intrusion detection system's next busy state B' as depending only on the current busy state and controller action.

TABLE I
INTRUSION DETECTOR OUTPUT TRANSITION PROBABILITY: $P(Y' \mid X, B, A_I)$

X	B	A_I	Y'	$P(Y' \mid X, B, A_I)$
benign	idle	inspect	benign	$1-\beta_{\text{FP}}$
benign	idle	inspect	null	β_{FP}
malicious	idle	inspect	null	$1-\beta_{\text{FN}}$
malicious	idle	inspect	malicious	β_{FN}
benign	idle	pass	benign	1
malicious	idle	pass	malicious	1
benign	busy	pass	benign	1
malicious	busy	pass	malicious	1
benign, malicious	busy	inspect	malicious	1

TABLE II
INTRUSION DETECTOR OPERATION TRANSITION PROBABILITY: $P(B' \mid B, A_I)$

B	A_I	B'	$P(B' \mid B, A_I)$
idle	pass	idle	1
idle	inspect	busy	$1-\phi$
idle	inspect	idle	ϕ
busy	inspect, pass	busy	$1-\phi$
busy	inspect, pass	idle	ϕ

The transitions of the aggregate host states are given by the probability function $P(W', H' \mid W, H, Y, A_H)$. This probability can be written in a fashion similar the other component probabilities. The form of this probability highlights that the host state is driven primarily by the message state Y and action A_H.

Our aim is to find optimal policies $\pi_*(s)$ for the MDP $(S, A, P, R \mid \Omega)$. We have specified the states S and actions A, as well as the transition probability P for the parameter tuple $\Omega = (\lambda, \mu, \phi, \theta, \psi, \beta_{\text{FP}}, \beta_{\text{FN}})$. The reward structure R remains to be specified for a complete description of the MDP. In this paper we will focus on the case where the rewards depend only on the aggregate state $S_H = (W, H, Y)$ of the host system. The rewards for the host states are specified in Table III.

TABLE III
REWARD STRUCTURE: $R^{(\alpha)} = R(S_H)$

W	H	Y	R
full	clean	benign	1.0
full	clean	null	$1.0 + \alpha\rho_+$
full	clean	malicious	$1.0 + \alpha\rho_-$
full	infected	benign, null, malicious	ρ
reset	clean, infected	benign, null, malicious	2ρ

In general the future-discounted expected return of applying a policy π when starting in state $S_0 = s$ is given by the value function:

$$V_\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | S_0 = s\right] \quad (1)$$

The discount factor γ is constrained such that $\gamma \in [0, 1]$. A solution to our MDP involves finding the optimal policy which

TABLE IV
FOUR SUBSYSTEMS

	A_H	
A_I	{wait}	{wait, reset}
{pass}	Σ_0	Σ_2
{pass, inspect}	Σ_1	Σ_3

maximizes the value function across all states. This objective is formally expressed as:

$$\pi_*(s) = \arg\max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \quad (2)$$

A scalar metric for characterizing the resiliency of our system is the maximal expected utility $\bar{V}_* = \mathbb{E}[V_\pi | \pi = \pi_*]$. This expected value is calculated using the stationary state probability distribution $\varphi(s)$. The stationary distribution can be computed as the normalized left eigenvector for the unit eigenvalue of the transition probability matrix under optimal policy.

V. EXPERIMENTS

Our MDP model contains several sub-models that arise by restricting the action space $A = A_I \times A_H$. In particular, consider the four systems that arise by restricting the action spaces as described in Table IV. These systems are viewed as candidate design proposals with different resilience features.

Σ_0 represents a system without an IDS and with no ability to reset or reconstitute the system if it becomes infected. By adding the ability to inspect messages, Σ_1 represents a system with an IDS but with no ability to reset. Σ_2 contains no IDS, and so has no ability to inspect messages, but a reset action can restore it to a full and clean state. Finally, Σ_3 is the full system described in the previous section. Viewed as MDPs, these systems share the same underlying transition probabilities given a set of parameters Ω, but the restriction of the action spaces reduces the set of policies we can choose from when solving the MDP.

As discussed in Section III, we choose to measure the performance of these systems using the maximal expected utility under the optimal policy π_*, which we denote $\bar{V}_*$. Of course, $\bar{V}_*$ really depends on the underlying system (i.e. the set of available policies), on the parameter set $\Omega = (\lambda, \mu, \phi, \theta, \psi, \beta_{\text{FN}}, \beta_{\text{FP}})$, and on the discount factor γ used to discount the value of future rewards. We also might consider each system under several reward structures parameterized by ρ as described in the previous section. Thus, if we let Ω^+ denote the augmented set of parameters $\Omega \cup \{\gamma, \rho\}$ we may write $\bar{V}^i_*(\Omega^+)$ to denote the resulting maximal expected utility under the optimal policy for system Σ_i. Beyond comparing the performance (i.e. the values of $\bar{V}^i_*(\Omega^+)$) for the various systems Σ_i, we are also interested in the resilience of the systems. We want to know which systems are more sensitive to changes in the parameters Ω^+. We can thus examine the effect that each of the functional mechanisms (IDS and reset capability) has on both performance and resilience. Through this type of examination a designer can assess which features should be included in the objective design.

Ω^+ is a large domain, and some of the parameters are more likely to change over time than others. From the viewpoint of cyber resilience, we are most interested in detecting sensitivity to changes in parameters that are likely to be under the influence of an adversary. By adjusting the number of malicious packets sent to the system, the adversary has a strong influence on the λ parameter. Similarly, an adversary may have the ability to adjust the μ parameter governing the likelihood of compromise given a malicious message. For example, the adversary could send messages tailored to the particular system that are more likely to cause infections than generic attacks.

We therefore fix values for most of the parameters of Ω^+ while varying the values of λ and μ. The fixed parameters are $\phi = \theta = \psi = 0.9$ and $\beta_{\text{FP}} = \beta_{\text{FN}} = 0.01$. This assumes a system that is likely to complete packet inspection in a single time step, and is equally likely to recover from reset in a single time step (whether or not it was infected at the time of reset). The IDS is assumed to have very high accuracy with very low false positive and negative rates. We also set $\gamma = 0.95$ indicating that we only slightly discount the value of future states. Finally, we fix the parameter for the rewards to be $\rho = -1.5$.

Fixing these parameters means that $\bar{V}^i_*(\Omega^+)$ defines a surface over the unit square as λ and μ each vary over $[0,1]$. We compute and plot various cross sections of this surface. In particular, we consider cross sections for $\mu \in \{0, 0.25, 0.5, 0.75\}$ letting λ range over $[0,1]$ in increments of 0.01. The next section contains the results.

VI. RESULTS

In this section we present results for the experiments described is Section V. Our experiments cover a limited parameter space: $\lambda \in [0,1]$ and $\mu \in [0,1]$. The remaining parameters in the set Ω have values: $\phi = 0.9$, $\theta = 0.9$, $\psi = 0.9$, $\beta_{\text{FP}} = 0.01$, $\beta_{\text{FN}} = 0.01$. These parameters characterize a system that can quickly recover from `busy` and `reset` states. This system is also fairly accurate in its classification of inspected incoming messages. We will examine the resiliency of this system for two different reward structures: $R_H^{(0)}$ and $R_H^{(1)}$ for reward parameters $\rho_+ = -1.75$, $\rho_- = -2.75$, and $\rho = -1.5$.

A. Baseline System

We start by considering the system Σ_0 with the reward structure $R_H^{(0)}$. This system has a restricted action space $(A_I, A_H) = (\{\texttt{pass}\}, \{\texttt{wait}\})$. The reward structure $R_H^{(0)}$ favors the system's occupancy of state $(w,h) = (\texttt{full}, \texttt{clean})$. In the absence of any incoming malicious messages ($\lambda = 0$), the host system is always in this state, with expected utility $\bar{V}_* = 1/(1-\gamma)$. This value of expected utility serves as an upper bound on the performance the system can achieve. For non-zero probability of malicious incoming messages ($\lambda > 0$) the host system is always in state $(w,h) = (\texttt{full}, \texttt{infected})$,

with expected utility $\bar{V}_* = \rho/(1-\gamma)$. We note that the expected utility remains constant for values of $\lambda > 0$, because the reward structure favors the state $(w, h) = (\texttt{full}, \texttt{infected})$ irrespective of the value y. Due to the low utility value, this baseline system may be viewed as a poor resilience candidate.

B. Adding an IDS

System Σ_1 adds an intrusion detection capability whose objective is to intercept malicious messages before they reach the host system. However the host in this system lacks the ability to `reset` if it is infected. The presence of an intrusion detection system provides some protection by dropping malicious messages. However, some malicious messages do ultimately get past the intrusion detection system and successfully infect the host. Without the benefit of a `reset`, the host eventually enters the state $(w, h) = (\texttt{full}, \texttt{infected})$, which serves as an absorbing state for the system for $\lambda > 0$. Thus the expected utility for this system has the same behavior as that for system Σ_0, for reward structure $R_H^{(0)}$. This result appropriately suggests that little value is derived by adding an intrusion detection system, without a host based mitigation mechanism, based on long-term (stationary) behavior of the system.

C. Adding Reconstitutive Actions

Now we consider the systems Σ_2 and Σ_3, which include hosts that can `reset` in order to mitigate the effects of malicious messages. For the reward structure $R_H^{(0)}$, the intrusion detection system has been shown to add little value. We therefore expect benefit under this reward structure to be derived mainly from mitigation mechanisms at the host. Computed results of the expected utility under optimal policy are consistent with this intuition. The expected utility curves for system Σ_3 are shown in Figure 4 over a range of values for (λ, μ). The expected utility $\bar{V}_*$ reaches its maximum value

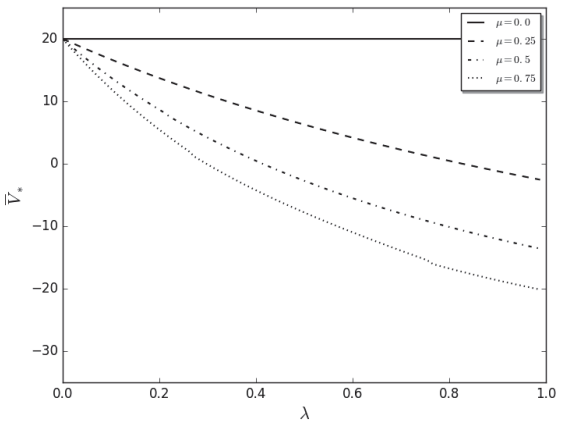


Fig. 4. Expected Utility for System Σ_3 with Reward Structure $R_H^{(0)}$

$(1/(1-\gamma) = 20)$ in the absence of incoming malicious messages ($\lambda = 0$), or when malicious messages have no impact on the host ($\mu = 0$). The expected utility diminishes quickly with increases in the probability of malicious messages (λ) and the probability of host infection (μ). The expected utility approaches its minimum value $(\rho/(1-\gamma) = -30)$ as $(\lambda, \mu) \to (1.0, 1.0)$.

With reward structure $R_H^{(0)}$ it is difficult to see the added benefit provided by an intrusion detection system, working in cooperation with host-based mitigation mechanisms. We can reveal this benefit by considering a modified reward structure $R_H^{(1)}$. In this modified structure we assign higher rewards for the absence of malicious passed messages, along with an uninfected and fully functioning host system (i.e. $y \in \{\texttt{benign}, \texttt{null}\}$ and $(w, h) = (\texttt{full}, \texttt{clean})$ respectively). The benefit of cooperative operation of an intrusion detection system with host mitigation mechanisms is highlighted by comparing the expected utility of systems Σ_2 and Σ_3 with the modified reward structure $R_H^{(1)}$. Results for system Σ_2 are shown in Figure 5. We note that this system lacks an intrusion

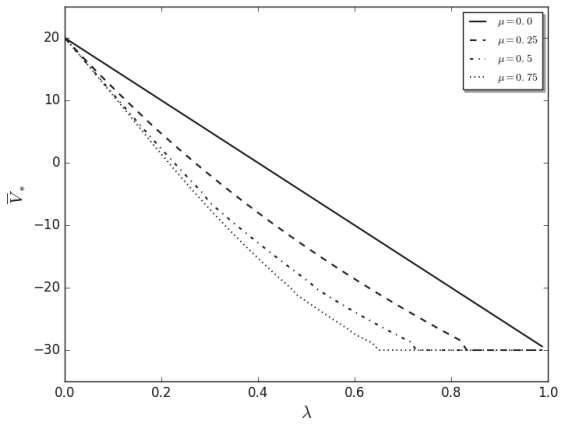


Fig. 5. Expected Utility for System Σ_2 with Reward Structure $R_H^{(1)}$

detection capability, but does possess host-based reconstitution. The observed monotonic reduction in this system's expected utility with increases in probability of malicious messages is consistent with the new reward structure, which favors the absence of such messages even for $\mu = 0$. A further decrease in utility is observed as the susceptibility of the host to `malicious` messages increases ($\mu > 0$). These results suggest the reset feature to be a valuable resilience design option for the system.

D. Combining Reconstitutive Actions with an IDS

The results for system Σ_3, which augments Σ_2 with an intrusion detection capability, are presented in Figure 6. The difference in resiliency offered by systems Σ_2 and Σ_3 can be understood by examining the plots in Figures 5 and 6 for corresponding values of μ. We recall that the intrusion detection system offers little benefit on its own. However, Figures 5 and 6 now reveal that the addition of intrusion detection capabilities provides an increase in expected utility. The cooperative operation of intrusion detection and host based mitigation capabilities is particularly evident for higher probabilities of incoming malicious messages (λ), and high values of host susceptibility ($\mu > 0.5$). It is interesting to

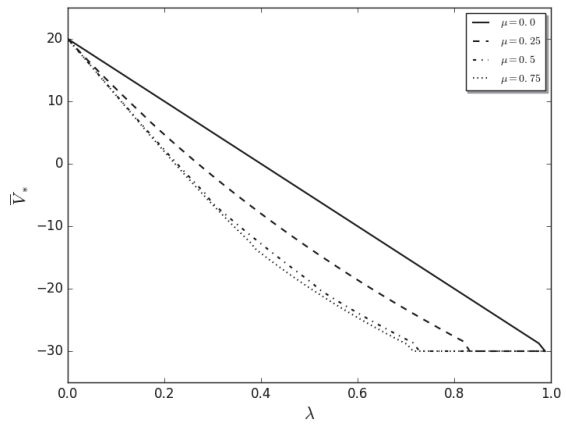


Fig. 6. Expected Utility for System Σ_3 with Reward Structure $R_H^{(1)}$

note that, for lower host susceptibilites, the host based `reset` mechanism offers the *best* mitigation to adversarial activity. As the host susceptibility increases, the intrusion detection system provides supplementary assistance against adversary attacks.

We note that the results shown in Figures 4-6 align well with our notion that resilience is connected to both the utility function's value and sensitivity (given by slope). As an example of this notion, note that we do not consider system Σ_0 resilient, despite the fact $\bar{V}_*$ is constant for all values of $\lambda > 0$; this is due to the utility over $\lambda > 0$ being at a minimum. These results highlight the value of our approach by revealing scenarios in which the inclusion of an IDS improves resilience.

VII. Conclusions

In this paper we have proposed decision theory as a suitably flexible framework for examining cyber resilience of computing systems. We have advocated the use of MDPs (or the more general variant POMDPs) for modeling a range of system designs, while incorporating uncertainty in system dynamics. MDPs also provide a flexible way of assigning value to desired system behaviors using rewards and a means of making optimal decisions that support resiliency goals.

In order to make our proposal concrete, we have chosen to analyze commonly used resiliency mechanisms: an intrusion detection system and host-based system reconstitution. The overall system has been modeled as a MDP. We have taken care however to treat the intrusion detection system and the host as separate components, with distinct state transition models. The coupling between the two components is via the messages that are passed from one to the other. Our model system has a number of tuning parameters, which facilitate the examination of a variety of behaviors and responses. However, we have chosen to focus on parameters that have a dominant connection with adversary behavior: the `malicious` message probability (λ) and the host infection probability (μ). Using this model we have illustrated how a comparative analysis of design alternatives can be formulated by examining different options for the action space and reward structure.

A quantitative comparative analysis of resiliency for a set of design alternatives has been performed for a common base system model that includes an IDS and host-based reconstitution capability. In this analysis we have selected reward structures that focus on the host's behavior, while allowing the coupling between the IDS and host to drive optimal actions for both components. The expected utility $\bar{V}_*$, derived from stationary probabilities based on optimal actions, has served as a scalar metric for understanding the resiliency of various design alternatives. We note that this particular metric is focused on assessing the defender's ability to weather adversarial activity.

In this paper we have examined resiliency from the perspective of the defender. For instance have not developed a metric that characterizes the amount of effort expended by an adversary to successfully impact system function. We have also assumed that state information about adversarial actions (`malicious` messages), and host infection state are known. The ability to jointly optimize actions of the IDS and host is also assumed. A natural extension of our work would seek to relax these assumptions.

References

[1] D. Bodeau, R. Graubart, W. Heinbockel, and E. Laderman, "Cyber resiliency engineering aid—the updated cyber resiliency engineering framework and guidance on applying cyber resiliency techniques," MITRE Corporation, Tech. Rep. MTR140499R1, 2015.

[2] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99–134, 1998.

[3] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[4] P. Ramuhalli, M. Halappanavar, J. Coble, and M. Dixit, "Towards a theory of autonomous reconstitution of compromised cyber-systems," in *IEEE Conference on Technologies for Homeland Security*, 2013.

[5] S. Choudhary, L. Rodriguez, D. Curtis, K. Oler, P. Nordquist, P. Chen, and I. Ray, "Action recommendation for cyber resilience," in *Workshop on Automated Decision Making for Active Cyber Defense*, 2015.

[6] D. Bodeau, R. Graubart, and E. Laderman, "Cyber resiliency engineering overview of the architectural assessment process," *Procedia Computer Science*, vol. 28, pp. 838–847, 2014.

[7] R. Caralli, J. Allen, P. Curtis, D. White, and L. Young, "Cert® resilience management model," Software Engineering Institute, Tech. Rep., 2010.

[8] A. Servin and D. Kudenko, "Multi-agent reinforcement learning for intrusion detection: A case study and evaluation," in *Proceedings of the 6th German Conference on Multiagent System Technologies*, 2008.

[9] T. Lane, "A decision-theoritic, semi-supervised model for intrusion detection," in *Machine Learning and Data Mining for Computer Security*, 2006.

[10] O. Kreidl, "Analysis of a Markov decision process model for intrusion tolerance," in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, 2010.

[11] G. Tjhai, M. Papadaki, S. Furnell, and N. Clarke, "The problem of false alarms: Evaluation with Snort and DARPA 1999 dataset," in *International Conference on Trust, Privacy and Security in Digital Business*, 2008.

[12] F. Aleserhani, M. Akhlaq, I. Awan, J. Mellor, A. Cullen, and P. Mirchandani, "Evaluating intrusion detection systems in high speed networks," in *International Conference on Information Assurance and Security*, 2009.

[13] E. Bakhoum, "Intrusion detection model based on selective packet sampling," *EURASIP Journal on Information Security*, 2011.

The Blockchain Anomaly

Christopher Natoli
NICTA/Data61-CSIRO
University of Sydney
chistopher.natoli@sydney.edu.au

Vincent Gramoli
NICTA/Data61-CSIRO
University of Sydney
vincent.gramoli@sydney.edu.au

***Abstract*—Most popular blockchain solutions rely on proof-of-work to guarantee that participants reach consensus on a unique block per index of the chain. As consensus is impossible in the general case, it seems that these blockchain systems require messages are delivered fast and no participant mines faster than the crowd. To date, no experimental settings have however been proposed to demonstrate this hypothesis.**

In this paper, we identify conditions under which these blockchain systems fail to ensure consensus and present a reproducible execution on our Ethereum private chain. To this end, we introduce the *Blockchain Anomaly*, the impossibility for the blockchain to guarantee that a committed transaction is not abortable. This anomaly may translate into dramatic consequences for the user of proof-of-work blockchains. Named after the infamous Paxos anomaly, this anomaly makes dependent transactions, like "Bob sends money to Carole after he received money from Alice" impossible and may lead to double spending. We also explain how the anomaly differs from a 51-percent attack and how one could avoid it by adapting the Ethereum implementation or by exploiting smart contracts.

Keywords: Ethereum; Paxos anomaly; smart contract

I. Introduction

Mainstream public blockchain systems, like Bitcoin [1] and Ethereum [2], require to reach consensus on the Internet despite the presence of malicious participants and possible congestions that delay messages. Yet, it is impossible for a distributed system including a faulty process to reach consensus if messages may not be delivered within a bounded time [3]. This raises interesting questions about the properties ensured by blockchains. Foundational consensus algorithms [4] were proposed to never reach a decision in case of arbitrary message delays, but to respond only correctly if ever. Surprisingly, these blockchain systems adopt a different approach, sometimes responding incorrectly, especially when delays occur [5]. These few last years, the concept of *private chain* gained traction for its ability to offer blockchain among multiple companies in a private, controlled environment. The R3 consortium is currently running an Ethereum private chain with more than 45 banks worlwide.[1] To understand the limitations of consensus and its potential consequences in the context of private chains, we deployed our own private chain and stress-tested the systems in corner-case situations.

In this paper, we present the *Blockchain anomaly*, a new problem named after the Paxos anomaly [6], [7], [8], that prevents Bob from executing a transaction based on the current state of the blockchain. In particular, we identified a complex scenario where the agreement on the state of the blockchain is not sufficient to guarantee immutability of the chain. This anomaly can lead to dramatic consequences, like the loss of virtual assets or a double-spending attack. We also show that some *smart contracts*, expressive code snippets that help defining how virtual assets can be owned and exchanged in the system, may suffer from the Blockchain anomaly. These results confirm the risk of using a blockchain in a private context without understanding its complex design features, which also confirms the need for solid research foundations [9]. We terminate this paper by providing the source code of a more complex smart contract that can circumvent a particular example of the Blockchain anomaly.

Most blockchain systems track a transaction by including it in a block that gets mined before being appended to the chain of existing blocks, hence called *blockchain*. The consensus algorithm guarantees a total order on these blocks, so that the chain does not end up being a tree. This process is actually executed speculatively in that multiple new blocks can be appended transiently to the last block of the chain—a transient branching process known as a *fork*. Once the fork is discovered, meaning that the participants learn about its branches, the "longest" (i.e., heaviest in Ethereum or deepest in Bitcoin) branch is adopted as the valid one. Blockchain systems usually assume that forks can grow up to some limited depth, as extending a branch requires to solve a cryptopuzzle that boils down to computing for a long time during which one gets likely notified of the longest chain. Bitcoin recommends six blocks to be mined after a transaction is issued to consider the transaction accepted by the system. Similarly, Ethereum states that five to eleven more blocks should be appended after a block for it to be accepted [2].

However, consensus cannot be solved in the general case. In particular, foundational results of distributed computing indicate that consensus cannot be reached if there is no upper-bound on the time for a message to be delivered and if some participant may fail [3]. Consensus is usually expressed in three properties: *agreement* indicating that if two non-faulty participants decide they decide on the same block, *validity* indicating that the decided block should be one of the blocks that were proposed and *termination* indicating that eventually a correct participant decides. The common decision that is taken by famous consensus protocols, like Paxos [10] and Raft [11], is to make sure that if the messages get delayed, at least

[1] http://www.coindesk.com/r3-ethereum-report-banks.

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

validity and agreement remain ensured by having the algorithm doing nothing, hence sacrificing termination to ensure that only correct responses—satisfying both validity and agreement—can be returned. These "indulgent" consensus algorithms [12] are appealing, because if after some time the network stabilizes and messages get delivered in a bounded time, then consensus can be reached [4].

We show experimentally that the Ethereum protocol can suffer from the Blockchain anomaly. We describe a distributed execution where even committed transactions of a private chain get reordered so that the latest transaction ends up being committed first. We chose Ethereum for our experiments as it is a mainstream blockchain system that allows the deployment of private chains. Although there exist solutions in the distributed computing literature to order some transactions [13] or to use unstructured overlays to cope with malicious participants [14], to our knowledge no experiment has ever been proposed to show that proof-of-work protocols are subject to such a problem.

We show how to reproduce the Blockchain anomaly by following the same execution, where messages get delayed between machines while some miner mines new blocks. Despite transactions being already committed the eventual delivery of messages produces a reorganisation reordering some of the committed transactions. In our execution, miners are setup to dedicate different number of cores to the mining process, hence mining at different speeds. We argue that the misconfiguration of a machine and the heterogeneous mining capabilities of machines belonging to different companies are sufficiently realistic to allow an attacker to execute a double-spending attack.

Section II overviews the blockchain technology, the Paxos anomaly and defines the important terms of the paper. In Section III, we present the blockchain anomaly. In Section IV, we present our experiments and illustrate how the anomaly is possible with less than half of the mining power. In Section V, we explain how replacing transactions by smart contracts could help bypassing the anomaly. Section VI presents the related work. And Section VII concludes.

II. Preliminaries

In this section, we present the key concepts of Bitcoin and Ethereum consensus protocols, the condition of their termination and the Paxos anomaly before presenting the general model. We consider a distributed blockchain system of n peers where peers can exchange coins from one to another through *transactions*. Peers can fail arbitrarily, they can stop working and can be malicious. Any peer can issue transactions that get recorded into the *transaction pool*. Only special peers, called *miners*, can bundle a subset of the pool of transactions into a block after ensuring that there are sufficient funds available on the accounts of the ledger and that these transactions do not conflict.

A. Blockchain Systems

A blockchain can be considered as a replicated state machine [15] where a reversed link between blocks is a pointer from a state to its preceding state as depicted in Figure 1(a). Consensus is necessary to totally order the blocks, hence maintaining the chain structure. To reach consensus despite arbitrary failures, including malicious behaviors, traditional blockchain systems adopted a technique based on proof-of-work, requiring a proof of computation [16]. Miners provably solve a hashcash crypto puzzle [17] to append a new block to the chain. Given a block and a threshold, a miner repeatedly selects a nonce and applies a pseudo-random function to this block and the selected nonce until it obtains a result lower than the threshold. The difficulty of this work limits the rate at which new blocks can be generated by the network.

B. From Nakamoto's Consensus to Smart Contracts

Nakamoto's consensus [1] is at the core of Bitcoin, the mainstream decentralised digital currency. Interestingly, Nakamoto's consensus does not guarantee agreement deterministically. Instead it guarantees that agreement is met with some probability close to 1. The difficulty of the crypto puzzles used in Bitcoin leads to mining a block every 10 minutes. The advantage of this long period, is that it is relatively rare for the blockchain to *fork* due to blocks being simultaneously mined and Bitcoin resolves these forks by choosing the longest branch and discarding the other(s).

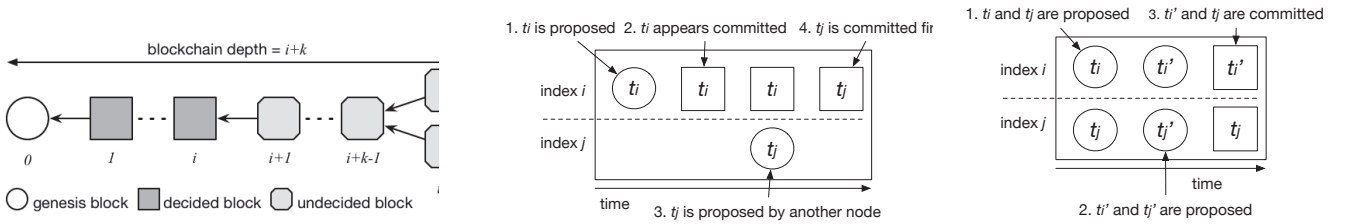
Ethereum [18] is a recent open source cryptocurrency platform that also builds upon proof-of-work. As opposed to Bitcoin's consensus protocol, Ethereum generates one block every 12–15 seconds. While it improves the throughput (transactions per second) it also favors transient forks as miners are more likely to propose new blocks simultaneously. To avoid frequently wasting mining efforts to resolve forks, Ethereum uses the GHOST (Greedy Heaviest Observed Subtree) protocol that does not necessarily discard all the, so called uncle, blocks of non selected branches. Ethereum offers a Turing-complete programming language that can be used to write *smart contracts* [19] that define new ownership rules.

C. Termination of Consensus

By relaxing the agreement property of consensus, blockchain systems can guarantee termination deterministically. In the context of blockchain, termination of consensus indicates that a block has been *decided* for the next available block index. We say that all the transactions of a decided block are *committed*.[2] This decision upon a block inclusion in the chain is necessary for cryptocurrency exchange platforms, for example, to determine that coins of a particular type that are newly minted[3] within this block can be converted into *altcoins*

[2]Here, we use the term "committed" rather than "confirmed" as, in the blockchain terminology, a transaction is meant to be "confirmed" sometimes when only its block is mined, and sometimes when $k+1$ blocks get mined (its own block and the k successor blocks).

[3]As opposed to *mining* that includes the computation of the miners, *minting* consists simply of the creation of coins.



(a) The blockchain structure starts with a genesi at index 0 and links successive blocks in reverse their index; a new block is decided at index $i >$ the blockchain depth reaches $i+k$ (note that a blo of depth of 0 is a genesis block)

(b) The Blockchain anomaly: a first client issues that gets successfully mined and committed the second client issues t_j, with t_j being conditio to the commit of t_i (note that $j \geq i+k$ t_i to be committed before t_j gets issued), the transaction t_j gets finally reorganized and s cessfully committed before t_i, hence violating dependency between t_i and t_j

(c) The Paxos anomaly: a first leader proposes t_i and t_j for slots i and $j > i$ (with t_j being implicitly conditional to the commit of t_i), a second leader proposes t'_i and t'_j while a third leader commits t'_i and t_j for slots i and j, respectively, hence violating the dependency between t_i and t_j

Fig. 1: An example of decided blocks and the difference between the Paxos and the Blockchain anomaly

(coins of a different type) or fiat currencies (e.g., EUR, USD). In particular, observing that a block was mined and appended to the chain is not sufficient to guarantee that it is decided: this block could be part of one branch of a transient fork without consensus being reached yet on any of these branches.

Figure 1(a) depicts the termination of consensus on the index i of a blockchain starting with the *genesis* block. An arrow pointing from right to left indicates that a block contains a hash of its predecessor block, the one located immediately on its left. Newly mined blocks are added to the right end of the blockchain that may fork transiently if multiple blocks referring to the same predecessor get mined concurrently. Forks are only transient and their resolution depends on the blockchain system in use. The consensus for an index i terminates when participants decide on the new block to be assigned at index i. The decision upon the block at index i occurs for all $i > 0$ when the blockchain depth reaches $i+k$, where $k \geq 0$ is a constant dependent on the Blockchain.

Different blockchain systems adopt different values of k to define termination. In Bitcoin (btc), $k_{btc} = 5$, meaning that the block at index i is decided—consensus for index i terminates—when the $k_{btc} + 1 = 6$ blocks at indices $i, ..., i+5$ have been successfully mined. As we previously mentioned, a new block is decided every 10 minutes in Bitcoin, hence it takes $(k_{btc}+1) * 10\,min = 1\,hour$ for a transaction to be committed in Bitcoin. In Ethereum (eth) since version 1.3.5 Homestead, $k_{eth} = 11$, meaning that the block at index i is decided—consensus for index i terminates—when the blockchain depth reaches $i+11$. Hence it takes $(k_{eth}+1)*15\,sec = 3\,min$ for transactions to be committed in Ethereum. Note that some cryptocurrency exchange platforms adopt different values of k to adjust the probability of agreement, hence QuadrigaCX Ether Trading waits for $k'_{btc}+1 = 4$ blocks to be mined in the Bitcoin blockchain while it waits for $k_{eth}+1 = 12$ blocks to be mined in the Ethereum blockchain.[4]

D. The Paxos Anomaly

Paxos is a famous consensus protocol originally guaranteeing agreement and validity despite crash failures [10]. The *Paxos anomaly* [7], [6] stems from the difficulty of implementing conditional requests (or transactions) in Paxos: Paxos decides on individual proposed transactions, potentially violating dependencies between transactions even when proposed by the same requester as depicted in Figure 1(c) where a slot can be viewed as the index of the decision. These dependencies can be useful to make the execution of a transaction t_j dependent on the successful execution of a previous transaction t_i: for example if Bob wants to transfer an amount of money to Carole (t_j) only if he successfully received some money from Alice (t_i). In centralised systems, this anomaly can be easily avoided by enforcing an ordering on these transactions by simply forwarding all requests to a primary node or coordinator [6]. However, in Paxos, as in fully decentralised systems, the first transaction may not be decided in favor of another proposed transaction in a first consensus instance, while in a subsequent consensus instance the second transaction may be successfully decided. This results in a violation of the condition that the second transaction should be decided only if the first transaction was decided.

Below we present the Blockchain anomaly due to the decentralised aspects of blockchain systems, like Bitcoin and Ethereum. The Blockchain anomaly shares similarities with the Paxos anomaly, except that it can occur when transactions, issued by different nodes of the system, are not even concurrent.

III. The Blockchain Anomaly

We present the Blockchain anomaly, an anomaly of blockchain consensus protocols.

A. Causes of the Blockchain Anomaly

The problem stems from the asynchrony of the network, in which message delays cannot be bounded, and the termination of consensus. Although two miners mine on the same chain starting from the same genesis block, a long enough delay in messages between them could lead to having the miners seemingly agree separately on different branches containing more than k blocks each, for any k. This anomaly is dramatic as it can lead to simple attacks within any network where users

[4] https://www.quadrigacx.com/faq.

have an incentive to maximise their profits—in terms of coins, stock options or arbitrary ownership. Moreover, this scenario is realistic in the context of (consortium or fully) private chain where the employees of an institution, like Data61-CSIRO, have direct access to some of the network resources. When messages get finally delivered, the results of the disagreement creates inconsistencies.

B. Uncommitting Transactions is Abnormal

Figure 1(b) depicts the Blockchain anomaly, where a transaction t_i gets committed as part of slot i. After observing that t_i is committed, a node proposes a new transaction t_j knowing that t_i was successfully committed. Again, one can imagine a simple scenario where "Bob transfers an amount of money to Carole" (t_j) only if "Bob had successfully received some money from Alice" (t_i) before. However, once these nodes get notified of another branch of committed transactions, they decide to reorganise the branch to resolve the fork. The reorganisation removes the committed transaction t_i from slot i. Later, the transaction t_j is successfully committed in slot i.

The anomaly stems from the violation of the dependency between t_j and t_i: t_j occurred meaning that Bob has transferred an amount of money to Carole, however, t_i did not occur meaning that Bob did not receive money from Alice. Note that in Bitcoin, transaction t_i gets discarded whereas in Ethereum transaction t_i may in some cases be committed in slot j.

C. Facilitating a Double-Spending Attack

One dramatic consequence of the Blockchain anomaly is the possibility for an attacker to execute a *double-spending* attack: converting, for example, all his coins into goods twice. The scenario is similar to a double-spending attack against Bitcoin [20] and consists of the attacker issuing a first transaction t_1 that converts all its coins into goods in block i and starting mining blocks after block $i-1$ in isolation of the network. As part of this mining, the attacker mines another transaction t_2 that also converts all its coins into goods. The attacker then waits for the blockchain depth to reach $i+k$ after which it can collect its goods as a result of transaction t_1, then it publicizes its longer chain without t_1 so that the chain gets adopted by the rest of network. t_2 gets committed in block j and after the chain depth reaches $j+k$, the peer can collect its goods for the second time. Note that even if one tries to recommit t_1 later, the transaction will be invalidated because the balance is insufficient, however, the double-spending already occurred.

D. Tracking Blockchain Anomalies

Another dramatic aspect of the Blockchain anomaly is that it goes undetected. More specifically, the Blockchain anomaly relies on a wrongly committed state of the blockchain. Once the wrongly committed state gets uncommitted, there is no way to a posteriori observe this problematic state and to notice that a blockchain anomaly occurred. Although it is possible to observe that a peer mined several blocks in a row, there is no way to track down the beneficiaries of the Blockchain anomaly. This dangerously incentivizes participants to leverage the Blockchain anomaly to attack the private chain.

IV. Experimental Evaluation

In this section, we describe a distributed execution involving a private chain that results in the Blockchain anomaly.

A. Experimental Setup

We deployed a private blockchain system in our local area network using geth version 1.4.0, which is a Go implementation of the command line interface for running an Ethereum node. We setup three machines connected through a 1 Gbps network, two consisting of miners, p_1 and p_3, generating blocks and one consisting of a peer p_2 simply submitting transactions. Peers p_1 and p_2 consist of 2 machines with 4 $\times$ AMD Opteron 6378 16-core CPU running at 2.40 GHz with 512 GB DDR3 RAM, each. Peer p_3 consists of a machine with 2 $\times$ 6-core Intel Xeon E5-260 running at 2.1 GHz with 32 GB DDR3 RAM.

We artificially created a network delay by transiently annihilating connection points between machines. Note that such artificial delays could be reproduced by simply unplugging an ethernet cable connecting a computer to the company network and does not require an employee to access physically a switch room.

Also, we made sure p_3 would mine faster than p_1, by mining with the 24 hardware threads of p_3 and a single hardware thread of p_1. The same speed difference could be obtained between a loaded server and a server that does run any other service besides mining. Note that hardware characteristics may also help one machine mine faster than the rest of a private chain network. For example, a machine equipped with an AMD Radeon R9 290X would mine faster in Ethereum than a pool of 25 machines, each of them mining with an Intel Core i7. The same setting as the one used above could allow us to conduct a 51-percent attack, however, the 51-percent attack is not necessary to encounter a blockchain anomaly. For example in a private blockchain adopting the longest branch, if the attacker only owns a minority of the mining power then not adapting the block size or the difficulty of the crypto-puzzle adequately with respect to the network delay could result in having the system adopting p_3's branch anyway.

B. Distributed Execution

For the sake of reproducibility, we present a simple execution where a malicious miner mines faster than a correct miner, the discussion of the anomaly when the malicious miner owns less than half of the mining power is defer to Section IV-E. In our experiment, the client only sends coins once the peer owns a verified amount of coins. The peer performs a transaction t_2 only if it was shown by the system that the previous transaction t_1 had been committed and the money was successfully transferred to its wallet.

Figure 2 depicts the distributed execution leading to the Blockchain anomaly where p_1, p_2 and p_3 exchange information about the blockchain whose genesis block is denoted 'G'.

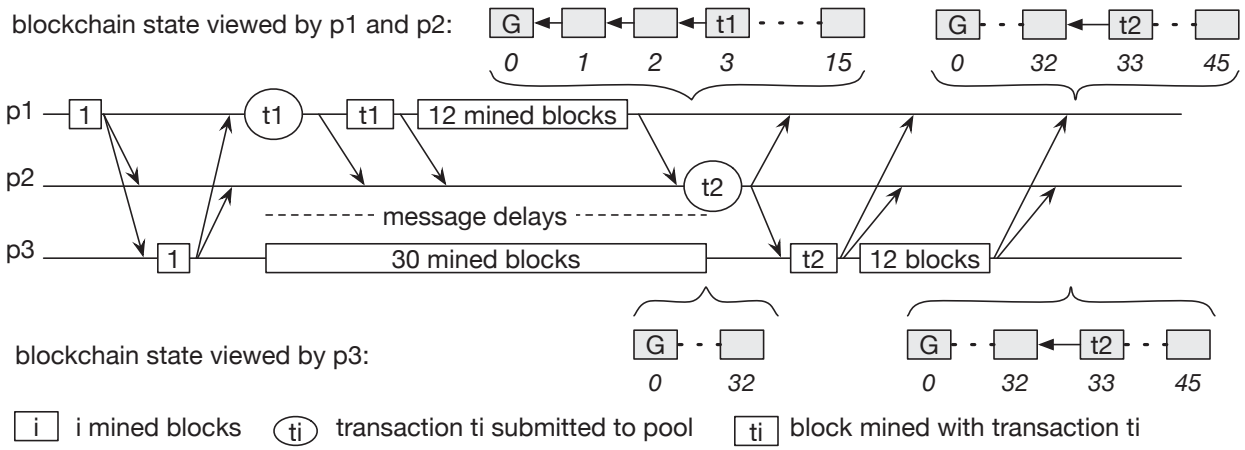


Fig. 2: Execution scenario leading to the Blockchain anomaly: p_3 mines a longer chain than p_1 without including t_1 and without disseminating new blocks until it forces a reorganisation that imposes t_2 to be committed while t_1 appears finally uncommitted

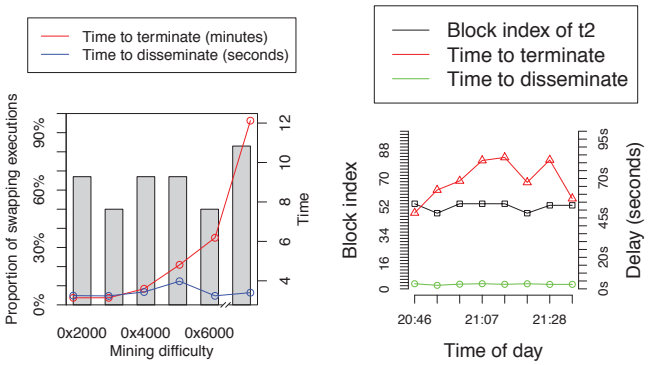
1) Peer p_1 mines a first block after the genesis block and informs p_2 and p_3 to update their view of the blockchain state.
2) Peer p_3 mines a second block and informs p_1 and p_2 of this new block.
3) A network delay is introduced between peers p_1 and p_2 on the one hand, and peer p_3 on the other hand.
4) Peer p_1 submits transaction t_1 and informs p_2 but fails to inform p_3 due to the network delay. In the meantime, peer p_3 starts mining a long series of 30 blocks.
5) Peer p_1 mines a block that includes transaction t_1 and mines 12 subsequent blocks; p_1 then informs p_2 but not p_3 due to the network delay.
6) Peer p_2 receives the notification from p_1 that t_1 is committed because its block and k subsequent blocks are mined; then p_2 decides to submit transaction t_2 that should only execute after t_1.
7) The network becomes responsive and p_3 who receives the information that t_2 is submitted, mined t_2 in a block along with 12 subsequent blocks.
8) Once peers p_1 and p_2 receive from p_3 the longest chain of 45 blocks, they adopt this chain, discarding or postponing the blocks that were at indices 2 to 15, including the transaction t_1, of their chain.
9) All peers agree on the final chain of 45 blocks in which t_2 is committed and where t_1 is finally not committed before t_2.

This execution results in a violation of the conditional property of transaction t_2 stating that t_2 should only execute if t_1 executed first. This violation occurred because transaction t_1 had been included in one chain, decided and agreed by two of the participants, it was then changed after the message of the third participant was finally delivered to the rest of the network.

C. Automating the Reproduction of the Anomaly

To illustrate the anomaly, we wrote a script that automated the execution depicted in Figure 2. Figure 3(b) represents the execution of a script that execute 8 iterations of the Blockchain anomaly over a period of 50 minutes. Again the goal is to wait until t_1 gets committed before issuing t_2 that ends up being committed while t_1 does not appear to be. Note that this is similar to Figure 1(b) except that t_2 is not necessarily included at the index t_1 occupied initially. In particular, the block in which t_2 gets included varies from one iteration to another due to the non-determinism of the execution as indicated by the curve with square points. This non-determinism is explained by the randomness of the mining process and the latency of the network that also impacts the time it takes for the consensus to terminate (curve with triangle points) in each iteration of the experiment. Note that we use $k = 11$ in this experiment, making sure that 12 blocks were successfully mined, as recommended since the release of Ethereum 1.3.5 Homestead, for the consensus to terminate.

As expected, in each of these eight cases we observed the Blockchain anomaly: even though t_2 was issued after t_1 was successfully observed as committed, if the messages get successfully delivered, then the reorganisation results in t_2 being committed while t_1 is not. Finally, we can observe that the time to disseminate a committed transaction to all the peers of the network is much shorter than the termination delay.



(a) The proportion of transaction swaps observed does not depend on the difficulty, as opposed to the consensus termination that increases with the difficulty

(b) Automated executions of the Blockchain anomalies over a period of $50\,min$, the execution is non-deterministic due to the randomness of the mining process and the network delay between peers

Fig. 3: Experimental evaluation of the anomaly

This is due to the time needed to mine a block, which is significantly larger than the latency of our network.

D. Swap Frequency with Different Mining Difficulties

In the previous experiment, we used the default Ethereum difficulty (0x4000) and automated the execution with a precise script. To better understand the cause of the anomaly we tried reproducing the anomaly by hand (without the script) with larger difficulties.

Figure 3(a) depicts the average number of blockchain anomalies leading to a *swap*, where both t_2 and t_1 are eventually committed in reverse order, occurring in our private chain for 6 different mining difficulties. Each bar results from the average number of anomalies observed during 6 manual runs of the scenario depicted in Figure 2.

We ran this particular experiment with $k = 10$ for the termination of consensus, meaning that t_1 was mined in block at index i and it was committed once the chain depth reached $i + 10$ blocks. (We presented the anomaly in the case where $k = 11$ in Section IV-C.)

We varied the difficulty from 0x2000 to 0x40000 and measure the frequency of the Blockchain anomaly and the time it would take for consensus to terminate (upper curve). We observed that the termination time was proportional to the difficulty while the occurrence of the anomaly was not significantly affected by the difficulty. This is explained by the fact that the difficulty impacts the time it takes to mine $k+1$ blocks for termination. In addition we report the time it would take for a transaction in a mined block to be disseminated to all the peers of the network (bottom curve) and observed that it was not related to the difficulty.

E. The Blockchain Anomaly Differs from the 51-Percent Attack

For the sake of reproducibility we simplified the execution leading to the Blockchain anomaly in Figure 2. It is however important to note that the Blockchain anomaly can occur even though the malicious user controls less than half of the mining power. To this end, Figure 4 depicts an execution where peer

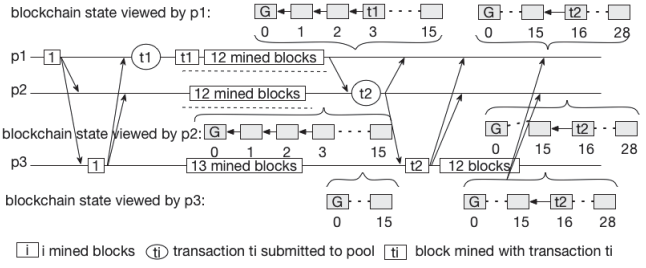


Fig. 4: Half of the mining power is not necessary for the blockchain anomaly as it is sufficient to discard the blockchain containing t_1

```
contract conditionalPayment {
  // to keep track of the amount paid by Alice
  uint32 paid;
  // map addresses to their respective balance
  mapping (address => uint256) public balances;
  // the address of Alice's account
  address A = 0x57ec7927841e2d25aad5f335e3b701369b177392;
  // the address of Bob's account
  address B = 0x5ae58375c89896b09045de349289af9034902905;
  // the address of Carole's account
  address C = 0x3b12387c88de7834ab3129e3949d0918c4a09122;

  // enables function depending on invoker
  modifier onlyFrom(address _address) {
    if (msg.sender != _address) throw;
    _
  }

  // Alice sends money to Bob
  function sendTo(address B, uint32 _amount) onlyFrom(A) {
    if (balances[A] >= _amount) { // sufficient funds?
      balances[A] -= _amount;
      balances[B] += _amount;
      paid = _amount; // sorting the amount paid
    }
  }

  // Bob sends money to Carole
  function sendIfReceived(address C, uint32 _amount) onlyFrom(B) {
    if (paid > _amount) { // only if previous payment
      balances[B] -= _amount;
      balances[C] += _amount;
    } else {
      throw; // cancel contract execution
    }
  }
}
```

Fig. 5: A smart contract written in the Solidity programming language to replace transactions prone to the blockchain anomaly: the `sendIfReceived` function checks that the transfer from A to B occurred before executing the transfer from B to C

p_3 owns strictly less than half of the mining power. As the network is delayed between all pairs of nodes, we can see that p_1 alone cannot mine a chain longer than p_3's and that the blockchain of p_1 containing t_1 gets eventually overridden.

V. Smart Contracts

Smart contracts are a foundational aspect of the *Ethereum* system, as they are distributed code execution based on conditional aspects. The contracts can be programmed to allow

```
1  contract problematicConditionalPayment {
2    ...
3    function checkPayment(address B, uint32 _amount) onlyFrom(B)
         constant returns (bool result) {
4      if (paid > _amount) { // check that Alice paid
5        return true;
6      } else throw;
7    }
8    // Bob sends money to Carole
9    function sendIfReceived(address C, uint32 _amount) onlyFrom(B) {
10     balances[B] -= _amount;
11     balances[C] += _amount;
12   }
13 }
```

Fig. 6: Executing the transfer to Carole in a separate function may suffer from the Blockchain anomaly

for certain conditions to be met in order for the code to be executed. What we found was that the anomaly prevention depended entirely on the programming of the smart contract. This means that if a smart contract was coded so that it did not properly check the condition that the first transaction had occurred, it would execute as normal, acting like a normal transaction and suffering from the anomaly.

In Figure 5, we illustrate the writing of a smart contract in the Solidity programming language with which we could not observe the anomaly. The key point is that the `sendIfReceived` function groups two steps: the check that the amount has been paid at Line 22 and the payment that results from this successful check at Lines 23 and 24. Because these two steps are executed on-chain, we know that one has to be necessarily true for the second to occur.

However, if the two steps were parts of two separate functions of the contract, one checking that the amount had been paid and another that would do the payment and be invoked upon the returned value of the former then the anomaly could arise. For example, consider Figure 6 where one function, `checkPayment`, checks that the payment from Alice proceeded correctly (Lines 3–7) and the other function, `sendIfReceived`, is modified to execute the payment unconditionally (Lines 9–13). Even if Bob invokes `checkPayment` and observes that it returns successfully before invoking `sendIfReceived` the anomaly may arise. The reason is that the check is made off-chain and nothing guarantees that the payment from Alice was not reorganized while Bob was checking the result off-line.

To conclude, the former contract in Figure 5 does not suffer from the Blockchain anomaly as it executes the check and the conditional transfer on-chain.

VI. Related work

Proof-of-work has been previously compared to Byzantine Fault Tolerant protocols [21], [22]. Some of this research [21] focuses on comparing experimentally Bitcoin against PBFT [23]. The Bitcoin blockchain and the PBFT consensus protocol were evaluated with nodes scattered at 8 locations around the world. As one could expect given the difficulty of the crypto puzzle of Bitcoin, the experiments showed that PBFT achieves a lower latency and a higher throughput than Bitcoin in serving transactions. However, PBFT suffers from scalability limitations and using sharding [24] could be necessary to scale to hundreds of nodes.

Another part of this research [22] discusses the probabilistic guarantees of proof-of-work systems and the deterministic guarantees of Byzantine fault tolerance. The proof-of-work consensus is compared to Byzantine agreement protocols along two axes, scalability and performance, where proof-of-work consensus protocols are considered as scalable but inefficient while Byzantine agreement protocols are considered as efficient but not scalable. For example, Bitcoin scales beyond 1000 nodes while achieving a performance lower than 100 transactions per second with a high latency, whereas standard Byzantine fault tolerant protocols achieve more than 10,000 transactions per second but scale only to tens of nodes.

Multiple attacks to Bitcoin share the "solo-mining" technique we used to illustrate the Blockchain anomaly in Ethereum. Some attacks assume that the merchant accepts transactions before they are confirmed [25], [26], [27]. Other attacks assume the merchant to accept transactions that are confirmed once [28]. According to our definition none of these transactions are however "committed", hence these attacks cannot be considered anomalies. The Blockchain anomaly affects transactions that are committed (or $k+1$ times confirmed).

Even though more than half of the mining power was controlled by the adversary to illustrate a simple scenario to reproduce the Blockchain anomaly, it is important to notice that the Blockchain anomaly is different from the 51-percent attack. First, note that the smart contract solution we proposed in Section V cannot fix the 51-percent attack. More generally, the blockchain anomaly could occur with n nodes with all attackers owning totally a q-th of the mining power if each correct node mines at a rate of q/n blocks every block propagation delay.

Some solutions to the Blockchain anomaly could be easily implemented in Ethereum. As an example, logical clocks is a well-known technique to order causally-related events in a distributed system [13]. A logical clock could be used to order two transactions issued by the same peer, simply associating messages to sequence numbers using a monotonically increasing counter at each peer. As this cannot be used for dependent transactions issued by different peers, one could use a special flag when issuing a transaction to inform the miners to either ignore the transaction or to mine it within the same block as its precedent dependent transaction. Designing the reward model to incentivize the miners to follow this protocol is out of the scope of this paper. Other technique to perform a transaction as soon as other were performed were used to enhance the scalability of Bitcoin [29].

Some solutions immune to the Blockchain anomaly also exist. PeerCensus [30] was proposed as an algorithm with two components: one to execute a Byzantine agreement protocol on top of Bitcoin with a simple voting system and another to minimize the effect of Sybil attacks during these votes.

The latter component makes it difficult for an attacker to create multiple identities so as to outnumber the votes with its own votes. Using this technique PeerCensus strengthens the guarantees of Bitcoin and resolves immediately the forks, hence avoiding the Blockchain anomaly.

Although the Paxos anomaly was not considered a problem in the original design of Paxos [10], this scenario was informally stated as an anomaly during the design of the Zookeeper distributed coordination service [6], due to the engineers needing to implement conditional concurrent requests: Zookeeper organizes nodes into a tree structure and it was desirable for the additions of a parent node and its child to be made concurrent. The child addition depended naturally on the success of the parent addition. Note that for other applications that do not need concurrent dependent requests Paxos is sufficient [31]. A major difference between the Paxos and the Blockchain anomalies is that if consensus is reached with Paxos, the index of the decision cannot change while the Blockchain anomaly precisely stems from the fact that the index of a decided transaction, or the order of its block in the chain, can change.

VII. Conclusion

In this paper, we demonstrate empirically the presence of the Blockchain anomaly in proof-of-work blockchain systems. Named after the Paxos anomaly, it prevents a user of mainstream blockchain systems from executing a conditional transaction, a transaction that should only execute in the current observable committed state or a later state of the system. A possible way to avoid the anomaly could be to write smart contracts rather than transactions, yet it adds to the level of complexity.

Our conclusion is that blockchain systems are difficult to use properly. This observation should discourage users from using blockchain systems unless they fully understand the underlying design principles and the guarantees they offer. Besides the prominent blockchain systems we have discussed, namely Bitcoin and Ethereum, there exist many alternatives. Exploring the alternatives that exclusively offer deterministic guarantees for private chains is part of future work.

References

[1] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2008, http://www.bitcoin.org.

[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger final draft - under review," 2014, https://github.com/ethereum/wiki/wiki/White-Paper.

[3] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[4] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, Apr. 1988.

[5] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, 2015, pp. 507–527.

[7] K. Birman, D. Malkhi, and R. van Renesse, "Virtually synchronous methodology for dynamic service replication," Microsoft Research, Tech. Rep. MSR-TR-2010-151, 2010.

[6] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *ATC*. USENIX, 2010, pp. 11–11.

[8] ——, "Appendix A: Virtually synchronous methodology for building dynamic reliable services," in *Guide to Reliable Distributed Systems: Building High-Assurance Applications and Cloud-Hosted Services*. London: Springer London, 2012, pp. 635–671.

[9] V. Gramoli, "On the danger of private blockchains," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL)*, July 2016.

[10] L. Lamport, "The Part-Time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, May 1998.

[11] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[12] R. Guerraoui, "Indulgent algorithms (preliminary version)," in *PODC*, 2000, pp. 289–297.

[13] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[14] E. Anceaume, Y. Busnel, and S. Gambs, "On the power of the adversary to solve the node sampling problem," *Trans. Large-Scale Data- and Knowledge-Centered Systems*, vol. 11, pp. 102–126, 2013.

[15] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, S. Chen, A. Ponomarev, and A. B. Tran, "The blockchain as a software connector," in *Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2016.

[16] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '92, 1993, pp. 139–147.

[17] A. Black, "Hashcash - a denial of service counter-measure," Cypherspace, Tech. Rep., 2002. [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf

[18] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2015, yellow paper.

[19] N. Szabo, "Formalizing and securing relationships on public networks," 1997. [Online]. Available: http://szabo.best.vwh.net/formalize.html

[20] M. Rosenfeld, "Analysis of hashrate-based double-spending," 2012.

[21] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," in *3rd Workshop on Bitcoin Research (BITCOIN), Barbados*, February 2016.

[22] M. Vukolíc, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Proceedings of the Workshop on Open Research Problems in Network Security (iNetSec 2015)*, ser. LNCS, 2016.

[23] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[24] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS)*, 2016.

[25] H. Finney, "Finney's attack," February 2011. [Online]. Available: https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384

[26] G. Karame, E. Androulaki, and S. Capkun, "Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin," *IACR Cryptology ePrint Archive*, vol. 2012, p. 248, 2012.

[27] T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten, "Have a snack, pay with bitcoins," in *Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2013, pp. 1–5.

[28] vector76, "The vector76 attack," August 2011. [Online]. Available: https://bitcointalk.org/index.php?topic=36788.msg463391#msg463391

[29] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016, draft Version 0.5.9.2. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[30] C. Decker, J. Seidel, and R. Wattenhofer, "Bitcoin meets strong consistency," in *Proceedings of the 17th International Conference on Distributed Computing and Networking (ICDCN)*, 2016, p. 13.

[31] V. Gramoli, L. Bass, A. Fekete, and D. Sun, "Rollup: Non-disruptive rolling upgrade with fast consensus-based dynamic reconfigurations," *IEEE Trans. on Parallel and Distributed Systems*, 2016.

Safety Analysis of Bitcoin Improvement Proposals

Emmanuelle Anceaume, Thibaut Lajoie-Mazenc
CNRS, UMR 6074 - IRISA
firstname.lastname@irisa.fr

Romaric Ludinard
ENSAI, UMR 9194 - CREST
romaric.ludinard@ensai.fr

Bruno Sericola
INRIA RBA
bruno.sericola@inria.fr

***Abstract*—Decentralized cryptocurrency systems offer a medium of exchange secured by cryptography, without the need of a centralized banking authority. Among others, Bitcoin is considered as the most mature one. Its popularity lies on the introduction of the concept of the blockchain, a public distributed ledger shared by all participants of the system. Double spending attacks and blockchain forks are two main issues in blockchain-based protocols. The first one refers to the ability of an adversary to use the very same bitcoin more than once, while blockchain forks cause transient inconsistencies in the blockchain. We show through probabilistic analysis that the reliability of recent solutions that exclusively rely on a particular type of Bitcoin actors, called miners, to guarantee the consistency of Bitcoin operations, drastically decreases with the size of the blockchain.**

***Keywords*—** Bitcoin; Peer-to-Peer Systems; Safety; Analytical Performance Evaluation

I. Introduction

The goal of decentralized cryptocurrency systems is to offer a medium of exchange secured by cryptography, without the need of a centralized banking authority. An important design issue of such a platform is to prevent the occurrence of double-spending attacks, which consists in using the same resources (the same "coins") in more than one transaction. Classically, the centralized banking authority serves as a trusted third-party that verifies the validity of every single transaction which prevents this kind of attacks. In absence of such a trusted entity, an alternative mechanism must be implemented.

Satoshi Nakamoto proposed a solution in 2008 [20]: the *Bitcoin cryptocurrency system*, the first decentralized ecosystem providing users with a virtual currency to buy and sell services or goods. Bitcoin relies on a public distributed ledger, the so-called *blockchain*, that records all the valid transactions ever issued in the Bitcoin system. Technically, the blockchain is built by some of the participants of the ecosystem, the *miners*, through the creation of blocks. Each block contains the set of most recently issued transactions. The strength of the blockchain design is that it does not require participants to trust each other, each one maximizing its self-interest. Thus, it can be viewed as a way to create a global trusted third-party from a network populated by untrusted participants.

To prevent double-spending attacks, the blocks (and thus all the transactions) must be totally ordered so that every participant of the system can check their validity. However, concurrent blocks can be created due to propagation delays and this must be carefully handled to enforce the consistency of the blockchain. This is achieved by introducing some synchronization among the miners: to successfully create a block, a miner must first solve a resource consuming computation, the so-called *proof-of-work*. Miners are rewarded for each successfully created block, which introduces a competition among them to create the next block as fast as possible.

Such a competition may in its turn give rise to concurrent blocks, and thus to the creation of different branches in the blockchain. This phenomenon is called a *blockchain fork*. Even if Bitcoin eventually converges to a legal state (*i.e.* a unique branch), stabilization may take time. For instance, the March 2013 fork [7] was resolved after several hours. During a fork, an attacker may repeatedly perform double-spending attacks. Given the increasing popularity of Bitcoin, one may expect that the abuse of the weaknesses in its design will increase.

A large amount of work has been devoted to mitigating this salient issue and, among them, three propositions have recently emerged as the very first convincing solutions to solve the double-spending attack. These solutions, respectively called Bitcoin-NG [6], PeerCensus [5], and BizCoin [13], propose to give additional specific power to miners, by coordinating their view of the blockchain through either executions of Byzantine-tolerant consensus protocols or leadership of one of them.

a) Our contributions: The paper is devoted to a thorough analysis of the behavior of these three works with respect to their capacity to handle numerous transactions and their resilience to malicious miners. Prior to this analysis, we provide an extensive description of the Bitcoin ecosystem from which we derive a formalization of its properties in terms of validity, confirmation, safety and liveness. We then present the model that allows us to investigate the properties of the three above mentioned solutions. The outcome of this study contains a mixture of both favorable and negative results. Bitcoin-NG, by relying on a leader, neither improves upon Bitcoin safety, nor scales to a large number of transactions. Despite the support of Byzantine-tolerant consensus algorithms, PeerCensus does not tolerate the well-known threshold of $1/3$ malicious miners. On the other hand, by combining the design of both Bitcoin-NG and PeerCensus and by relying on the CoSi protocol [24] for collective signing, BizCoin shows good theoretical performance. Its resilience to malicious miners corroborates results of Byzantine tolerant distributed algorithms [15] for large enough signature groups: in the presence of less than one third of Byzantine miners, BizCoin is safe with high probability if the number of miners involved in the signature group exceeds $1,000$. However, BizCoin has

978-1-5090-3216-7/16/$31.00

not yet been implemented with more than 148 miner because of the complexity of the underlying CoSi protocol [12].

b) Road map: To summarize, the remainder of the paper is organized as follows. Section II presents the necessary background to understand Bitcoin properties in terms of safety and liveness. Section III presents an analysis of the safety guaranteed by respectively Bitcoin-NG, PeerCensus and BizCoin. Section IV presents a brief survey of the crypto systems that have appeared since 2009. Finally, Section V concludes.

II. Background on The Bitcoin Network

The Bitcoin network [20] is a peer-to-peer payment network that relies on distributed algorithms and cryptographic tools to allow entities to pseudonymously buy goods or services with digital currencies called bitcoins. Its main ingredients are *(i) transactions* issued by buyers each time they wish to spend bitcoins, *(ii)* the *blockchain*, an ordered sequence of blocks, each one being a set of issued transactions, maintained distributively by the entities of the system, and *(iii)* a pool of pending transactions eligible for confirmation in the blockchain, locally maintained by each entity. Three types of entities participate in the Bitcoin ecosystem: *users*, that send and receive bitcoins, *peers* that propagate transactions in the network and maintain a local copy of the blockchain, and *miners*, that establish which transactions will appear in the blockchain, and the order in which they will appear. Of course, at any time, an entity may play any of the roles in the Bitcoin ecosystem.

A. The Bitcoin protocol

To illustrate the description of the Bitcoin ecosystem, we will take the example of individual users called Alice, Bob and Carol. Alice owns bitcoins, and she wishes to send them to Bob and Carol for the goods they provide to her. Bitcoins are entirely virtual. They are accessible via Bitcoin accounts. An account, which refers to the elementary functional entity of the Bitcoin ecosystem, is described by a key, derived from the public key of the public/private key generated by Bitcoin users. Keys are used to prove the ownership of bitcoins. A bitcoin account is locked by its owner, and spending bitcoins amounts to unlocking that account and transferring its value to that of the recipient of the transaction who will be credited once the transaction is confirmed in the blockchain (more details will be given in the sequel). Note that to hide their profile, it is recommended that users generate a new public/private key for each transaction they are recipient of. An important aspect of Bitcoin accounts is their indivisibly, meaning that once an account has been created by a user, it will be credited by a single transaction and will be debited by a single subsequent transaction. Note that Alice may voluntarily pay a small *transaction fee* which will be kept by the miner that will succeed in confirming Alice's transaction in the blockchain. In this case, the total amount of bitcoins in the input accounts is greater than the amount of bitcoins transferred to the output accounts. The successful miner creates an account that will be credited with the fees from all the transactions of the block, along with a block reward (whose amount is currently set to 12.5 bitcoins; it is halved every 210,000 blocks, which last occurred in July 2016) through a *coinbase* transaction, the way Bitcoin creates money. Specifically, to send bitcoins to Bob and Carol, Alice creates a transaction $T = (\{a_{i_1}, \ldots, a_{i_m}\}, \{b_j, c_\ell\})$, where $\{a_{i_1}, \ldots, a_{i_m}\}$, $m \geq 1$, refers to Alice's credited accounts that she wishes to spend. Set $\{a_{i_1}, \ldots, a_{i_m}\}$ is called the input set of transaction T, and is denoted by I. The inputs of a transaction are actually the hash of the transactions that credited Alice's accounts. We refer directly to the accounts for a sake of clarity. Accounts b_j and c_ℓ have respectively been created by Bob and Carol to receive bitcoins from Alice's accounts. Set $\{b_j, c_\ell\}$ is called the output set of T, and is denoted by O. The amount of bitcoins of account u_i is denoted by $v(u_i)$. T also includes Alice's digital signature on the input and output accounts; thus, any user can verify its integrity by checking the chain of signatures.

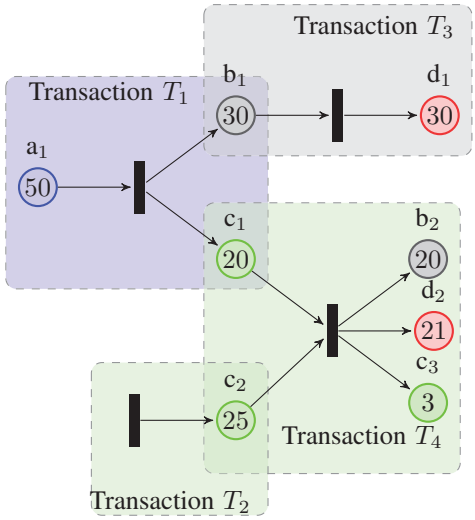


Fig. 1. Modelling the evolution of users' accounts

To describe the evolution of user accounts, we have adopted a place/transition model as depicted in Figure 1. User accounts are represented by places (circles) and transactions by transitions (vertical bars). The place from which an arc runs to a transition is an input place of the transition, and the place to which an arc runs to is an output place of the transition. The number of bitcoins in a user account represents the tokens of the place. A transition may fire if there are sufficiently many tokens in its input places, and it consumes all of them upon firing. Places and transitions are dynamically created.

In Figure 1, Alice creates transaction T_1 to transfer the 50 bitcoins of her account a_1 to Bob and Carol's accounts: 30 bitcoins to b_1 and 20 to c_1. Transaction T_4 contains a transaction fee equal to $(25+20)-(20+21+3) = 1$ bitcoin. T_2 is a coinbase transaction.

We say that a transaction $T = (I, O)$ is *well-formed* if the transition can be fired, *i.e* $\sum_{i \in I} v(i) \geq \sum_{o \in O} v(o)$, and the creator of T is the owner of the input accounts of T. In the following we consider that all the transactions are well-formed.

When Bob and Carol receive the digitally signed transaction

T, they submit it to any peer p of Bitcoin for a validity check. Informally, a transaction $T = (I, O)$ is valid if p has received all the transactions that have credited all the accounts in I and not received any transaction already using any of those same accounts. To formally define the validity property we introduce the notion of a local view. The local view of p is the pool of pending transactions at p together with the content of p's blockchain $\mathcal{B}^{(p)}$. If we respectively denote by $\mathcal{V}_k^{(p)}$ and $\mathcal{P}_k^{(p)}$ the local view of p and p's pool of pending transactions when p receives its k-th transaction, then we have

$$\mathcal{V}_k^{(p)} = \mathcal{P}_k^{(p)} \cup \mathcal{B}^{(p)}.$$

The current view and the current pool of pending transactions of peer p are simply denoted by $\mathcal{V}^{(p)}$ and $\mathcal{P}^{(p)}$, respectively. We suppose that p has initialised $\mathcal{V}_1^{(p)}$ with the first block broadcast by Satoshi, containing a single coinbase transaction. When p receives a new transaction T, p declares it valid according to the following definition.

Definition 1 (Validity Property)**.** *Given a peer p of the Bitcoin network, p considers its k-th transaction $T = (I, O)$ as locally valid if and only if the following three properties hold:*

$$\forall a \in I, \exists T' = (I', O') \in \mathcal{V}_{k-1}^{(p)}, \quad a \in O' \tag{1}$$

$$\forall T' = (I', O') \in \mathcal{V}_{k-1}^{(p)}, \quad I \cap I' = \emptyset \tag{2}$$

$$\forall a \in O, \forall T' = (I', O') \in \mathcal{V}_{k-1}^{(p)}, \quad a \notin O'. \tag{3}$$

In the following, in accordance with Bitcoin requirements, we suppose that a user creates a new account for each transaction she receives. That is Relation (3) is always satisfied.

As soon as $T = (I, O)$ is considered locally valid, p inserts it in its transaction pool, that is $\mathcal{P}_k^{(p)} \leftarrow \mathcal{P}_{k-1}^{(p)} \cup \{T\}$, then positively acknowledges Bob and Carol, and finally disseminates T in the Bitcoin network. On the contrary, if $T = (I, O)$ is not locally valid, two cases must be considered: either Relation (1) or Relation (2) does not hold. In the former case, p inserts T in its local pool (*i.e.* $\mathcal{P}_k^{(p)} \leftarrow \mathcal{P}_{k-1}^{(p)} \cup \{T\}$) and disseminates it in the Bitcoin network. Transaction T becomes locally valid when p receives the transactions crediting the missing accounts. In the latter case, p has already received a locally valid transaction $T' = (I', O')$ such that $a = I \cap I'$: account a is in a double-spending situation. Formally,

Definition 2 (Double-spending situation)**.** *Given a Bitcoin account a_i, account a_i is in a double-spending situation if there exist two transactions $T_1 = (I_1, O_1)$ and $T_2 = (I_2, O_2)$ such that :*

$$T_1, T_2 \in \bigcup_p \mathcal{V}^{(p)} \wedge a_i \in I_1 \cap I_2.$$

A transaction T is conflict-free if none of the inputs of $T = (I, O)$ is involved in a double-spending situation and all of the transactions that credited T's inputs are conflict-free:

Definition 3 (Conflict-free transaction)**.** *A transaction $T = (I, O)$ is conflict-free if $\forall a \in I$, a is not involved in a double-spending situation and the transaction $T' = (I', O') \in \mathcal{V}^{(p)}$ with $a \in O'$ is conflict-free.*

By construction, the induction is finite because Bitcoin creates money only through coinbase transactions, which are by definition conflict-free. Each T' exists by Relation (1).

Preventing a double-spending situation from transforming into a successful double-spending attack (i.e. Alice succeeds in converting the content of one of her accounts into goods twice) is the key challenge of many virtual crypto-systems.

The solution adopted in Bitcoin to mitigate double-spending attacks, without relying on a central trusted authority, consists in gathering transactions into blocks and totally ordering them in a publicly accessible and distributively managed ledger. This is the role of *miners*. Briefly, the construction of a well-formed block uses the hashcash proof-of-work function which consists in computing $\mathfrak{h}(x)/2^m(m-k)$, where $\mathfrak{h}$ is the double SHA-256 hash function, m is the size of the hash output, *i.e.*, $m = 256$, and k is the work factor. The value of x is obtained by combining, among others, the sequence number of the last block in the blockchain, the set of locally pending transactions and a counter c incremented by the miner until the first k bits of the hash output are 0. The work factor k is adjusted every 2016 blocks to provide an average block creation time of 10 minutes. Once the proof-of-work has been generated by some miner q, it forms, together with the set of locally pending transactions $\mathcal{P}^{(q)}$, a numbered block b_ℓ that q appends to $\mathcal{B}^{(q)}$. Miner q disseminates this block in the Bitcoin network so that each peer appends it to its local copy of the blockchain. The status of a transaction changes from *pending* to *locally confirmed* whenever it is included in a block.

Definition 4 (Local confirmation)**.** *Given a peer q of the Bitcoin network, and a locally valid transaction T,*

$$T \text{ is locally confirmed} \iff \exists! B \in \mathcal{B}^{(q)}, T \in B.$$

The *local confirmation level* of transaction T at peer q is equal to the depth of block B, which corresponds to the number of blocks appended in $\mathcal{B}^{(q)}$ after B, including B.

Bitcoin miners are incentivized by receiving, when they successfully generated a block, a reward in the form of the coinbase transaction, defined above. Blocks, being generated at a regular and very slow rate, provide a fully distributed synchronization of the network. Since bitcoins are only created through block rewards, it also ensures their rarity, leading to their high financial value and hence to the high incentive to create blocks.

B. Blockchain forks

Rewarding the creation of blocks introduces a competition among miners. This competition may lead to concurrent blocks (*i.e.* equally numbered blocks) and thus to a blockchain with a tree structure. This phenomenon is called a *blockchain fork*. A blockchain fork is resolved as soon as a miner generates a proof-of-work and disseminates the corresponding block b_ℓ quickly enough so that no concurrent miner has found a valid proof-of-work before it receives it. In that case, the branch of the local blockchain that contains b_ℓ is longer than any other concurrent branches, which are pruned from the

tree, leading to a blockchain with a unique branch. Note that "pruned" transactions that do not already belong to the unique branch are added again in the local transaction pools for a possible confirmation in subsequent blocks. Blockchain forks must be quickly resolved for two main reasons. Firstly, malicious miners can take advantage of this phenomena to trigger double-spend attacks. Such an attack is successful if the branch that remains after the resolution of the fork contains the illegitimate transaction issued by the attacker. Nakamoto's analysis, as well as subsequent ones [8], [11], [19] focus on the race between malicious miners and honest ones to generate the longer branch of the blockchain. Suppose that Bob is the recipient of a transaction issued by the malicious sender Alice, and that Alice manipulates a proportion μ of the miners of the system. Nakamoto has shown that with probability less than 0.1%, Bob's transaction will be rejected if its level of confirmation z in the local blockchain of some peer is less than 5 when $\mu = 10\%$. The level of confirmation must increase to $z = 8$ when μ increases to $\mu = 15\%$, and to $z = 15$ when 25% of the miners are corrupted. In the following, a transaction is said *deeply confirmed* once it reaches such a confirmation level. The second reason is that, in the presence of several branches, the global computing power of the miners is spread over the branches. This leads to an increase of the average block generation time, and accordingly to the augmentation of the time needed by transactions to become deeply confirmed.

C. Bitcoin properties

We can now state Bitcoin's fundamental properties:

Property 1 (Bitcoin Liveness)**.** *A conflict-free transaction will eventually be deeply confirmed in the blockchain of an honest peer.*

Property 2 (Bitcoin Safety)**.** *A conflict-free transaction deeply confirmed by some honest peer will eventually be deeply confirmed by all honest peers with the same confirmation level.*

Two important remarks are in order.

Remark 1. *Properties 1 and 2 guarantee that the view of all honest peers have the same prefix.*

Remark 2. *Properties 1 and 2 each apply to transactions issued by honest users, but honest recipients of conflictual transactions have no guarantee of receiving the corresponding coins.*

To summarize, to prevent money counterfeiting, Bitcoin opens the door to double-spending attacks against users that optimistically assume that valid or even locally confirmed transactions will eventually be deeply confirmed.

Given the increasing popularity of Bitcoin, any user may legitimately expect a stronger liveness property than the one implemented by Bitcoin. Indeed, in its current implementation, a user cannot detect that a transaction it is recipient of is conflictual and, thus, has no guarantee to be paid for the service provided before said transaction becomes deeply confirmed, which takes one hour in average.

In the sequel of the paper, we present three recent works that aim at providing stronger guarantees to honest users through linearizable operations on Bitcoin accounts. We show that none of these works provide sufficient guarantees in presence of malicious miners.

III. Relying on miners as a trusted third party

Three recent works, Bitcoin-NG [6], PeerCensus [5], and BizCoin [13], have proposed to rely exclusively on miners to take in charge the full process of validation and confirmation to guarantee that all the operations triggered on the transactions are atomically consistent. Atomic consistency guarantees that all the updates on shared objects are perceived in the same order by all entities of the system. In all these protocols, time is divided into epochs. An epoch ends when a miner successfully generates a new block. This miner becomes the leader of the subsequent epoch. Each of these solutions rely on a dedicated set $\mathcal{E}_\ell$, with $\ell \in \{1, w, \infty\}$. This set is built along consecutive epochs as follows. At epoch k, if $|\mathcal{E}_\ell| < \ell$, the new leader is added to $\mathcal{E}_\ell$. Otherwise, the leader at epoch $k + 1 - \ell$ is removed from $\mathcal{E}_\ell$ and the new leader is added. Once set $\mathcal{E}_\ell$ reaches size ℓ, it remains at constant size ℓ.

Strong consistency is implemented in these protocols by different means. In Bitcoin-NG, it is achieved by delegating the validation process to $\mathcal{E}_1$, *i.e.* the leader of the current epoch. In PeerCensus it is implemented by relying on Byzantine Fault Tolerant consensus protocols (*e.g.* [4], [9], [14]) run by $\mathcal{E}_\infty$ (recall that it contains all the miners that successfully generated a block). Finally, BizCoin leverages both ideas by using the leader and a consensus run by $\mathcal{E}_w$. In all these protocols, members of $\mathcal{E}_\ell$, with $\ell \in \{1, w, \infty\}$, are entitled to validate and confirm issued transactions and blocks and to disseminate them so that each peer integrates them in its local blockchain. In the remainder of the section we show that, surprisingly enough, relying on miners to confirm transactions does not prevent malicious users from successfully double-spending bitcoins. Prior to doing that, we present the model we use to analyze the safety of these protocols.

A. Model

We assume the presence of an adversary controlling a proportion $\mu \in (0, 1)$ of the whole set of miners. This adversary aims at exploiting the protocol under consideration in order to perform double spending attacks. Miners that are controlled by the adversary and the blocks they generate are called Byzantine or malicious. On the contrary, miners that are not controlled by the adversary and their blocks are considered honest (*i.e.* they follow the prescribed protocol) and represent a proportion $(1 - \mu)$ of the whole set of miners. We assume that each miner (honest or not) has the same computational power. Finally, we assume a constant block generation time.

Let $B_k = (h, m)$ denote the state of the blockchain at epoch k, where h and m represent the number of honest (respectively malicious) blocks. We assume that Nakamoto, the Bitcoin system creator, is honest and thus we have $B_0 = (1, 0)$. Process $B = \{B_k \mid k \geq 0\}$ represents the evolution of the

blockchain composition over epochs. From state $B_k = (h, m)$ two transitions are possible: the next block can either be generated by a honest miner, and the blockchain goes to state $B_{k+1} = (h + 1, m)$ with probability $1 - \mu$, or generated by a malicious miner, and $B_{k+1} = (h, m + 1)$, which happens with probability μ. Process B is an homogeneous discrete time Markov chain over the discrete state space $\mathbb{N}^* \times \mathbb{N}$. Non null probability transitions are given for all $(h, m) \in \mathbb{N}^* \times \mathbb{N}$ by

$$\mathbb{P}\{B_{k+1} = (h + 1, m) \mid B_k = (h, m)\} = 1 - \mu, \quad (4)$$

$$\mathbb{P}\{B_{k+1} = (h, m + 1) \mid B_k = (h, m)\} = \mu. \quad (5)$$

B. Analysis of Bitcoin-NG Safety

As previously described, in Bitcoin-NG each epoch is led by a single miner entitled to validate the set of transactions it receives. Upon reception of a new one, the leader has to check if it is locally valid, *i.e.* if it satisfies Definition 1. If so, the leader cryptographically signs it and disseminates it. Only signed transactions may be confirmed, *i.e.* inserted in a block.

Note that if the leader is malicious, it may easily create double-spending transactions and sign them with no consideration for the other transactions whose recipients are honest. By assumption a proportion $\mu \in (0, 1)$ of miners are controlled by the considered adversary. Thus in expectation, a proportion μ of blocks are malicious as well. The evolution of the blockchain can be seen as a random walk over $\mathbb{N}^* \times \mathbb{N}$. Given $k \geq 0$, $h \geq 1$ and $m \geq 0$, and with $B_0 = (1, 0)$ as initial state, we easily derive :

$$\mathbb{P}\{B_k = (h, m)\} = \binom{k}{h-1}(1-\mu)^{h-1}\mu^m 1_{\{k=h+m-1\}} \quad (6)$$

The probability that at epoch k the blockchain does not contain any malicious block is equal to $\mathbb{P}\{B_k = (h, 0)\} = (1-\mu)^{k-1}$ if $h = k-1$ and 0 otherwise. Currently, the Bitcoin blockchain counts more than 420 000 blocks, making this probability close to 0. Furthermore, one can note the scalability issue in this protocol: in the current setting, a leader has to sign on average 1500 transactions per epoch, this volume being steadily growing. Consequently, Bitcoin-NG approach cannot cope with an adversarial environment, and hardly scales to a high number of transactions.

C. Analysis of PeerCensus Safety

Contrarily to Bitcoin-NG, PeerCensus [5] proposes to involve the whole set $\mathcal{E}_\infty$ of successful miners in a Byzantine Fault Tolerant consensus protocol like PBFT [4]. Prior to focusing on PeerCensus safety, one may easily notice that the scalability of this solution highly depends on the blockchain size. Indeed, by involving in the k-th execution of the Byzantine tolerant consensus algorithm the $k-1$ previously successful miners, this would lead, today, to a consensus run by $k \geq 420,000$ miners. The message complexity of Byzantine tolerant consensus is classically in $\mathcal{O}(k^3)$, leading these algorithms to barely scale beyond 10 participants which clearly weakens the feasibility of this approach.

Beyond this aspect, making $\mathcal{E}_\infty$ membership at the k-th execution of consensus depend on the decision obtained at the $(k-1)$-th consensus execution leads with high probability to the permanent pollution of $\mathcal{E}_\infty$. By pollution we mean the presence of more than one third of byzantine miners in $\mathcal{E}_\infty$, even if from a global point of view, the Bitcoin network contains less than one third of byzantine entities (*i.e.* $\mu < 1/3$). The following analysis proves our assertion.

According to [15], a consensus cannot be reached among n participants if more than $(n-1)/3$ participants are byzantine. We say that the state $B_k = (h, m)$ of $\mathcal{E}_\infty$ at epoch k is *polluted* if the number m of byzantine miners belonging to $\mathcal{E}_\infty$ is larger than or equal to $(k-1)/3$. Conversely, a state that is not polluted is said to be *safe*. We partition the space state $\mathbb{N}^* \times \mathbb{N}$ into two sub-spaces $\mathcal{S}_\infty$ and $\mathcal{P}_\infty$ corresponding respectively to the set of safe and polluted states. We have

$$\mathcal{S}_\infty = \{(h, m) \in \mathbb{N}^* \times \mathbb{N} \mid h \geq 2m + 1\}$$
$$\text{and } \mathcal{P}_\infty = \{(h, m) \in \mathbb{N}^* \times \mathbb{N} \mid h \leq 2m\}.$$

Thus, using Relation (6), the probability that $\mathcal{E}_\infty$ is in a safe state at epoch k is given by

$$\begin{aligned}\mathbb{P}\{B_k \in \mathcal{S}_\infty\} &= \sum_{h=1,\ 3h\geq 2k+3}^{k+1} \binom{k}{h-1}(1-\mu)^{h-1}\mu^{k-h+1} \\ &= \sum_{h=\lceil 2k/3 \rceil}^{k} \binom{k}{h}(1-\mu)^h\mu^{k-h}.\end{aligned}$$

Using the central limit theorem, we get

$$\lim_{k\longrightarrow\infty} \mathbb{P}\{B_k \in \mathcal{S}_\infty\} = \begin{cases} 0 & \text{if} \quad \mu > 1/3 \\ 1/2 & \text{if} \quad \mu = 1/3 \\ 1 & \text{if} \quad \mu < 1/3. \end{cases} \quad (7)$$

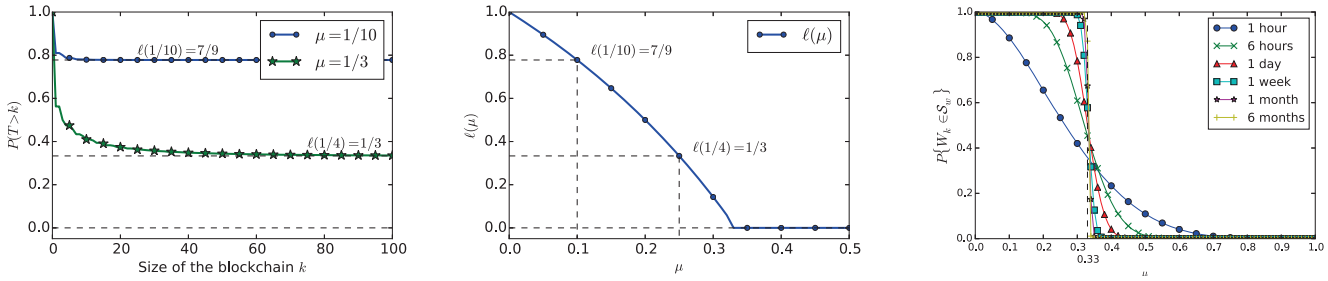
Relation (7), while in accordance with [5], does not allow one to claim that the execution that led to state B_k was safe, *i.e.*, $\forall k' \leq k, B_{k'} \in \mathcal{S}_\infty$. This argument is of prime importance, as once $\mathcal{E}_\infty$ is polluted, the adversary will be able to impose its decision at each forthcoming consensus, either on the transactions to be confirmed or on the blocks to be included in the blockchain.

We now derive the probability of k consecutive safe executions of the consensus. Let T be the number of epochs spent in states of $\mathcal{S}_\infty$ before reaching for the first time a state of $\mathcal{P}_\infty$. Formally, the random variable T is defined by $T = \min\{k \geq 0 \mid B_k \in \mathcal{P}_\infty\}$, and we have $\mathbb{P}\{T > k\} = \mathbb{P}\{B_0 \in \mathcal{S}_\infty, B_1 \in \mathcal{S}_\infty, \ldots, B_k \in \mathcal{S}_\infty\}$. Theorem 1 provides a way to compute the probability of being in a given state $B_k = (h, m) \in \mathcal{S}_\infty$ before the first corruption.

Theorem 1. *For all* $(h, m) \in \mathcal{S}_\infty$ (i.e. $h \geq 1$, $m \geq 0$ *et* $h \geq 2m + 1$) *and* $k = m + h - 1$, *we have*

$$\begin{aligned}&\mathbb{P}\{T > k, B_k = (h, m)\} \\ &= \left[\binom{k+1}{h} - 3\binom{k}{h}\right](1-\mu)^{h-1}\mu^m 1_{\{k=m+h-1\}}. \quad (8)\end{aligned}$$

Proof. We define $f(h, m) = \mathbb{P}\{T > k, B_k = (h, m)\}$ for $(h, m) \in \mathcal{S}_\infty$ (with $k = m+h-1$) and $f(h, m) = 0$ otherwise.



(a) $\mathbb{P}\{T > k\}$ as a function of μ and the blockchain size k

(b) Asymptotic behavior of $\mathbb{P}\{T > k\}$ as a function of μ

(c) $\mathbb{P}\{W_k \in \mathcal{S}_w\}$ as a function of μ and w

Fig. 2. Analysis of the safety of PeerCensus and BizCoin algorithms

The initial state being $(1, 0)$, we have $f(1, 0) = 1$. Using the Markov property, we have

$$f(h, m) = (1 - \mu)f(h - 1, m)1_{\{h \geq 2m+2\}} + \mu f(h, m - 1).$$

The relation is true for $m = 0$. Indeed the previous relation gives, for $m = 0$,

$$f(h, 0) = (1 - \mu)f(h - 1, 0)1_{\{h \geq 2\}},$$

that is $f(h, 0) = (1 - \mu)^{h-1}$, for every $h \geq 1$. Moreover, Relation (8) gives the same result for $m = 0$. We now use a recurrence on two levels. Suppose that Relation (8) is true for integer $m - 1$. For $h = 2m + 1$, we have

$$\begin{aligned} f(2m + 1, m) &= \mu f(2m + 1, m - 1) \\ &= \left[\binom{3m}{2m + 1} - 3\binom{3m - 1}{2m + 1}\right](1 - \mu)^{2m}\mu^m \\ &= \frac{(3m)!}{(2m + 1)!m!}(1 - \mu)^{2m}\mu^m, \end{aligned}$$

which is the result given by Relation (8).

Suppose that the relation is true for integers $h - 1$ and m, with $h \geq 2m + 2$. The recurrence hypothesis gives

$$\begin{aligned} f(h, m) &= (1 - \mu)f(h - 1, m) + \mu f(h, m - 1) \\ &= \left[\binom{m + h - 1}{h - 1} - 3\binom{m + h - 2}{h - 1}\right. \\ &\quad \left. + \binom{m + h - 1}{h} - 3\binom{m + h - 2}{h}\right](1 - \mu)^{h-1}\mu^m. \end{aligned}$$

Grouping the first and the third term, and the second and the fourth leads, since $k = m - h + 1$, to

$$f(h, m) = \left[\binom{k + 1}{h} - 3\binom{k}{h}\right](1 - \mu)^{h-1}\mu^m,$$

which completes the proof. □

Theorem 2 gives the distribution of the first instant T of pollution of $\mathcal{E}_\infty$, as well as its asymptotic behavior.

Theorem 2. *For all $\mu \in (0, 1)$ and $k \geq 0$, we have*

$$\begin{aligned} \mathbb{P}\{T > k\} &= \frac{1}{1 - \mu}\sum_{h=\lceil 2k/3\rceil+1}^{k+1}\binom{k + 1}{h}(1 - \mu)^h\mu^{k+1-h} \\ &\quad - \frac{3\mu}{1 - \mu}\sum_{h=\lceil 2k/3\rceil+1}^{k}\binom{k}{h}(1 - \mu)^h\mu^{k-h}. \end{aligned}$$

The limit $\ell(\mu) = \lim_{k\longrightarrow\infty} \mathbb{P}\{T > k\}$ is then given by

$$\ell(\mu) = \begin{cases} 0 & \text{if} \quad \mu > 1/3 \\ \\ 1 - \dfrac{2\mu}{1 - \mu} & \text{if} \quad \mu \leq 1/3. \end{cases} \tag{9}$$

Proof. Theorem 1, gives for all $k \geq 0$,

$$\begin{aligned} \mathbb{P}\{T > k\} &= \sum_{(h,m)\in\mathcal{S}}\left[\binom{k + 1}{h} - 3\binom{k}{h}\right] \\ &\quad \times (1 - \mu)^{h-1}\mu^m 1_{\{k=m+h-1\}} \\ &= \sum_{h=1,\ 3h\geq 2k+3}^{k+1}\binom{k + 1}{h}(1 - \mu)^{h-1}\mu^{k-h+1} \\ &\quad - \sum_{h=1,\ 3h\geq 2k+3}^{k}\binom{k}{h}(1 - \mu)^{h-1}\mu^{k-h+1} \\ &= \frac{1}{1 - \mu}\sum_{h=\lceil 2k/3\rceil+1}^{k+1}\binom{k + 1}{h}(1 - \mu)^h\mu^{k+1-h} \\ &\quad - \frac{3\mu}{1 - \mu}\sum_{h=\lceil 2k/3\rceil+1}^{k}\binom{k}{h}(1 - \mu)^h\mu^{k-h}. \end{aligned}$$

The second result is derived from the central limit theorem. □

We observe in Figure 2(a) the fast convergence of T to its limit $\ell(\mu)$, while Figure 2(b) shows that when $0 < \mu \leq 1/3$, the probability to have a series of safe consensus executions is strictly less than 1. For instance, for $\mu = 1/4 < 1/3$, we have $\ell(\mu) = 1/3$ meaning that among all the trajectories of k consensus executions, only $1/3$ of them are safe. This result clearly shows the limitations of the PeerCensus approach.

D. BizCoin

BizCoin [13] combines some of the ideas proposed in PeerCensus and Bitcoin-NG: BizCoin uses the last successful miner as the leader of the current epoch but the confirmation process is handled by tolerant Byzantine consensus executions, implemented through a cryptographic collecting signing scheme [24]. Furthermore, differently from PeerCensus, which relies on $\mathcal{E}_\infty$, BizCoin restricts the consensus membership to $\mathcal{E}_w$, containing the current leader and the $w-1$ previous ones.

We proceed as above to analyze BizCoin safety. Let us consider the random variable M correponding to the type of the current leader. According to Relations (4) and (5), at epoch k, the leader is honest, *i.e.* $M = 0$, with probability $\mathbb{P}\{M = 0\} = 1 - \mu$ and byzantine, *i.e.* $M = 1$, with probability $\mathbb{P}\{M = 1\} = \mu$. We denote by $M_{0,k}, \ldots, M_{w-1,k}$ the type of the last w leaders at epoch k. The vector $W_k = (M_{0,k}, \ldots, M_{w-1,k}) \in \{0,1\}^w$ represents the state of $\mathcal{E}_w$. The process $W = \{W_k,\ k \geq 0\}$ evolves as follows:

$$\forall k \geq 1, \forall 1 \leq i \leq w-1, M_{i,k} = M_{i-1,k-1} \tag{10}$$

where $(M_{0,k})_{k\geq 1}$ is a sequence of independent and identically distributed Bernoulli random variable with $\mathbb{P}\{M_{0,k} = 0\} = 1-\mu$ and $\mathbb{P}\{M_{0,k} = 1\} = \mu$. The process $W = \{W_k,\ k \geq 0\}$ is thus a homogeneous discrete-time Markov chain over the state space $\{0,1\}^w$, representing the evolution of the composition of $\mathcal{E}_w$ over epochs.

Similarly to Section III-C, a state W_k is *polluted* if the number of Byzantine miners belonging to $\mathcal{E}_w$ is larger than $(w-1)/3$. Conversely, a state that is not polluted is *safe*. We are interested in the number of Byzantine miners in $\mathcal{E}_w$ at epoch k. We denote this random variable by N_k which is given by $N_k = \sum_{i=0}^{w-1} M_{i,k}$. We partition the space state $\{0,1\}^w$ into two subsets $\mathcal{S}_w$ and $\mathcal{P}_w$ corresponding respectively to the set of safe and polluted states. We then have

$$\mathcal{S}_w = \{(m_0, \ldots, m_{w-1}) \in \{0,1\}^w \mid \sum_{i=0}^{w-1} m_i \leq (w-1)/3\},$$

$$\mathcal{P}_w = \{(m_0, \ldots, m_{w-1}) \in \{0,1\}^w \mid \sum_{i=0}^{w-1} m_i > (w-1)/3\}.$$

Theorem 3 derives the probability of having $\mathcal{E}_w$ in a safe state in steady state, and shows that the steady state is reached at epoch $k = w$.

Theorem 3. *For all $k \geq w$, we have*

$$\mathbb{P}\{W_k \in \mathcal{S}_w\} = \sum_{\ell=0}^{(w-1)/3} \binom{w}{\ell} \mu^\ell (1-\mu)^{w-\ell}.$$

Proof. From Relation (10), we easily get for any $k \geq 1$, $N_k = N_{k-1} + M_{0,k} - M_{w-1,k-1}$. Expanding this relation gives

$$N_k = N_0 + \sum_{\ell=1}^{k} M_{0,\ell} - \sum_{\ell=0}^{k-1} M_{w-1,\ell}. \tag{11}$$

Observing that

$$\sum_{\ell=0}^{k-1} M_{w-1,\ell} = \begin{cases} \sum_{\ell=0}^{k-1} M_{w-1-\ell,0} \text{ if } k \leq w \\ \sum_{\ell=0}^{w-1} M_{w-1-\ell,0} + \sum_{\ell=w}^{k-1} M_{0,\ell-w+1} \text{ if } k > w \end{cases}$$

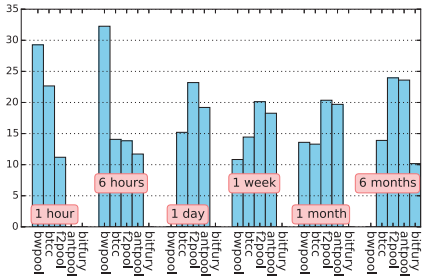


Fig. 3. Proportion of blocks mined by the most represented mining pools according to the epoch length w

and putting this last relation into Relation (11) provides the following result:

$$N_k = \begin{cases} \sum_{\ell=1}^{k} M_{0,\ell} + \sum_{\ell=0}^{w-1-k} M_{\ell,0} \text{ if } k \leq w-1 \\ \sum_{\ell=k-w+1}^{k} M_{0,\ell} \text{ if } k \geq w. \end{cases} \tag{12}$$

Thus, for $k \geq w$, N_k is the sum of w i.i.d. Bernoulli random variables with the parameter μ, so we have

$$\begin{aligned} \mathbb{P}\{W_k \in \mathcal{S}_w\} &= \mathbb{P}\{N_k \leq \lfloor (w-1)/3 \rfloor\} \\ &= \sum_{\ell=0}^{(w-1)/3} \binom{w}{\ell} \mu^\ell (1-\mu)^{w-\ell}. \end{aligned}$$

Note that this expression does not depend on k, for $k \geq w$, meaning that the stationary regime is reached at epoch w. □

Note that the result provided by Theorem 3 is consistent with Relation (7) when w tends to infinity.

Figure 2(c) depicts the proportion of safe execution of BizCoin as a function of w and μ. At the time of writing this paper, the Bitcoin blockchain contains $422,579$ blocks. We derive for $k \geq w$ values of $\mathbb{P}\{W_k \in \mathcal{S}_w\}$ for a size of 1 hour ($w = 6$), 6 hours ($w = 36$), 1 day ($w = 144$), 1 week ($w = 1008$), 1 month ($w = 4320$), 6 months ($w = 25920$). There are two trends depending on μ: against a weak adversary ($\mu \leq 1/3$), the system is safer with a large window, and conversely against a strong adversary ($\mu > 1/3$). The larger the window, the lower the variance (*i.e.* the deviation from the expected Byzantine fraction μ of $\mathcal{E}_w$); a lower variance prevents a weak adversary from randomly gaining power while a higher variance helps honest nodes to "steal" safe runs from strong adversaries.

Considering the effective power of an adversary, we investigate the Bitcoin blockchain. Over the last year, almost all blocks were generated through mining pools, which refer to groups of miners gathering their computational resources so as to increase their probability to successfully mine a block. If the block is effectively appended in the blockchain, its reward is shared among mining pool participants. Mining pools may embed a text data in blocks, allowing them to later identify all the blocks they generated. At the time of writing this paper, around 95.8% of the blocks generated over the last year contain such a text signature.

Figure 3 depicts the proportion of blocks generated by the most important mining pools, namely BWPool, BTCC, F2Pool, AntPool and BitFury, over different sizes w of set $\mathcal{E}_w$. These proportions are derived from two different blockchain trackers [1], [2]. We can note that for all sizes w that we considered for set $\mathcal{E}_w$, no mining pool has generated more than $w/3$ blocks, *i.e.* if we consider that an adversary controls the totality of a mining pool, we have $\mu < 1/3$. In this case, tuning the size of $\mathcal{E}_w$ to 1 week provides a good tradeoff between the probability of safe executions of BizCoin and the algorithmic complexity of these executions. Should these mining pools be colluding, *i.e.* under the control of a unique adversary, they would control around 60% of the miners, which clearly jeopardizes the reliability of BizCoin.

To summarize, we have shown that none of the studied solutions enhances Bitcoin's behavior. Beyond the complexity introduced by the consensus executions, the main issue comes from the fact that all important decisions of Bitcoin are solely under the responsibility of (a quorum of) miners, and the membership of the quorum is decided by the quorum members. This magnifies the power of malicious miners.

IV. Related Work

Bitcoin [20] is considered as the pioneer cryptocurrency systems. Since its inception, several altcoins [3] have emerged. Most of their differences lie in practical details like use of a database [17], block generation time [25], used hashing algorithm [16] or an unlimited number of coins [23]. The GHOST protocol [22] proposes a different rule to solve blockchain forks, based on the number of blocks contained in each blockchain subtree (in case of consecutive forks). Meanwhile, CoinJoin [18] and CoinShuffle [21] propose to mix transactions to avoid user linkability. Recent works have focused on Bitcoin modeling and evaluation. Authors of [19] prove that the Bitcoin protocol achieves consensus with high probability, while [8] show that peers participating in the Bitcoin network agree on a common prefix for the transaction history, both in failure-free environments. In contrast, authors of [10], [11] focused on adversarial environments. These works study the feasibility of double spending attacks and their detection. Finally, as analyzed in this paper, different approaches [6], [5], [13] have been proposed to enforce Bitcoin safety.

V. Conclusion

In this paper, we have formally exhibited the key concepts ruling the Bitcoin protocol. These concepts are used to derive fundamental properties of Bitcoin. To the best of our knowledge, this is the first time that they are highlighted. We then study three recent propositions aiming at enforcing strong consistency in Bitcoin. These propositions exclusively rely on miners. We have shown that *i)* none of these propositions is safe in an adversarial environment, *ii)* worse, these solutions amplify the ability of malicious users to exploit Bitcoin flaws. We are currently working on the protocol vulnerabilities related to double spending, and implementing our solution to demonstrate its feasibility and evaluate its performance in a real setting.

References

[1] Bitcoin Network Hashrate - Bitcoinity.org. https://data.bitcoinity.org/bitcoin/hashrate/, 2016.

[2] BlockTrail — Bitcoin API and Block Explorer. https://www.blocktrail.com/BTC, 2016.

[3] S. Ahamad, M. Nair, and B. Varghese. A survey on crypto currencies. In *Proceedings of the International Conference on Advances in Computer Science (AETACS)*, 2013.

[4] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 1999.

[5] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin Meets Strong Consistency. In *Proceedings of the International Conference on Distributed Computing and Networking (ICDCN)*, 2016.

[6] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse. Bitcoin-ng: A scalable blockchain protocol. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[7] N. Fincham. https://mineforeman.com/2013/03/14/what-the-fork-was-that-a-forking-post-mortem/.

[8] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques - Advances in Cryptology (EUROCRYPT)*, 2015.

[9] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The Next 700 BFT Protocols. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2010.

[10] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*, 2012.

[11] G. O. Karame, E. Androulaki, M. Rzeschlin, A. Gervais, and S. Capkun. Misbehavior in bitcoin: A study of double-spending and accountability. *ACM Transactions on Information and System Security*, 2015.

[12] E. Kokoris-Kogias, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Poster: Bitcoin meets collective signing. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2016.

[13] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, 2016.

[14] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, 2007.

[15] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 1982.

[16] Litecoin. Global Decentralized currency based on blockchain technology. https://litecoin.org, 2011.

[17] A. Loibl. Namecoin. http://namecoin.info/, 2014.

[18] G. Maxwell. CoinJoin: Bitcoin privacy for the real world. https://en.wikipedia.org/wiki/CoinJoin, 2013.

[19] A. Miller and J. LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. Available on line: http://nakamotoinstitute.org/research/anonymous-byzantine-consensus/, 2014.

[20] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[21] T. Ruffing, P. Moreno-Sanchez, and A. Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *Proceedings of European Symposium on Research in Computer Security (ESORICS)*, 2014.

[22] Y. Sompolinsky and A. Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *IACR Cryptology ePrint Archive*, 2013:881, 2013.

[23] S. N. Sunny King. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012.

[24] E. Syta, I. Tamas, D. Visher, D. Wolinsky, L. Gasser, N. Gailly, and B. Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2016.

[25] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/Paper.pdf.

A Comparison of GPU Execution Time Prediction using Machine Learning and Analytical Modeling

Marcos Amarís*, Raphael Y. de Camargo†, Mohamed Dyab‡, Alfredo Goldman*, Denis Trystram‡
* Institute of Mathematics and Statistics
University of São Paulo
São Paulo, Brazil
{amaris, gold}@ime.usp.br
† Center for Mathematics, Computation and Cognition
Universidade Federal do ABC
Santo André, Brazil
raphael.camargo@ufabc.edu.br
‡ Grenoble Institute of Technology
Grenoble, France
{mohamed.dyab, denis.trystram}@imag.fr

***Abstract*—Today, most high-performance computing (HPC) platforms have heterogeneous hardware resources (CPUs, GPUs, storage, etc.) A Graphics Processing Unit (GPU) is a parallel computing coprocessor specialized in accelerating vector operations. The prediction of application execution times over these devices is a great challenge and is essential for efficient job scheduling. There are different approaches to do this, such as analytical modeling and machine learning techniques. Analytic predictive models are useful, but require manual inclusion of interactions between architecture and software, and may not capture the complex interactions in GPU architectures. Machine learning techniques can learn to capture these interactions without manual intervention, but may require large training sets.**

In this paper, we compare three different machine learning approaches: linear regression, support vector machines and random forests with a BSP-based analytical model, to predict the execution time of GPU applications. As input to the machine learning algorithms, we use profiling information from 9 different applications executed over 9 different GPUs. We show that machine learning approaches provide reasonable predictions for different cases. Although the predictions were inferior to the analytical model, they required no detailed knowledge of application code, hardware characteristics or explicit modeling. Consequently, whenever a database with profile information is available or can be generated, machine learning techniques can be useful for deploying automated on-line performance prediction for scheduling applications on heterogeneous architectures containing GPUs.

***Keywords*-Performance Prediction, Machine Learning, BSP model, GPU Architectures, CUDA.**

I. Introduction

Today, most computing platforms for HPC have heterogeneous hardware resources (CPUs, GPUs, storage, etc.). The most powerful supercomputers today have millions of those resources [1]. In order to use all the computational power available, applications must be composed of multiple tasks that must use all available resources as efficiently as possible.

The Job Management System (JMS) is the middleware responsible for distributing computing power to applications. The JMS requires that users provide an upper bound of the execution times of their jobs (wall time). Usually, if the execution goes beyond this upper bound, the job is killed. This leads to very bad estimations, with an obvious bias that tends to overestimate their durations [2].

Graphics Processing Units (GPUs) are specialized processing units that were initially conceived with the purpose of accelerating vector operations, such as graphics rendering. GPUs are general purpose parallel processing units with accessible programming interfaces, including standard languages such as C, Java and Python. In particular, the Compute Unified Device Architecture (CUDA) is a parallel computing platform that facilitates the development on any GPU manufactured by NVIDIA [3]. CUDA was introduced by NVIDIA in 2006 for their GPU hardware line.

978-1-5090-3216-7/16/ 31.00

Information from profiling and traces of heterogeneous applications can be used to improve current JMSs, which require a better knowledge about the applications [4-5]. Predicting execution times in heterogeneous applications is a great challenge, because hardware characteristics can impact their performance in different ways. Some parallel programs can be efficiently executed on some architectures, but not on others.

Parallel computing models have been an active research topic since the development of modern computers [6-9]. Preliminary works on the characterization of the performance of GPU applications on heterogeneous platforms showed that simple analytical models can be used to predict performance of such applications [10-11].

In this paper, we implemented a fair comparison between different machine learning approaches and a simple BSP-based model to predict the execution time of GPU applications [10]. The experiments were made using 9 different applications that perform vector operations. We used 9 different NVIDIA GPUs in the experiments, 6 from Kepler and 3 from Maxwell architecture.

Our main contribution was showing that machine learning techniques provided acceptable predictions for all the applications over all the GPUs. Although the analytical model provided better predictions, it requires knowledge on the application and hardware structure. Consequently, machine learning techniques can be useful for deploying automated on-line performance prediction for scheduling applications on heterogeneous architectures containing GPUs, whenever a large data set with information about similar applications is available.

The rest of this paper is organized as follows: In Section II, we present important concepts to understand this work. In Section III, we review the literature about the area. In Section IV, we describe our experiments and methodology. In Section V, we present the results from the experiments. Finally, in Section VI, we present the conclusions of our work and future work.

II. Background

A. NVIDIA GPU Microarchitecture and CUDA

NVIDIA GPU architectures have multiple asynchronous parallel Streaming Multiprocessors (SMs) which contain Scalar Processors (SPs), Special Function Units (SFUs) and load/store units. These GPU architectures vary on a large number of features, such as number of cores, registers, SFUs, load/store units, on-chip and cache memory sizes, processor clock frequency, memory bandwidth, unified memory spaces and dynamic kernel launches. Those differences are summarized in the Compute Capability (C.C.) of an GPU.

The main advantage of GPUs is that they contains thousands of simple cores, which can be used concurrently by many threads. NVIDIA GPUs have hierarchical memory configuration with a global memory, which is shared among all threads. Concurrent accesses by threads from the same warp (groups of 32 threads) to contiguous addresses can be coalesced in a single transaction. But it has a latency of about 400 or 600 cycles per access [12]. To improve memory access efficiency, they provide a small on-chip shared memory, which has a low-latency and can be accessed by all threads in a single SM. Fermi and Kepler also provide a low-latency on-chip L1 cache, with a small access latency. A L2 off-chip cache is also present, with a latency higher than L1 cache, but lower than the global memory. Figure 1 shows the hierarchy memory accessed by any thread executed in a Kepler and Maxwell architecture.

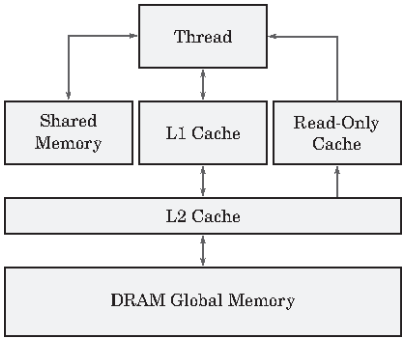


Fig. 1. Memory hierarchy for threads from a kernel executed in Kepler architectures

The CUDA programming model and platform enables the use of NVIDIA GPUs for scientific and general purpose computations. A single *master* thread runs in the CPU, launching and managing computations on the GPU. Data for the computations has to be transferred from the main memory to the GPU's memory.

B. Bulk Synchronous Parallel Model

The main goal of parallel computing models is to provide a standard way of describing and evaluating the performance of parallel applications. For a parallel computing model to succeed, it is paramount to consider the characteristics of the underlying architecture of the hardware used.

One of the most well-established models for parallel computing is the Bulk Synchronous Parallel (BSP), first introduced by Valiant in 1990 [13]. The computations in BSP model are organized in a sequence of *supersteps*, each one divided into three successive—logically disjointed—phases. On the first phase, all processors use their local data to perform local sequential computations in parallel (i.e., there is no communication among the processors.) The second phase is a communication phase, where all nodes exchange data performing personalized all-to-all communication. The last phase consists of a global synchronization barrier, that guarantees that all messages were delivered and all processors are ready to start the next superstep. Figure 2 depicts the phases of a BSP application.

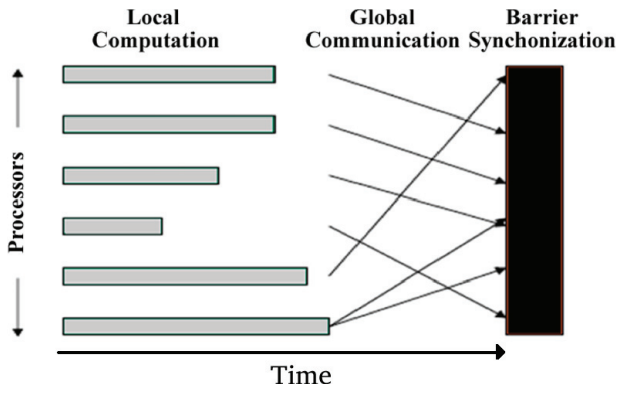


Fig. 2. Superstep in a Bulk Synchronous Parallel Model.

The cost to execute the i-th superstep is then given by:

$$w_i + gh_i + L \tag{1}$$

where w_i is the maximum amount of local computations executed, and h_i is the largest number of packets sent or received by any processor during the superstep. If $W = \sum_{i=1}^{S} w_i$ is the sum of the maximum work executed on all supersteps and $H = \sum_{i=1}^{S} h_i$ the sum of the maximum number of messages exchanged in each superstep, then the total execution time of the application is given by:

$$T = W + gH + LS \tag{2}$$

It is common to present the parameters of the BSP model as a tuple (w, g, h, L).

C. BSP-based Model for GPU Applications

In [10] the authors created a simple BSP-based model to predict performance in GPU applications. This model abstracts all the heterogeneity of GPU architectures and many optimizations that GPU application can perform in a parameter λ. We have used this model to do the comparison with the machine learning approaches. The equation 3 shows the predicted running time of a kernel T_k using this model.

$$T_k = \frac{t \cdot (Comp + Comm_{GM} + Comm_{SM})}{R \cdot P \cdot \lambda} \tag{3}$$

t is the number of threads launched, $Comp$ is the computational cost of one thread, number of cycles spent by each thread in computations, $Comm_{GM}$ is the communication cost of global memory accesses of one thread (Equation 5), $Comm_{SM}$ is the communication cost of shared memory accesses of one thread (Equation 4), R is the clock rate, P is the number of cores, λ models the effects of application optimizations.

$$Comm_{SM} = (ld_0 + st_0) \cdot g_{SM} \tag{4}$$

$$Comm_{GM} = (ld_1 + st_1 - L1 - L2) \cdot g_{GM} + L1 \cdot g_{L1} + L2 \cdot g_{L2} \tag{5}$$

where g_{SM}, g_{GM}, g_{L1} and g_{L2} are constants representing the latency in communication over shared, global, L1 cache and L2 cache memory, respectively. ld_0 and st_0 represent the average number of

load and stores for one thread in the shared memory, and ld_1 and st_1 global memory. $L1$ and $L2$ are average cache hits in $L1$ and $L2$ cache for one thread. $L1$ caching in Kepler and Maxwell architectures is reserved for register spills in local memory. For this reason $L1$ is always 0 for all the experiments. Global loads are cached in $L2$ only.

The parameter λ captures the effects of thread divergence, global memory access optimizations, and shared memory bank conflicts. t is used to adjust the predicted application execution time with the measured one and is defined as the ratio between these values. It needs to be measures only once, for a single input size and a single board. The same lambda should work for all input sizes and boards of the same architecture. For a better description of this analytical model, more info can be found in [10].

Intra-block synchronization is very fast, and did not need to be included. Nevertheless, we maintained the inspiration on the BSP-model because the extended version of the model for multiple GPUs needs global synchronizations.

D. Machine Learning

Machine learning refers to a set of techniques for understanding data. The theoretical subject of "learning" is related to prediction. Machine learning techniques involve building a statistical model for predicting, or estimating an output based on one or more inputs. Regression models are used when the output is a continuous value. In this paper, we used three different machine learning methods: Linear Regression, Support Vector Machines and Random Forest. There exists other machine learning techniques with sophisticated learning process. However, in this work, we wanted to use simple models to prove that they achieve reasonable predictions.

*1) **Linear Regression (LR)**:* Linear regression is a straightforward technique for predicting a quantitative response Y on the basis of a single or multiple predictor variables X_p. It assumes that there is approximately a linear relationship between each X_p and Y. It gives to each predictor a separate slope coefficient in a single model. Mathematically, we can write the multiple linear regression model as

$$Y \approx \beta_0 + \beta_1 X_1 + + \beta_2 X_2 + \ldots + + \beta_p X_p + \epsilon \qquad (6)$$

where X_p represents the pth predictor and β_p quantifies the association between that variable and the response.

*2) **Support Vector Machines (SVM)**:* Support Vector Machines is a widely used technique for classification and regression problems. It belongs to the general category of kernel methods, which are algorithms that depend on the data only through dot-products. The dot product can be replaced by a kernel function which computes a dot product in some possibly high dimensional feature space Z. It maps the input vector x into the feature space Z though some nonlinear mapping.

*3) **Random Forest (RF)**:* Random Forests belong to decision tree methods, capable of performing both regression and classification tasks. In general, a decision tree with M leaves divides the feature space into M regions R_m, $1 \leq m \leq M$. The prediction function of a tree is then defined as $f(x) = \sum_{m=1}^{M} c_m I(x, R_m)$, where M is the number of leaves in the tree, R_m is a region in the features space, c_m is a constant corresponding to region m and I is the indicator function, which is 1 if $x \in R_m$, 0 otherwise. The values of c_m are determined in the training process. Random forest consists of an ensemble of decision trees and uses the mode of the decisions of individual trees.

III. Related Work

Juurlink et al. were one of the firsts authors to compare performance predictions of parallel computing models [7], comparing BSP, E-BSP and BPRAM over different parallel platform. Some authors have also focused their work in performance prediction of parallel applications using machine learning [14-18]. All this work is about parallel applications executed over CPUs and not GPU applications.

In recent years, studies on GPU performance using different statistical and machine learning approaches have appeared. Baldini et al. showed that machine learning can predict GPU speedup from OpenMP applications [19]. They used K-nearest neighbor and SVM as classifier to know the performance of these applications over different GPUs. Wu et al. described a GPU performance and power estimation model [20], using K-means to create sets of scaling behaviors representative of the training kernels and neural networks that map kernels to clusters, with experiments using OpenCL applications over AMD GPUs. Karami et al. proposed a statistical performance prediction model for OpenCL kernels on NVIDIA GPUs [21] using a regression model for prediction and principle component analysis for extracting features of higher weights, thus reducing model complexity while preserving accuracy. Zhang et al. presented a statistical approach on the performance and power consumption of an ATI GPU [22], using Random Forest due to its useful interpretation tools. Hayashi et al. constructed a prediction model that estimates the execution time of parallel applications [23] based on a binary prediction model with Support Vector Machines for runtime CPU/GPU selection. Kerr et al. developed Eiger [24], which is a framework for automated statistical approaches for modeling program behaviors on diverse GPU architectures. They used various approaches, among them principal component analysis, clustering techniques, and regression analysis. Madougou et al. presented a comparison between different GPGPU performance modeling tools [25], they compare between analytical model, statistical approaches, quantitative methods and compiler-based methods. Meswani et al. predicted the performance of HPC applications on hardware accelerators such as FPGA and GPU from applications running on CPU [26]. This was done by identifying common compute patterns or idioms, then developing a framework to model the predicted speedup when the application is run on GPU or FPGA using these idioms. Ipek et al. trained multilayer neural networks to predict different performance aspects of parallel applications using input data from executing applications multiple times on the target platform [27].

In this work, we compare three different machine learning techniques to predict kernel execution times over NVIDIA GPUs. We also perform a comparison with a BSP-based analytical model to verify when each approach is advantageous. Although some works have compared analytical models, statistical approaches and quantitative methods, to the best of our knowledge this is the first work that compares analytical model to machine learning techniques to predict running times of GPU applications. Moreover, it offers a comparison between different machine learning techniques.

IV. Methodology

In this section we discuss the algorithms and GPU testbed, the analytical model and the methodology used in the learning process. During our evaluation, all applications were executed using the CUDA profiling tool *nvprof*. Each experiment is presented as the average of ten executions, with a confidence interval of 95%.

A. Algorithm Testbed

The benchmark contains 4 different strategies for *matrix multiplication* [3], 2 algorithms for *matrix addition*, 1 dot product algorithm, 1 vector addition algorithm and 1 *maximum sub-array problem* algorithm [28].

*1) **Matrix Multiplication**:* We used four different memory access optimizations: global memory with non-coalesced accesses (MMGU); global memory with coalesced accesses (MMGC); shared memory with non-coalesced accesses to global memory (MMSU); and shared memory with coalesced accesses to global memory (MMSC). The run-time complexity for a sequential matrix multiplication algorithm using two matrices of size $N \times N$ is $O(N^3)$. In a CUDA application with N^2 threads, the run-time complexity is $O(N)$

*2) **Matrix Addition**:* We used two different memory access optimizations: global memory with non-coalesced accesses (MAU); and global memory with coalesced accesses (MAC); The run-time complexity for a sequential matrix addition algorithm using two matrices of size $N \times N$ is $O(N^2)$. In a CUDA application with N^2 threads, the run-time complexity is $O(1)$.

*3) **Vector Addition Algorithm (vAdd)**:* For two vectors A and B, the Vector Addition $C = A + B$ is obtained by adding the corresponding components. In a GPU algorithm, each thread performs an addition of a position of the vectors A and B and stores the result in the vector C.

*4) **Dot Product Algorithm (dotP)**:* For two vectors A and B, the dot product $C = A \cdot B$ is obtained by adding the multiplication of corresponding components of the input, the result of this operation is a scalar. In a GPU algorithm, each thread performs a multiplication of a position of the vectors A and B and stores the result shared variable. Then a reduction per blocks is performed and a vector of size equal to the number of block in the grid is transferred to the CPU memory for later processing.

*5) **Maximum Sub-Array Problem (MSA)**:* Let X be a sequence of N integer numbers $(x_1, ..., x_N)$. The Maximum Sub-Array Problem (SSM) consists of finding the contiguous sub-array within X which has the largest sum of elements. The implementation used in this paper creates a kernel with 4096 threads, divided in 32 blocks with 128 threads [28]. The N elements are divided in intervals of N/t, and each block receives a portion of the array. The blocks use the shared memory for storing segments, which are read from the global memory using coalesced accesses. Each interval is reduced to a set of 5 integer variables, which are stored in vector of size $5 \times t$ in global memory. This vector is then transferred to the CPU memory for later processing.

B. GPU Testbed

We performed our comparisons over several different NVIDIA microarchitectures. We used 9 GPUs, described in Table I. GPUs with Compute Capability 3.X belong to Kepler architecture. GPUs with Compute Capability 5.X belong to Maxwell architecture.

TABLE I
HARDWARE SPECIFICATIONS OF THE GPUS IN THE TESTBED

Model	C.C.	Memory	Bus	Bandwidth	L2	Cores/SM	Clock
GTX-680	3.0	2 GB	256-bit	192.2 GB/s	0.5 M	1536/8	1058 Mhz
Tesla-K40	3.5	12 GB	384-bit	276.5 GB/s	1.5 MB	2880/15	745 Mhz
Tesla-K20	3.5	4 GB	320-bit	200 GB/s	1 MB	2496/31	706 MHz
Titan Black	3.5	6 GB	384-bit	336 GB/s	1.5 MB	2880/15	980 Mhz
Titan	3.5	6 GB	384-bit	288.4 GB/s	1.5 MB	2688/14	876 Mhz
Quadro K5200	3.5	8 GB	256-bit	192.2 Gb/s	1 MB	2304/12	771 Mhz
Titan X	5.2	12 GB	384-bit	336.5 GB/s	3 MB	3072/24	1076 Mhz
GTX-980	5.2	4 GB	256-bit	224.3 GB/s	2 MB	2048/16	1216 Mhz
GTX-970	5.2	4 GB	256-bit	224.3 GB/s	1.75 MB	1664/13	1279 Mhz

C. Data sets

For the analytical model, each application was executed with input sizes of power of two. For problems of one dimension, 10 samples were taken, from 2^{18} until 2^{27}. For problems of two dimensions, 6 samples were taken, all of them were squares matrices, with number of lines from 2^8 until 2^{13}.

For the machine learning analysis, we first collected the performance profiles (metrics and events) for each kernel and GPU. To be fair with the analytical model, we then choose similar communication and computation parameters to use as data input for the machine learning algorithms. We performed the evaluation using cross-validation, that is, for each target GPU, we performed the training using the other 8 GPUs, testing the model in the target GPU.

To collect data for the machine learning algorithms, we executed the two-dimensional applications using three different size for the CUDA thread blocks, 8^2, 16^2 and 32^2, and input sizes from 2^8 to 2^{13}. We took 32 samples per block size, resulting in 96 samples per GPU and a total of 864 samples. For the uni-dimensional problems we used input sizes from 2^{18} to 2^{27} and took 69 samples for each configuration, resulting in 207 samples per GPU and a total of 1863 samples. For sub-array maximum problem, 96 samples with the original configuration were taken, for a total of 864 samples.

We also evaluate a scenario were we collected more examples of a single application. We executed the matrix multiplication with shared memory and coalesced accesses (MMSC) using 8 configurations: 16, 64, 144, 256, 400, 576, 784, and 1024 threads per blocks. This resulted in a total of approximately 256 samples for GPU, and more than 2000 samples.

For each sample, the metrics, events and trace information were collected in different phases, therefore avoiding the overhead over the measured execution time of the application. The features which we used to feed the Linear Regression, Support Vector Machines and Random Forest algorithms are described in the Table II.

TABLE II
FEATURES USED AS INPUT IN THE MACHINE LEARNING TECHNIQUES

Feature	Description
`num_of_cores`	Number of cores per GPU
`max_clock_rate`	GPU Max Clock rate
`Bandwidth`	Theoretical Bandwidth
`Input Size`	Size of the problem
`totalLoadGM`	Load transaction in Global Memory
`totalStoreGM`	Store transaction in Global Memory
`TotalLoadSM`	Load transaction in Shared Memory
`TotalStoreSM`	Store transaction in Global Memory
`FLOPS SP`	Floating operation in Single Precision
`BlockSize`	Number of threads per blocks
`GridSize`	Number of blocks in the kernel
`No. threads`	Number of threads in the applications
`Achieved Occupancy`	Ratio of the average active warps per active cycle to the maximum number of warps ed on a multiprocessor.

To generate the flags `totalLoadGM`, `totalStoreGM`, `TotalLoadSM` and `TotalStoreSM`, the number of requests was divided by the number of transactions per request for each operation.

We first transformed the data to a log_2 scale and, after performing the learning and predictions, we returned to the original scale using a 2^{pred} transformation [29], reducing the non-linearity effects. Figure 3 shows the difference between the trained model without (left-hand

side graph) and with (right-hand side graph) logarithmic scale. The linear regression resulted in poor fitting in the tails, resulting in poor predictions. This problem was solved with the log transformation.

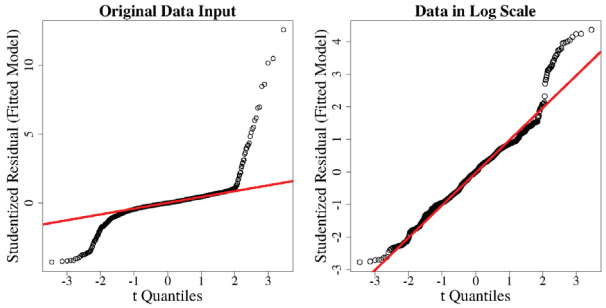


Fig. 3. Quantile-Quantile Analysis of the generated models

In this work, we applied these methods over profiling information about metrics and events of the executions of GPU applications over NVIDIA GPUs. To measure the progress of the learning algorithm we have used the normalized mean square error. With this error we have analysed the reliability of our approaches.

We used R to automate the statistical analyses, in conjunction with the `e1071` and `randomForest` packages to use the `svm` and `randomForest` functions respectively [30-31].

V. Results

The source code for all the experiments and results are available[1] under Creative Commons Public License for the sake of reproducibility. The comparison between analytical models and machine learning approaches are done taking the accuracy of the predictions, defined as the ratio between the predicted and true values of execution times, i.e, $\frac{y_{pred}}{y_{true}}$.

The rest of this section is organized as follows: In subsection V-A, the Analytical model results are presented. In subsection V-B, results for Machine Learning approach are presented. In subsection V-C, a comparison between the results of both approaches is presented.

A. Analytical Model

The number of computation ($Comp$) and communication (g_{SM}, g_{GM}, g_{L1} and g_{L2}) steps were extracted from the application source codes. These parameters are the same for all the simulations, and are presented in Table III. We did not include the values of the cache L2 for these experiments because they did not impact the execution times.

TABLE III
Values of the model parameters over 9 different applications

Par.	Matrix Multiplication				Matrix Addition		vAdd	dotP	MSA
	MMGU	MMGC	MMSU	MMSC	MAU	MAC			
comp	$N\cdot$ FMA				$1\cdot 24$			$1\cdot 96$	$(N/t)\cdot 100$
ld_1	$2\cdot N$				2			2	N/t
st_1	1				2			1	N
ld_0	0		$2\cdot N$		0			0	N/t
st_0	0		1		0			$1+log(t)$	5

Different micro-benchmarks were used to measure the number of cycles per computation operation in GPUs [32], with FMAs, additions and multiplications taking approximately 1, 24 and 96 cycles of clock. For all simulations, we considered 5 cycles for latency in the communication for shared memory and 500 cycles for global memory [3]. Finally, when the models were complete, we executed a single instance of each application on each GPU to determine the λ values, described in the Table IV.

TABLE IV
Values of the parameter λ for each application in each GPU

	MMGU	MMGC	MMSU	MMSC	MAU	MAC	dotP	vAdd	MSA
GTX-680	4.25	19.00	18.00	68.00	0.85	11.00	14.00	11.00	0.68
Tesla-K40	4.30	20.00	19.00	65.00	2.50	9.50	9.00	10.00	0.48
Tesla-K20	4.50	21.00	18.00	52.00	2.50	9.00	9.00	10.00	0.50
TitanBlack	3.75	17.00	16.00	52.00	1.85	8.00	7.00	8.50	0.35
Titan	4.25	21.00	17.00	50.00	2.50	10.00	9.50	12.00	0.48
Quadro	5.00	22.00	22.00	68.00	1.25	10.00	12.00	11.00	0.50
TitanX	9.00	38.00	38.00	118.00	2.75	10.50	7.50	10.50	1.05
GTX-980	9.00	40.00	40.00	110.00	3.25	9.75	10.00	10.00	1.65
GTX-970	5.50	26.00	24.00	75.00	1.85	5.90	7.00	6.00	1.05

B. Machine Learning Approaches

Figure 4 shows the box plots of the accuracy of the machine learning techniques using many samples. The box plots show the median for each GPU and the upper and lower first quartiles, with whiskers representing the 95% confidence interval. Outliers are marked as individual points.

In this experiment, approximately 260 samples of the application MMSC were collected in each one of the 9 GPUs. For the training set 8 GPUs were used, and the remaining GPU was used for the test set. This was made for each GPU in the three techniques of machine learning. We can see that Linear Regression, Support Vector Machines and Random Forest have a reasonable accuracy for all the GPUs, with a mean between 0.75 and 1.5, for most cases, with some outliers.

The linear kernel in the support vector machine achieved the best performance and accuracy in the prediction. For this reason, Figures 4, 5 show similar results for Linear Regression and for Support Vector Machines. Other kernel like Polynomial, Gaussian (RBF) and Sigmoid were tested, they resulted in worse predictions.

For the random forest, we have changed two default parameters, the number of trees and the number of variables as candidates at each split. For the first parameter, the default value was 500 and for the second parameter, the default value was $p/3$, where p is the number of predictors, 13 in this case according to Table II. We set the number of trees to 50, and the number of predictors to split to 5. These values achieved better prediction, and they were determined manually after many simulations.

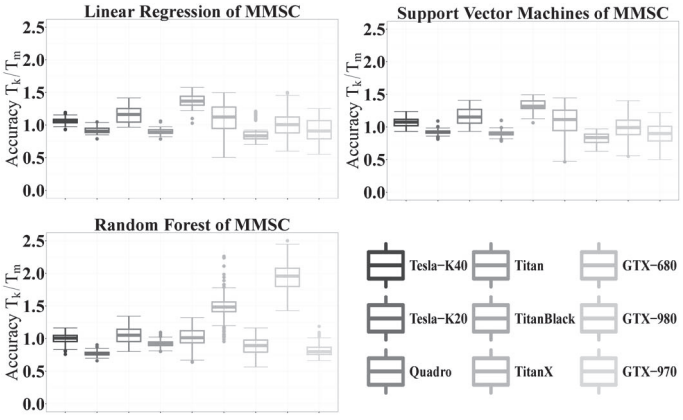


Fig. 4. Accuracy of Machine Learning Algorithms of matMul-SM-Coalesced with many samples

[1]Hosted at GitHub: `https://github.com/marcosamaris/svm-gpuperf` [Accessed on 19 June 2016]

Table V shows the comparison between the different regression models used, in terms of mean accuracy and mean squared error. In this table, we can see that the accuracy of the predictions are between 0.75 and 1.2 for almost all the cases, only the predictions of the GTX-980 with the Random Forest showed irregular predictions, we think that was because the application MMSC showed the best performance in this GPU and the selected parameters to split the decision tree lied at the moment of the predictions.

TABLE V
STATISTICS OF THE MACHINE LEARNING WITH MORE OF 1000 SAMPLES FOR TRAINING PROCESS

GPUs	Accuracy Mean			NMSE		
	LR	SVM	RF	LR	SVM	RF
GTX-680	0.85 ± 0.09	0.82 ± 0.07	0.78 ± 0.08	0.033	0.037	**0.026**
Tesla-K40	1.21 ± 0.05	1.20 ± 0.06	0.97 ± 0.06	0.006	0.008	**0.005**
Tesla-K20	0.85 ± 0.03	0.84 ± 0.03	0.77 ± 0.02	**0.008**	**0.008**	0.051
Titan-Black	1.18 ± 0.07	1.16 ± 0.06	1.12 ± 0.12	**0.145**	0.115	0.019
Titan	0.96 ± 0.04	0.96 ± 0.04	0.98 ± 0.06	0.012	0.012	**0.008**
Quadro	1.00 ± 0.10	1.01 ± 0.10	0.98 ± 0.10	0.041	0.043	**0.017**
TitanX	1.34 ± 0.28	1.30 ± 0.27	1.45 ± 0.17	0.064	**0.059**	0.254
GTX-980	1.05 ± 0.17	1.04 ± 0.17	2.08 ± 0.50	0.029	**0.027**	0.855
GTX-970	0.73 ± 0.13	0.71 ± 0.13	0.75 ± 0.08	**0.035**	0.039	0.039

C. Machine Learning VS Analytical Model

Figure 5 shows a comparison between the accuracy of the Analytical Model (AM), Linear Regression (LR), Random Forest (RF) and SVM Regression (SVM) to predict execution times of each application on each target GPU. Each box plot represents accuracy per GPU, with each column representing a different technique and each line a different application.

We used matrix and vector algorithms with regular behavior, but the usage of thread blocks of different sizes and input sizes resulted in varying levels of occupancy in the GPUs, which made the problem challenging

We could reasonably predict the running time of 9 kernel functions over 9 different GPUs using the analytical model and machine learning techniques. For the Analytical model, the accuracy for all applications and GPUs were approximately between 0.8 and 1.2, showing a good prediction capability. For the machine learning models, the accuracy for all the applications (except MAU) and GPUs for Linear Regression and Random Forest were between 0.5 and 1.5.

When using machine learning, we considered different thread blocks configurations, which resulted in nonlinear changes in the occupancy of the GPU multiprocessors, as this affects the number of active blocks and threads, and in the effective memory bandwidth. This resulted in large variations in the execution times for each application. Also, to predict the results on each GPU, we used training data from the other 8 GPUs, which caused additional errors. These factor caused some prediction errors, but for the vast majority of cases, the predictions were reasonable.

Table VI shows the comparison between both analytical model and machine learning approaches in terms of normalized mean squared error (MSE). This table shows that although the analytical model obtained the best predictions for almost all the cases, machines learning techniques also provided good predictions. Our next step is to use feature extraction to improve these predictions.

TABLE VI
NORMALIZED MSE OF THE DIFFERENT TECHNIQUES USED

Apps	NMSE			
	AM	LR	SVM	RF
MMGU	**0.0291**	0.105	0.061	0.096
MMGC	**0.0110**	0.036	0.036	0.079
MMSU	**0.007**	0.055	0.040	0.071
MMSC	**0.008**	0.046	0.044	0.097
MAC	**0.047**	0.293	0.212	0.262
MAU	0.044	0.037	**0.035**	0.114
dotP	**0.015**	0.052	0.054	0.061
VecA	**0.010**	0.021	0.018	0.062
MSA	**0.007**	0.066	0.059	0.087

VI. CONCLUSIONS AND FUTURE WORKS

We performed a fair comparison between analytical model and machine learning techniques to predict the execution times of applications running on GPUs using similar parameters to both approaches. The machine learning techniques were Linear Regression, Support Vector Machine and Random Forest.

The Analytical model provides relatively better prediction accuracy than machine learning approaches, but it requires calculations to be performed for each application. Furthermore, the value of λ has to be calculated for each application executing on each GPU.

Machine learning could predict execution time with less accuracy than the analytical model, but this approach provides more flexibility because performing specific calculations is not needed as in the analytical model. A machine learning approach is more generalizable for different applications and GPU architectures than an analytical approach.

As future work, we will consider other irregular benchmarks (Rodinia, Sparse and dense matrix linear algebra operation kernels and graph algorithms). We will also consider the scenario of multiple kernels and GPUs where global synchronization among kernels and one extra memory level, the CPU RAM, needs to be considered.

Also for the learning process in the machine learning: we will perform feature selection from a large set of features (All profiling and metrics data) to choose the most relevant ones and try them on all the regression models we tried before.

ACKNOWLEDGMENT

This project was grant-aided by São Paulo Research Foundation (FAPESP) (processes #2012/23300-7 and #2013/26644-1) by CAPES and by CNPq. Thanks to NVIDIA Corporation who donate us a some GPUs of the testbed. Experiments presented in this paper were carried out using the Digitalis platform (http://digitalis.imag.fr) of the Grid'5000 testbed. Grid'5000 is supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

REFERENCES

[1] TOP-500-Supercomputer, " [Web site http://www.top500.org] Visited May 2016." [Online]. Available: http://www.top500.org

[2] K. Gaj, T. A. El-Ghazawi, N. A. Alexandridis, F. Vroman, N. Nguyen, J. R. Radzikowski, P. Samipagdi, and S. A. Suboh, "Performance evaluation of selected job management systems," in *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, ser. IPDPS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 260–. [Online]. Available: http://dl.acm.org/citation.cfm?id=645610.660898

[3] NVIDIA, *CUDA C: Programming Guide, Version 7.*, March 2015.

[4] J. Emeras, C. Ruiz, J.-M. Vincent, and O. Richard, "Analysis of the jobs resource utilization on a production system," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, N. Desai and W. Cirne, Eds. Springer Berlin Heidelberg, 2014, vol. 8429, pp. 1–21. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-43779-7_1

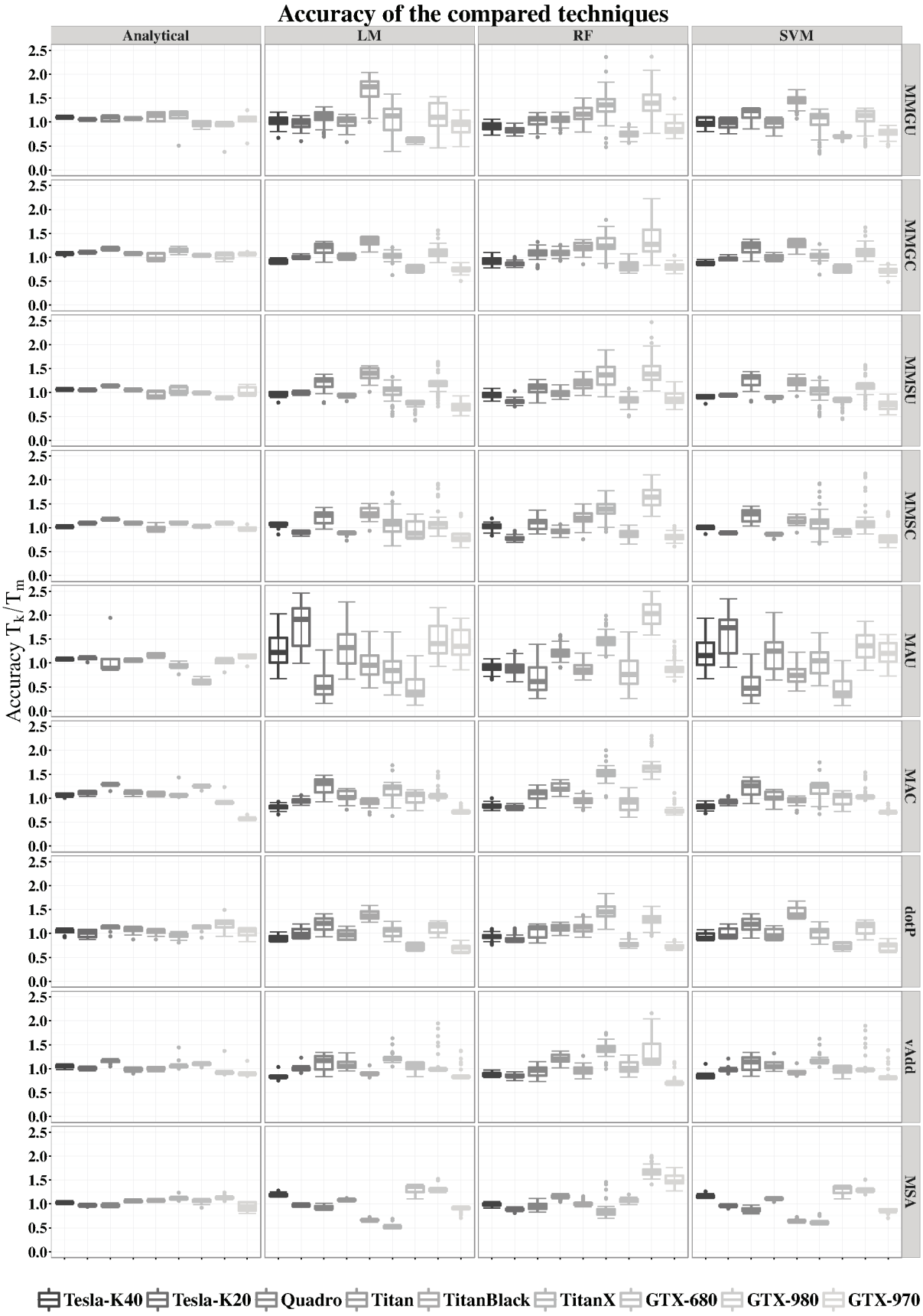


Fig. 5. Accuracy of compared techniques to predict execution times of applications on each GPU.

[5] E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 64:1–64:10. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807646

[6] P. Gibbons, Y. Matias, and V. Ramachandran, "The queue-read queue-write asynchronous PRAM model," *Theoretical Computer Science*, vol. 196, no. 1–2, pp. 3–29, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S030439759700193X

[7] B. H. H. Juurlink and H. A. G. Wijshoff, "A quantitative comparison of parallel computation models," *ACM Transactions on Computer Systems*, vol. 16, no. 3, pp. 271–318, Aug. 1998. [Online]. Available: http://doi.acm.org/10.1145/290409.290412

[8] D. B. Skillicorn and D. Talia, "Models and languages for parallel computation," *ACM Computing Surveys*, vol. 30, no. 2, pp. 123–169, Jun. 1998. [Online]. Available: http://doi.acm.org/10.1145/280277.280278

[9] A. Goldchleger, A. Goldman, U. Hayashida, and F. Kon, "The implementation of the bsp parallel computing model on the integrade grid middleware," in *Proceedings of the 3rd International Workshop on Middleware for Grid Computing*, ser. MGC '05. New York, NY, USA: ACM, 2005, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/1101499.1101504

[10] M. Amaris, D. Cordeiro, A. Goldman, and R. Y. Camargo, "A simple bsp-based model to predict execution time in gpu applications," in *High Performance Computing (HiPC), 2015 IEEE 22nd International Conference on*, December 2015, pp. 285–294.

[11] K. Kothapalli, R. Mukherjee, M. Rehman, S. Patidar, P. J. Narayanan, and K. Srinathan, "A performance prediction model for the CUDA GPGPU platform," in *High Performance Computing (HiPC), 2009 International Conference on*, 2009, pp. 463–472.

[12] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying gpu microarchitecture through microbenchmarking," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, March 2010, pp. 235–246.

[13] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990.

[14] J. Li, X. Ma, K. Singh, M. Schulz, B. de Supinski, and S. McKee, "Machine learning based online performance prediction for runtime parallelization and task scheduling," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, April 2009, pp. 89–100.

[15] B. Ozisikyilmaz, G. Memik, and A. Choudhary, "Machine learning models to predict performance of computer system design alternatives," in *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, Sept 2008, pp. 495–502.

[16] K. Singh, E. İpek, S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana, "Predicting parallel application performance via machine learning approaches: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 19, no. 17, pp. 2219–2235, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1002/cpe.v19:17

[17] E. Ipek, B. de Supinski, M. Schulz, and S. McKee, "An Approach to Performance Prediction for Parallel Applications," in *Euro-Par 2005 Parallel Processing*, ser. Lecture Notes in Computer Science, J. Cunha and P. Medeiros, Eds. Springer Berlin Heidelberg, 2005, vol. 3648, p. 196–205.

[18] A. Matsunaga and J. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, May 2010, pp. 495–504.

[19] I. Baldini, S. J. Fink, and E. Altman, "Predicting gpu performance from cpu runs using machine learning," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2014 IEEE 26th International Symposium on*, Oct 2014, pp. 254–261.

[20] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "Gpgpu performance and power estimation using machine learning," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 564–576.

[21] A. Karami, S. A. Mirsoleimani, and F. Khunjush, "A statistical performance prediction model for opencl kernels on nvidia gpus," in *The 17th CSI International Symposium on Computer Architecture Digital Systems (CADS 2013)*, Oct 2013, pp. 15–22.

[22] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and power analysis of ati gpu: A statistical approach," in *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*, July 2011, pp. 149–158.

[23] A. Hayashi, K. Ishizaki, G. Koblents, and V. Sarkar, "Machine-learning-based performance heuristics for runtime cpu/gpu selection," in *Proceedings of the Principles and Practices of Programming on The Java Platform*, ser. PPPJ '15. New York, NY, USA: ACM, 2015, pp. 27–36. [Online]. Available: http://doi.acm.org/10.1145/2807426.2807429

[24] A. Kerr, E. Anger, G. Hendry, and S. Yalamanchili, "Eiger: A framework for the automated synthesis of statistical performance models," in *High Performance Computing (HiPC), 2012 19th International Conference on*, Dec 2012, pp. 1–6.

[25] S. Madougou, A. Varbanescu, C. de Laat, and R. van Nieuwpoort, "The landscape of {GPGPU} performance modeling tools," *Parallel Computing*, vol. 56, p. 18–33, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819116300114

[26] M. R. Meswani, L. Carrington, D. Unat, A. Snavely, S. Baden, and S. Poole, "Modeling and predicting performance of high performance computing applications on hardware accelerators," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, May 2012, pp. 1828–1837.

[27] E. Ipek, B. R. de Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *Proceedings of the 11th International Euro-Par Conference on Parallel Processing*, ser. Euro-Par'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 196–205. [Online]. Available: http://dx.doi.org/10.1007/11549468_24

[28] C. Silva, S. Song, and R. Camargo, "A parallel maximum subarray algorithm on gpus," in *5th Workshop on Applications for Multi-Core Architectures (WAMCA 2014). IEEE Int. Symp. on Computer Architecture and High Performance Computing Workshops*, Paris, 2014, pp. 12–17.

[29] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ser. ICS '08. New York, NY, USA: ACM, 2008, pp. 368–377.

[30] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch, *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2015, r package version 1.6-7. [Online]. Available: https://CRAN.R-project.org/package=e1071

[31] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: http://CRAN.R-project.org/doc/Rnews/

[32] X. Mei, K. Zhao, C. Liu, and X. Chu, "Benchmarking the memory hierarchy of modern gpus," in *Network and Parallel Computing*, ser. Lecture Notes in Computer Science, C.-H. Hsu, X. Shi, and V. Salapura, Eds. Springer Berlin Heidelberg, 2014, vol. 8707, pp. 144–156.

A QoS-Aware Controller for Apache Storm

M.Reza HoseinyFarahabady*, Hamid R. Dehghani Samani†,Yidan Wang‡, Albert Y. Zomaya§, Zahir Tari¶

*,†,§ Centre for Distr. & High Performance Computing, School of IT, The Uni. of Sydney, Australia
‡,¶ School of Computer Science & IT, RMIT University, Melbourne, Australia
*,†,§{reza.hoseiny,hamid.samani,albert.zomaya}@sydney.edu.au, ‡,¶{yidan.wang,zahir.tari}@rmit.edu.au

***Abstract*—Apache Storm has recently emerged as an attractive fault-tolerant open-source distributed data processing platform that has been chosen by many industry leaders to develop real-time applications for processing a huge amount of data in a scalable manner. A key aspect to achieve the best performance in this system lies on the design of an efficient scheduler for component execution, called topology, on the available computing resources. In response to workload fluctuations, we propose an advanced scheduler for Apache Storm that provides improved performance with highly dynamic behavior. While enforcing the required Quality-of-Service (QoS) of individual data streams, the controller allocates computing resources based on decisions that consider the future states of non-controllable disturbance parameters, e.g. arriving rate of tuples or resource utilization in each worker node. The performance evaluation is carried out by comparing the proposed solution with two well-known alternatives, namely the Storm's *default* scheduler and the *best-effort* approach (i.e. the heuristic that is based on the first-fit decreasing approximation algorithm). Experimental results clearly show that the proposed controller increases the overall resource utilization by 31% on average compared to the two others solutions, without significant negative impact on the QoS enforcement level.**

***Index Terms*—Streaming Data Processing, Apache Storm, Model Predictive Control, Resource Allocation/Scheduling**

I. Introduction

In a complex stream event processing engine, data is considered as a real-time flow of events that must be analyzed on the fly [1], [2]. With such a paradigm, mainly used in big data applications, continuously produced data streams are applied and as a result new events/data streams are generated for further computation. To evaluate the success of a resource allocation strategy, three performance metrics are used to examine its adaptivity in case of workload fluctuates. These performance metrics include *processing delay* [3] (i.e. the average time needed for processing an event), *resource throughput*, and *QoS satisfaction* imposed by users [2], [4]–[6].

A number of stream processing frameworks have emerged recently [2], [7], [8]. Apache Storm is one of the popular systems which hugely attracts both industries and researchers' attention [8]. In Apache Storm ecosystem, *operators*, which are defined as the basic computational components, are aggregated into a single topology for execution. Unlike other applications, such as Bag-of-Tasks (BoT) [9] and scientific workflow [10], streaming data processing poses new challenges in terms of varying arrival rate of real-time data flows as well as ever-changing operating conditions. Therefore, achieving efficiency of scheduling decisions becomes of the key aspects of QoS enforcement, which is translated as finding an optimal placement of Storm's topology operators across active physical nodes.

978-1-5090-3216-7/16/$31.00

Devising an elastic solution that can cope with abrupt fluctuation in incoming data streams is a recent vivid research area [6]. Traditional scheduling schemes [11]–[14] rely heavily on the measurement of a set of performance metrics to make appropriate (scheduling) decisions by comparing them with another set of predefined thresholds. Such schemes suffer from a lack of adaptability to quickly respond to the live variations in the workload patterns and available resource capacity.

This paper proposes a dynamic resource allocation scheme based on a control model with predicting capability, famously known in the literature as Model Predictive Controller or *MPC*. Our aim is to build a model that represents as accurately as possible the working conditions of the Storm ecosystem. Such a scheme enables to forecast system's behavior as well interprets the near-optimal configurations by knowing the limits and errors that exist in the prediction model. Surprisingly there is only a few work in the literature that exploited such a powerful concept in scheduling/resource allocation of computer platforms [15], while its usage in other engineering disciplines is quite prevalent [16]–[18].

The proposed scheme is able to response to changes in a variation of workload or resource capacity once they are detected. It makes its decisions based on three factors: (1) prediction of the system behavior over a future time framework (e.g. the incoming workload for each stream), (2) QoS violence incidents for each individual stream measured by a newly defined metric called *QoS detriment* in which the processing *latency* of each stream is considered as the main parameter for evaluating a scheduling decision, and (3) the resource consumption of each worker node (CPU utilization). Specifically, the latter factor is reflected in the MPC cost function to ensure that the system throughput runs at the desired level (for CPU resource in our study). We also provide a detailed performance analysis of the proposed controller in comparison with state-of-the-art techniques.

The outline of this paper is as follows. Section II provides the necessary background knowledge of Apache Storm. Section III presents a formal definition of QoS detriment metric to reflect the enforcement level of each stream, and Section IV gives the details of the proposed dynamic linear control

model (based on MPC) as a solution for Storm's resource allocation problem. The performance evaluation is summarized in Section V, and Section VI provides an overview of the existing work on scheduling in data stream processing systems. We conclude this paper in Section VII.

II. Background on Apache Storm

In recent years we have seen an increasing demand for continuous processing on high-volume data, ranging from applications in the stock market, transportation management, cyber-security to manufacturing processes [2]. Apache Storm is receiving an important momentum over the past few years as a streamlined framework for dealing with real-time data processing. It offers a scalable and fault-tolerant design that is supported by a broad set of tools to address the upcoming issues in streaming data processing systems. Examples of Storm implementations, as either part or the core of computational framework, can be found in big enterprises, including Twitter, Alibaba, Baidu and Yahoo.

A. Components

Following the idea of the master/slave paradigm, nodes in a Storm cluster are categorized as either *master* or *worker* nodes. A daemon runs in the master node, called "Nimbus", enables the node to schedule tasks across other physical machines as workers. By maintaining the availability of workers through cooperation with a ZooKeeper service, Nimbus ensures a high throughput and low-latency coordination among distributed processes by providing a shared in-memory namespace [19]. On the other hand, each worker node runs a daemon, called "Supervisor", to manage the execution of the "working processes" assigned by Nimbus. Any number of working processes can be allocated to a worker node, where the number can be determined by either the available worker nodes or the scale of tasks.

While both *Nimbus* and *Supervisor* are designed to act as stateless and fail-fast daemons, *ZooKeeper* is responsible for providing fault-tolerant computing by storing the coordination information of the Storm cluster, configuration files, and status information in its memory space [8].

B. Topology and Stream

A topology in Storm terminology is simply a computational element. Being implemented as an event listener architecture, a topology is not a self-terminate component, in contrary to batch processing system (e.g. Hadoop) in which terminates itself after accomplishing the job execution. Each topology can be structured as a directed acyclic graph that provides both an overview of the computation components (operators) as well as the data flow between connected components.

Storm's data flows can be abstracted as streams, which are unbounded sequences of *tuples* (the atomic data model). Each stream is transformed by a specific logic, which is precisely defined at computational component. In Apache Storm, the source of data streams in a topology is referred as "Spout". Spout enables a topology to retrieve data from external data generators to transform it later into normalized tuples. Once a topology fed by input tuples, Spouts can emit streams along the edges of the directed graph.

Processing nodes that receive tuples from Spout are known as "Bolt". They execute a set of pre-defined functions on tuples sent by Spout or upstream Bolts. Common functions deployed at each Bolt consists of tuple exploration, join or aggregation operations. This can emit new streams for further processing in the downstream Bolts.

C. Parallelism of Topology

Storm auto parallelization serves the purpose of ensuring *high throughput* and *fault-tolerance*. Data processing operation in a Storm terminology is referred as a "task", which could be an instance of a *Spout* or *Bolt* node. Tasks defined at a computation component constitute an "executor" running on the host machine as a single thread. Such threads are executed as part of a topology identified by a "worker process". Each worker process hosts a Java Virtual Machine (JVM) to sustain a certain degree of isolation between different topologies.

By manipulating the number of tasks, executors as well as worker processes, *parallelism* in Storm topology can be achieved. In particular, the number of executors is chosen according to its execution priority or critical level of operations. Such a decision has a direct impact in improving the throughput of the available resources. In the default Storm system, the privilege of determining instance numbers of an executor is granted to users/developers who are not well aware of varying QoS objectives. This justifies our investigation for the design of a dynamically self-adjustment parallelism in Storm to meet a wide variation of QoS expectations.

D. Default Storm Scheduler

The default Storm scheduler uses Round-Robin (RR) strategy. At any given time, the scheduler aims to evenly distribute the number of executors among available worker processes, while trying to fairly allocate worker processes across physical machines. The default scheduler balances the workload amongst active hosts, which obviously results in less awareness of computational capability and/or resource demands, especially in heterogeneous Storm clusters.

The heterogeneity of physical nodes and user-defined operation demands justify the need to improve the default scheduler to fulfill the SLA and QoS. Over/Under-utilizing the hosting resources, which is a natural consequence of the naive scheduling decisions, could lead to resource wastage or even serious execution failures. For example, Storm's operations can fully stop simply because memory/computing units are not affordable for processing of further executors' demands. Conversely, an extra cost can be incurred when far more resources than the executors' expectation are provisioned by the scheduler [20]. On this basis, it can be inferred that under ever-changing operating conditions, scheduling policies must be carefully designed with a variety of QoS objectives.

III. QoS Detriment Metrics

Although *fair* allocation of available resources among submitted topologies can be seen as a reasonable strategy to ease the imbalance resource usage in a share environment [21]–[23], careful observations revealed that fairness cannot always produce a desirable output as expected in practice [24]. In reality, applications tolerate delays differently with regards to the response time. Some[1] are highly sensitive to any type of delay, while others[2] might be less sensitive to such problem. Even within the context of a single application, different users might enforce different service levels and be charged accordingly by the service provider.

Therefore, designing an efficient mechanism to respond differently to QoS violation is essential. As example, the work in [25] showed that a fair allocation of resources among applications in a shared environment does not necessarily provide an appropriate QoS satisfaction level. Therefore, a situation could be considered to be "good" as long as we have the following situation: when a data stream experiences some amount of QoS violation, other data streams should also experience almost the same QoS violation.

Even a strategy similar to the one suggested in [6], which attempts to minimize the number of QoS violations across the entire platform, can lead to undesirable practical outcomes. Let us consider a scenario where the input rates of several streams suddenly go up, hence, the scheduler struggles to assign enough resources to all streams. Inevitability, it decides to give fewer resources to some streams. If the objective function only cares about minimizing the total number of QoS violation incidents (e.g. [6]), then it might end up allocating fewer resources to those streams that are more important than others. Our aim, however, is to design an objective function that *penalizes* such decisions rather than trying to minimize the number of QoS violation incidents for more important streams. This, therefore, makes the metric similar to one presented in [25], called *QoS detriment*, more suitable to be exploited in a platform with shared resources.

In our context, there are two performance parameters that should be satisfied: (1) the average latency of each tuple[3], and (2) the average instruction per cycle (IPC) in the worker nodes. These parameters can successfully reflect both the user's satisfaction and system throughput in a variety of scenarios. The QoS enforcement, as a minimum service level contract between the system and the clients, can be quantified as a function that maps desirable level of such performance metrics into the unit interval (0,1].

Stream QoS Violation. In what follows, we introduce a metric to reflect the QoS violation for a set of streaming data during a given interval. Let ρ_s^* and $\rho_s^{\triangleright}$ be the desirable ($*$) level and the measured ($\triangleright$) values of a specific performance metric (like the average latency) for a particular stream s during an arbitrary interval, respectively. We can define $\rho_s^{*/\triangleright} = \frac{\rho_s^*}{\rho_s^{\triangleright}}$ as a building block to quantify the amount of QoS level attained by a resource allocation/scheduling (RA/S) algorithm when running multiple data streams. A "fair" scheduling suggests that the most desirable RA/S algorithm is the one that minimizes the variance of $\rho_s^{*/\triangleright}$ among different streams. However, one can argue that trying to reach such an objective to equalize the QoS violation that each stream experiences is a serious defect.

To address this deficiency, we define an alternative metric that models the amount of QoS violations from a different perspective. The owner of each data stream, say s, is asked to submit the required QoS level as a real number, say $\mathcal{Q}_s \in (0, 1]$, referred as the *QoS enforcement level* of s. We also need to design a truthful mechanism rule (like a VCG-based online mechanism) that forces users to report *truthfully* the private value of required $\mathcal{Q}_s$ for each stream. We assume such a mechanism exists for the sake of this research. To identify the set of streams experiencing QoS violations, the value of $\rho_s^{*/\triangleright}$ needs to be computed so to be compared with $\mathcal{Q}_s$. However, it is almost impossible in practice to avoid occurrence of QoS violations for every data stream. To relax such a requirement, we allow the RA/S algorithm to violate QoS level every now and then. We can define a function, denoted as $\mathcal{F}_{\Delta t}(\mathcal{Q}_s)$, to take a $\mathcal{Q}_s$ value as the input, and its output determines an upper bound for the percentage of QoS violations tolerated during an arbitrary interval Δt. One good candidate for $\mathcal{F}$ function is a simple linear rule, such as $\mathcal{F}(\mathcal{Q}_s) = 1 - \mathcal{Q}_s$. For example, a stream with an associated $\mathcal{Q}_s = 0.9$ can have the value of $\rho_s^{*/\triangleright}$ below a certain threshold of $\mathcal{Q}_s$ only during 10% of any arbitrary interval.

Equipped with the above statements, we introduce the new metric *QoS detriment* to quantify the amount of QoS violation happening during an interval $T = (t, t + \Delta t)$:

$$\mathcal{D}_T = \sum_{s \in S} \mathcal{I}_s \tag{1}$$

where S represents all streams that experience a QoS violation during T. $\mathcal{I}_s$ is the importance factor of an individual stream s. A good candidate for $\mathcal{I}$ function can be $\mathcal{I}_s = \mathcal{Q}_s$. An interpretation for this choice is as follows: if a stream s imposes a high QoS requirement ($\mathcal{Q}_s$), then this contributes more in the system's QoS detriment metric if it experiences a violation. One objective of this work is to reduce the value of $\mathcal{D}_T$ progressively.

IV. MPC Based Scheduling for Storm

The proposed resource allocation algorithm is an advanced process controlling technique based on model predictive control (MPC) method. The basic concept of MPC lies in using a dynamic model to forecast future system's behavior as well as optimize it to produce the best possible decision [26]. The MPC model can be used to (1) effectively portray the nonlinear operations of a complex system, and (2) repetitively optimize the controlling variables during short intervals, while

[1]e.g. high-frequency trading or health monitoring applications

[2]e.g. applications in social media, environmental monitoring, or network intrusion detection

[3]The time elapsed from getting a tuple and providing the result.

considering the future states. It uses a prediction model to prognosticate the alternation of dependent variables, which is often caused by changes in the input variables.

While applications of MPC controller are very well established and ubiquitous in the manufacturing industries, its usage in complex computer systems is quite new [15]. The work in [25], [27], [28] successfully applied MPC as a solution for the dynamic resource provisioning and power management problem in large cloud systems. In [29], authors designed a controller to maximize the total revenue of cloud providers, subject to capacity, QoS availability, and migration constraints. A comprehensive tutorial review in the theory and design of model predictive control systems and possible future research trends can be found in [26], [30]–[33].

A. Predictive Controller

The basic idea of a predictive controller is that at any time τ, the system output, denoted as $\mathbf{Z}(\tau)$, should follow an ideal vector/signal, denoted as $\mathbf{s}(\tau)$. Such a reference vector is also known as *set-point trajectory*. We use bold case to denote a vector (or a matrix) quantity. The main goal of the designer is to create a mechanism to force the system's output follows the set-point trajectory within the near short periods.

It is normally assumed that the system's output should approach the set-point trajectory exponentially from the current output with a response speed, denoted as T_{ref}. Particularly, if $\epsilon(\tau) = \mathbf{s}(\tau) - \mathbf{Z}(\tau)$ represents the current error value, then such an error in the next i steps must satisfy $\epsilon(\tau + i) = e^{-iT_s/T_{ref}}\epsilon(\tau)$, where T_s is the sampling interval. An important MPC characteristic is that $T_s/T_{ref} < 1$.

In a typical MPC controller, the designer must determine the values for the following three multi-dimensional vectors:

1) An internal state vector, shown by $\mathbf{X}$,
2) An input vector, shown by $\mathbf{U}$,
3) An output vector, shown by $\mathbf{Z}$.

In most cases, the input variable $\mathbf{U}$ itself consists of two disjoint sets of either *controllable* or *uncontrollable* variables, denoted a $\mathbf{U}^{\sigma}$ and $\mathbf{U}^{\omega}$ respectively.

The relationship between $\mathbf{X}$, $\mathbf{U}$, and $\mathbf{Z}$ vectors in a general discrete-time MPC model at any arbitrary time of τ: $\mathbf{X}_{\tau+1} = \mathbf{g}_1(\mathbf{X}_\tau, \mathbf{U}_\tau)$, and $\mathbf{Z}_\tau = \mathbf{g}_2(\mathbf{X}_\tau)$, where g_1 and g_2 are some (non-linear) functions that can be determined by observing the run-time behavior of a system, often referred as the system's *model*. In our study, we use a simpler version of the original MPC that omits the necessity to define the internal state vector of $\mathbf{X}$. It builds a model to directly relate the input and output vectors using a model $\mathbf{f}$ as $\mathbf{Z}_\tau = \mathbf{f}(\mathbf{U}^{\sigma}_\tau, \mathbf{U}^{\omega}_\tau)$. Interested readers are referred to [26], [32]–[34] for a deep understanding of MPC model.

The sequence of actions at any arbitrary τ in a typical MPC is as follows:

1) Obtaining the measurement of $\mathbf{Z}_\tau$,
2) Predicting the values for non-controllable input values, $\mathbf{U}^{\omega}_{\tau+1}$
3) Computing the best possible values for future values of controllable input vector, i.e. $\mathbf{U}^{\sigma}_{\tau+1} \dots \mathbf{U}^{\sigma}_{\tau+i}$,
4) Applying one step input vector, $\mathbf{U}^{\sigma}_{\tau+1}$, to the system.

At time $\tau + 1$, MPC measures the new values for output vectors as the *feedback* signal, and the whole cycle of prediction, trajectory determination, and optimization process is repeated.

The proposed solution divides the entire time into intervals with equivalent duration. We allow controller's decisions to occur only at the beginning of each interval shown by $\tau = 1, 2, \cdots$. The controllable input vector $\mathbf{U}^{\sigma}$ at any time τ consists of the following elements: (1) set of active virtual/physical machines as Storm worker nodes, shown by $\mathcal{M}_\tau$, (2) set of worker nodes appointed to run a given stream s, shown by $\mathbf{W}^s_\tau = \{\mathcal{W}^s_{1,\tau} \cdots \mathcal{W}^s_{w(s),\tau}\}$, and (3) the amount of memory and CPU in each worker nodes $\mathcal{W}^s_{i,\tau}$ which stream s is allowed to use.

B. Prediction

The prediction of future values for non-controllable input vectors, also known as *disturbances*, at f future steps is imperative in any MPC controller. At any given time τ, we need to estimate λ^s_t, the value for the arriving rate (intensity) of tuples for each stream s coming to the system at future time points $t \in \{\tau + 1, \tau + 2 \cdots \tau + f\}$.

Let Λ be the random variable associated with the arriving rate of tuples (λ) in a given interval. If the probability distribution of Λ is already known by some means, then $\hat{\lambda}$ can be estimated by applying basic probability theories on the previous observations. Otherwise, an existing prediction model (e.g. ARIMA - Auto Regressive Integrated Moving Average) can be used to estimate the value of future $\hat{\lambda}$ values through past observations.

In ARIMA model, a future value of an observation, $\lambda_{\tau+1}$, can be estimated using a series of past observations as: $\hat{\lambda}_{\tau+1} = c + \epsilon_\tau + \sum_{i=1\dots h} \alpha_i \lambda_{\tau-i} + \beta_i \epsilon_{\tau-i}$, where c is a constant and ϵ's represent errors which are i.i.d samples from a normal distribution with mean 0 and finite variance (like a white noise process). α_i's and β_i's coefficients can be computed and updated using least-squares regression method after each new observation. To predict ahead the f step values of $\hat{\lambda}_t$, we can simply repeat the above-mentioned one-step ahead prediction to obtain the multi-step prediction conditionally.

C. Cost function

A cost function in MPC penalizes derivations of the controlled output, namely $\mathbf{Z}(\tau)$, from the target signal $\mathbf{s}(\tau)$ at any given time. The measured output vector in our model consists two major components during the interval of $T = (\tau, \tau + \Delta\tau)$: (1) Average CPU utilization at each worker node w, shown by $\bar{\mathcal{U}}^{CPU}_{w,T}$, and (2) the total amount of QoS detriment, i.e. $\mathcal{D}_T$.

Ideal CPU Utilization Point. Previous studies [35] confirmed that the ideal CPU utilization in steady state working condition should be a value between 75% to 90%. We refer this ideal utilization as $\mathcal{U}^{*,CPU}$ throughout this paper. Such a limitation is, in fact, beneficial as it allows CPU to avoid

a serious issue known as "meltdown point", which is caused by exploitation of the full capacity of computing resources. Working at this utilization level is shown to be more energy-efficient than working on a higher level of CPU utilization, especially if accomplishing a job using a certain energy-cap is more important than its make-span. The right value of $\mathcal{U}^*$ depends on CPU characteristics that needs to be determined through experimental measurements.

The following equations are produced to model the CPU's utilization of each worker node close to the optimal level, while keeping the amount of QoS violation at minimum level.

$$\min \quad \sum_{T_i} \hat{\mathcal{D}}_{T_i} \tag{2}$$

$$\text{s.t.} \quad \hat{\mathcal{U}}^{CPU}_{w,T_i} \leq \mathcal{U}^{*,CPU} \quad \forall T_{i=1\cdots f}, \forall w \tag{3}$$

where $T_{i=1\cdots f}$ are the future f steps.

Equation 2 indicates that a resource allocation schema (say σ_1) is preferred to another one (say σ_2) if and only if the predicted sum of QoS detriment metric caused by σ_1 is less than the one caused by σ_2 over future f steps. Condition 3 ensures that the proposed controller avoids meltdown point issue to occur in all worker nodes.

D. Linear Dynamic Model

As a designer's knowledge about the internal structure of the platform is limited, we try to take an approach that suppresses the state variable effect (**X**) as well relate the inputs (**U**) and outputs (**Z**) together using a time-varying linear model (**f**). In this way, we perform a reasonable amount of experiments to identify system model in macroscopic scale. During the experiments, we manipulate input parameters, and measure output variables to come up with a linear model for **f**.

An overall overview of the proposed controller is outlined in Algorithm 1. In Line 3, $t_{s\downarrow}$ indicates a time that a new stream s is submitted to the system, and t_τ denotes the beginning of a time-frame. We update the system model (Line 5) when a new observation of system **Z** becomes available (Line 4). Using the proposed model as well as prediction tool, we can predict the next steps (Line 7-10). Then the total amount of QoS detriment occurring at each stream is computed (Line 11). A forward/backward dynamic programming method is used to decompose and solve recursively the optimization problem given by Equations 2 and 3 (Line 13). Finally, after making a resource allocation decision, the controller transmits a message to the worker nodes to apply the new scheduling decisions on each (Line 14).

Algorithm 1 MPC-based resource allocation & scheduling

```
1:  procedure MPC-RA(s_(τ), T_ref, Q_s, F, I, U^{*,CPU}, f)
2:      while true do
3:          at every time slot t_τ or t_{s↓} do
4:          Z ← PERCEPTOUTPUTVARIABLE()
5:          UPDATESYSTEMMODEL(Z)
6:          for each stream s do
7:              λ̂_s ← ESTIMATEARRIVALRATE(f, s)
8:          end for
9:          for each node m ∈ M, resource r ∈ CPU_m do
10:             Û^r_m ← ESTIMATERESOURCEUSAGE()
11:             D_m ← QOSDETERMINENETMODEL
12:         end for
13:         U ← COMPUTEOPTIMAL(s_(τ), Z, D, U^*, λ̂_s, Û^r_m)
14:         APPLYSYSTEMINPUT(U, T_ref)
15:     end while
16: end procedure
```

V. Experimental Evaluation

We carried out extensive experimental work to evaluate the proposed MPC-based resource allocation/scheduling algorithm. We benchmarked our work against two well-known solutions, namely the *default* Apache Storm scheduler and the *best-effort* greedy approach (presented in Section V-B).

A. Setting

We used a local virtualized cluster consists of three machines with total 96 cores, and 304GB of RAM. The first machine is a 80 hyper-threaded-core system composed of four 2.40 GHz Intel(R) Xeon(R) E7-8870 CPUs, with 30Mb LLC, 256GB of RAM, and two 2TB SCSIv3 Disks. Each of the other two machines has a 3.40 GHz i7 CPU, 16GB of RAM, 8MB LLC. We ran Xen release 4.2.0-42-generic to build a private cloud using these machines. Ubuntu 14.04.1 ran on Dom0 and it has been assigned a fixed amount of 3GB of RAM and one dedicated core during all experiments, the rest of resources used exclusively by Apache Storm cluster.

Based on the decisions made by MPC controller in the runtime, we use Xen to flexibly adjust: (1) the number of worker nodes, and (2) the amount of CPU/RAM resources that each worker node should be receiving. As an example, consider a scenario that there are initially 32 worker nodes deployed to handle some streams (each worker node has access to use one CPU core and 2 GB of RAM at the beginning). After a while assume that two new events occur: (1) the arrival rate of an important (with high QoS demand) stream, say s_1, increases abruptly, and (2) three new data streams, again with high QoS demands, submitted to the system.

Observing the new platform's conditions, the MPC controller decides to (a) assign more resources (cores and RAM amount) to those worker nodes which are responsible to serve the stream s_1, and (b) create new five worker nodes to server requests generated by new three streams. Both decisions can be accomplished straightforwardly using Xen command tool without any need to restart the worker nodes[4]. We used Xen command tools[5] to pin each worker thread to a distinct logical core, decided by MPC, too.

We evaluated the performance of underlying system for different scenarios using two metrics: the average latency

[4] `vcpu-set`, `mem-set`, and `sched-credit`

[5] `vcpu-pin`

experienced by emitted events, and the amount of system's QoS detriment metric (as an indicator of QoS violations occurred in the entire system). All evaluations were performed on a Storm cluster (version 0.94) with k worker nodes, where $5 \leq k \leq 95$. The number of slots in each worker node is a variable that is set by the MPC controller, which is below 5. We also assigned one node hosting exclusively the Nimbus and Zookeeper services. The performance of the proposed solution has been tested on a specific topology, as first introduced by [3]. Such a topology can be considered as a representative of a broad class of streaming topologies.

B. Opponents & Workload Attributes

The proposed solution is compared against two other approaches, namely *default storm*, and *best-effort*. The default Storm scheduler uses a RR (Round Robin) policy to balance the load amongst the resources. In essence, the default scheduler aims to distribute evenly the existing executors among worker processes by allocating in a fair manner worker processes across various PMs. On the other hand, the best-effort approach uses **f**irst **f**it **d**ecreasing (FFD)[6] method to determine the set of appropriate workers so to achieve a compromise between resources' usages and QoS violations. This approach tries to add an additional worker node when the total amount of QoS violation occurring continuously exceeds a certain threshold ($\in [1, 5]$ minutes). It also assigns the maximum number of bolts on an individual worker node till it becomes fully utilized.

The general topology used in this study is compatible to the one introduced in [3]. Each component c_i in the topology has an associated stage number $s(c_i)$ that represents the longest path length that a tuple should pass from a spout to a component. Components that have same $s(c_i)$ number will not communicate with each other, while components at stage $s(c_i)$ can only receive tuples from upstream components such as c_j where $s(c_j) < s(c_i)$. This model has been investigated and used extensively in other works such as [37]–[39], too.

Such a topology can be characterized by two factors: (a) number of stages (i.e. $N \in [5, 10]$) and (b) the replication parameter for each stage (i.e. $|c_j| \in \{2, 4, 8\}, 1 \leq j \leq N$). The spout executor sets its tuple rate from either a *Poisson* or *Weibull* distribution with parameters of $\lambda \in \{1, 4, 16\}$, for Poisson, or $\lambda \in \{1, 4, 16\}$ and $\beta \in \{1.5, 3\}$, for Weibull distribution, to mimic realistic scenarios (especially those with heavy-tailed traffic patterns). In Poisson process, λ represents the average number of events per millisecond. In Weibull process λ and β are the scale and shape parameters of a standard Weibull distribution. The average number of events per millisecond in Weibull process can be obtained via $\lambda\Gamma(1 + \beta^{-1})$[7].

To imitate the QoS patterns of real applications, we have used two mainstreams, namely *UniQoS* and *NormQoS*, to represent the probability distribution of QoS enforcement requested. In the *UniQoS* (*NormQoS*) schema, the QoS enforcement of each stream is taken from a *uniform* (standard normal) probability distribution over the unit interval with an average at 0.5. By changing the above parameters, we have created 54 scenarios that represent different possible generated workloads.

C. Evaluation

Three performance metrics were considered in the experimental evaluation, namely (a) the average event latency, (b) the average resource (CPU) utilization, and (c) total amount of QoS violation. We only report here those results that reflect the behavior of Apache Storm in the stable state, which occur right after passing a short period of a transient state that lasts approximately a couple of few minutes (depending on the scenario). Within such a transient period, the latency of serving requests are significantly higher than its average. After passing the transient state however, the system performance becomes and remains fairly stable. During the transient period, the proposed scheduler collects performance metrics continuously to identify and build the system model (Section IV-A). Afterwards, it converges to the performance level that is expected to work on consistently.

Figures 1 and 2 show that our algorithm outperforms both *default* and *best-effort* polices in terms of average response time (latency) and CPU utilization. The x axis represents the number of stages increases from 5 to 10. These figures report the results for different Poisson arrival rates λ and replication factors $|c_.|$ in $UniQoS$ case. The trends of latency and CPU utilization in all other scenarios are similar to ones shown in Figure 1 and 2, and we have not plotted them here.

The trend confirms that in all three policies, both the latency and CPU utilization become larger when the number of stages increases, and no anomalies can be seen in any policy. Such a result is expected as each tuple must travel more among processing units when $|N|$ increases, and each worker node needs to do more work as the total number of nodes is fixed.

By increasing the replication factor, as can be seen the trend from left to right columns in the figures, the traffic of communication patterns among bolts, spouts and executors in worker nodes becomes increasingly complex, resulting fewer intra-node passing of tuples in each worker node. This explains why the effectiveness of the default scheduler quickly dropping down. On the other hand, the proposed controller can adapt this change by assigning more resources to those workers that just close to become a bottleneck. If such workers are sitting in different hosts where the communication bandwidth became a bottleneck, the controller would shift their data streams to other nodes where the bandwidth has not congested yet.

Arrival rate. The effect of arrival rate parameter on each scheduler has also been studied here. Each column of Figures 1 and 2 report the results for a setting of same experiments that differ only on their arrival rate, e.g. λ in Poisson case. As expected, larger arrival rates provide larger latencies among all schedulers without detection of any noticeable anomaly. In

[6]FFD is a $11/9$-approximation bin packing solution: items are ordered in non-increasing order, the next item is packed into the first bin it fits [36]

[7]$\Gamma(.)$ represents Gamma function, the continuous extension of the factorial

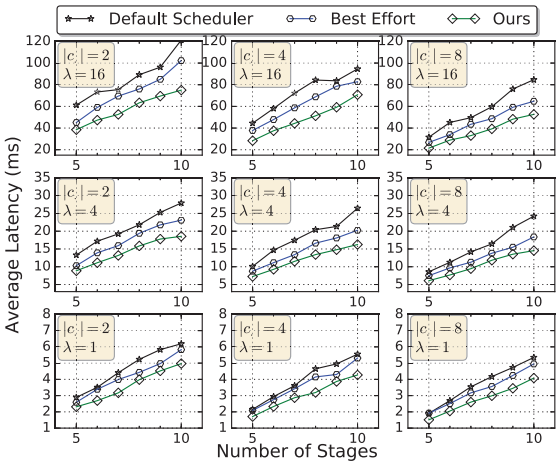


Fig. 1. Average latency achieved by *default*, *Best-Effort* and the *proposed* schedulers as the number of stages, N, varies. Nine scenarios are recognized by different values of λ_i, β_i, and $|c_.|$. QoS pattern is $UniQoS$ while the tuple rate distribution is *Poisson* for all scenarios. Maximum number of worker nodes set here to be 20.

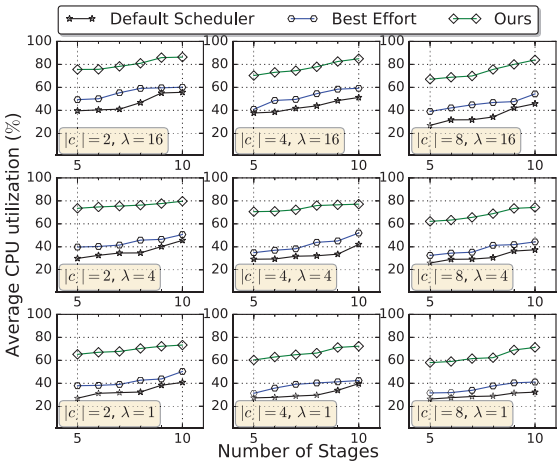


Fig. 2. Average CPU utilization of worker nodes in the stable state achieved by *default*, *Best-Effort* and the *proposed* scheduling policies as N varies. QoS pattern and tuple rate distribution are $UniQoS$ and *Poisson*, respectively, for all nine scenarios. Maximum number of worker nodes are 40. The data gathered within the first 1-hour working time-frame.

fact, a larger arrival rate decreases the inter-node communication while makes a worker node (and its associated executors) too busy to quickly respond to events (congestion situation). The proposed controller can handle this situation much better than other two opponents. By predicting the future arriving rate of tuples few steps ahead, our controller can adaptively prevent larger response time for each tuple by either switching on more worker nodes or assigning more resources to available ones.

Resource utilization. A significant achievement by applying MPC controller is its ability to keep the utilization of CPU cores of active working nodes around 80% (the ideal level), which is one of the main objectives of this work. This can be leveraged by switching off or putting other non-working cores into the deep sleeping mode to save the total power usage. Both best-effort and default schedulers are uninformed about the resources' throughput, i.e. they keep the utilization level of some cores higher than the ideal level, while let the rest of cores work on the range much below than the ideal level. Altogether, results from all scenarios confirmed that our controller achieves an average of 31% improvement in resource utilization and an average of 30% reduction in average tuple latencies comparing to the best-effort strategy, which allocates worker nodes based on the FFD method (a 1.22-approximation bin packing solution).

QoS violation. To find out the QoS detriment value for each scheduler, we need to identify the average latency achieved by tuples in each data stream. If the observed latency of a stream s within a specific time-frame is higher than an expected threshold, we count it as a QoS violation (Section III).

Table I summarizes the normalized percentage of QoS violations obtained by applying each heuristic in scenarios where the total traffic and latency are more problematic. The results

TABLE I
NORMALIZED PERCENTAGE OF AVERAGE QOS VIOLATION INCIDENTS FOR TUPLE LATENCY METRIC IN DIFFERENT SCENARIOS. MAXIMUM NUMBER OF NODES ARE 40. POIS. AND WEIB. STAND FOR POISSON AND WEIBULL ARRIVAL DISTRIBUTION, WHILE UNI. AND NORM. REPRESENT THE UNIFORM AND NORMAL QOS PATTERN, RESPECTIVELY.

Scenario Description			Norm. QoS Violation(%)		
QoS/Arrival pattern	(λ, β)	$(\|N\|, \|C_.\|)$	Ours	Best effort	Default policy
Uni./Pois.	(4, -)	(10, 2)	11	60	79
Uni./Pois.	(16, -)	(10, 2)	15	66	88
Norm./Weib.	(16, 1.5)	(10, 2)	12	71	91
Norm./Weib.	(16, 3)	(10, 2)	18	78	100

shows that the QoS-aware controller can effectively reduce the QoS violation incidents by a factor of ≥ 4 comparing to both default scheduler and best-effort heuristic in all scenarios. Our solution can keep the QoS violations below 18% even for the most extreme cases while others do not care about neither the requested QoS level nor the overall CPU throughput.

Our solution achieves this result by applying two simple methods. It first tries to avoid collocating data streams with high level of QoS enforcement in a worker node. Furthermore, if a continuous set of QoS violation incidents is detected, it dynamically makes more CPU/memory resources available to the worker node that recognized as the bottleneck, hence, avoids further QoS detriment in the upcoming time-frames.

Computational running time. The computational time of the proposed controller to calculate a solution in a system with 95 worker nodes was 0.1 seconds while the time-frame is in order of minutes[8] with almost negligible overhead time. This

[8] $\Delta T = 1$ minute throughout our experiments

suggests the applicability of proposed controller in large-scale platforms.

VI. RELATED WORK

The design of efficient scheduling schemes for real-time distributed stream processing received significant attention in recent years. Most of the studies that addressed the limitation of RR (Round Robin) default scheduler of Apache Storm, the most popular stream processing system used in the industry could be categorized to either *online* or *offline* scheduling.

In an offline scheduling, components are placed by exploring the parallel partition and data dependencies of a given topology. This aims to reduce the communication cost among connected components. For example, the offline algorithm [3] could successfully reduce the processing delays of streams comparing with Storm default scheduler. However, such offline scheduling decisions require executing before an event is triggered in lieu of making schedule decisions during execution. Hence, its limitation is quickly revealed as it fails to adapt to varying traffic conditions in run-time.

Authors in [3] proposed an online scheduling to deal with dynamic traffic conditions particularly. By measuring both inter-node and inter-slot run-time traffic patterns, they can improve by 30% the effectiveness of latency in comparison with default Storm scheduler. Another notable work in this area conducted is presented in [40], where an online mechanism is devised to automatically explore the parallel level of a given topology based on measured congestion status and throughput. This work provides a solution regarding the stateful migration if a rescheduling is happening. This issue of stateful migration has not been covered in our work, and we left it as a future work in this line.

The online approach in T-Storm [41] is also concerned with the run-time traffic patterns. T-storm enables dynamic adjustment of schedule parameters to support running fewer worker nodes while speeding up the overall time for data processing. The evaluation showed that T-storm provides 84% and 27% speedup on lightly and heavily loaded topologies, respectively, while it achieves 30% less utilization of worker nodes comparing with the default scheduler. However, T-storm does not support any mechanism to guarantee the QoS enforcements. R-storm [20] implements a resource-aware scheduling in Storm by respecting to CPU and Memory constraints, network distance between components that communicate with each other, and the variety of resource types involved. Some node and task selection algorithms, which use the minimum Euclidean distance along the axis of CPU and network dimension such that the memory constraint, are never violated. Awareness of resource makes R-storm achieve up to 47% throughput improvement and at least 69% improvement of CPU utilization comparing to Storm default scheduler. None of the above-mentioned mechanisms gives an effective solution to handle changes in the arrival rate of tuples, workload pattern, and operating conditions, however.

Some studies implied a control loop to automatically adjust resources for stream processing applications by dynamic monitoring and analysis of resource usage. As a notable example, authors in [42] implements such a control system to deal with both under- and over- provisioning of resources in a virtualized environment. They suggested that the number of processing nodes can be self-configured by uninterrupted monitoring of performance of Storm components, especially the size of queues.

More recently, authors in [15] proposed a set of proactive strategies that can dynamically adjust Storm's configurations. Unlike reactive approaches, proactive scheduling scheme take actions even before the performance metric or varying workload is observed. In particular, they adopted MPC to explore the optimal configuration of target applications (i.e. latency-sensitive applications) under ever changing operational conditions.

Similar to what we exploited in this work, the MPC-based algorithm suggested in [15] not only enables the prediction of arrival rate for incoming data, but it also forces the system to follow the set-point trajectory through adjustment of some controllable factors, which includes parallelism degree of an operator, CPU frequency and the distribution scheme. On the other hand, our study focuses more on keeping CPU utilization at the ideal level as well avoiding QoS violations. We defined the active number of the physical or virtual machine, scheduling scheme and maximum CPU/Memory usage allowed for each stream as the controlling metrics. Accordingly, the desired outputs of [15] are the reflecting response time as well as the power consumed for each set of tuples.

The algorithm in [15] also imposes a greater penalty to measured latencies that exceed defined threshold subject to QoS objectives. Compared to all other previous, this work provided an efficient elastic scaling mechanism with flexible and dynamic reconfigurability. Considering the fact that their work is presented and evaluated based on a system in the form of homogeneous infrastructure with multiple-core CPUs, we are suggesting that further experiments/adjustments are required to fully understand the unknown benefit or disadvantage of using an MPC controller in multi-node heterogeneous scenarios with QoS diversity among individual streams.

VII. CONCLUSION

Understanding run-time characteristics of individual data streams is of great practical importance to design an automatic resource allocator for distributed stream processing platforms (Apache Storm particularly). We have devised a solution based on *model predictive controller* (MPC) for achieving three goals of (1) well-utilization of resources, (2) reducing tuple response times, and (3) satisfying the QoS demand levels of each stream.

The effectiveness of the proposed solution has demonstrated its efficacy with an average improvement of 31% in total servers' resource utilization and an average of 30% reduction in average tuple latencies, comparing to the best-effort policy, a heuristic based on first fit decreasing (FFD) algorithm (a 1.22-approximation schema for bin packing problem). While

initial assessments have shown the potential benefits of our proposed technique, there are some restrictions we plan to manage in future. These include finding an automatic way to determine the optimal values of the parameters on which our method intensely depends, and managing situations where unusual *burst* streaming of data happens abruptly.

Acknowledgement

Authors acknowledge support of the Australian Research Council Linkage-Industry Grant (LP140100980) titled "Energy-Efficient Computing: Expanding the Role of Scheduling in Cloud Data Centres".

References

[1] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring streams: a new class of data management applications," in *Proc. of Intl. Conf. on Very Large Data Bases*, pp. 215–226, VLDB Endowment, 2002.

[2] H. C. M. Andrade, B. Gedik, and D. S. Turaga, *Fundamentals of Stream Processing: Application Design, Systems, and Analytics*. New York, NY, USA: Cambridge University Press, 1st ed., 2014.

[3] L. Aniello, R. Baldoni, and L. Querzoni, "Adaptive online scheduling in storm," in *Distributed event-based systems*, pp. 207–218, ACM, 2013.

[4] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *SIGMOD Rec.*, vol. 34, pp. 42–47, Dec. 2005.

[5] Y. Liu, W. Liu, J. Song, and H. He, "An empirical study on implementing highly reliable stream computing systems with private cloud," *Ad Hoc Netw.*, vol. 35, pp. 37–50, Dec. 2015.

[6] T. D. Matteis and G. Mencagli, "Proactive elasticity and energy awareness in data stream processing," *Journal of Systems and Software*, pp. 1 – 18, Aug 2016.

[7] IBM, "Infosphere platform datastage," 2016.

[8] Apache Software Foundation, "Storm, an open source distributed real-time computation system," 2016.

[9] M. HoseinyFarahabady, Y. C. Lee, and A. Y. Zomaya, "Pareto-optimal cloud bursting," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 10, pp. 2670–2682, 2014.

[10] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *J. of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.

[11] A. Kumbhare, M. Frincu, Y. Simmhan, and V. K. Prasanna, "Fault-tolerant and elastic streaming mapreduce with decentralized coordination," in *Distributed Computing Systems (ICDCS)*, pp. 328–338, 2015.

[12] B. Gedik, "Partitioning functions for stateful data parallelism in stream processing," *The VLDB Journal*, vol. 23, no. 4, pp. 517–539, 2014.

[13] T. Heinze, Z. Jerzak, G. Hackenbroich, and C. Fetzer, "Latency-aware elastic scaling for distributed data stream processing systems," in *Distributed Event-Based Systems*, DEBS '14, (NY), pp. 13–22, ACM, 2014.

[14] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch, "Integrating scale out and fault tolerance in stream processing using operator state management," pp. 725–736, 2013.

[15] T. Matteis and M. Gabriele, "Keep calm & react with foresight: strategies for low-latency & energy-efficient elastic dasp," in *Principles & Practice of Paral. Programming (PPoPP'16)*, pp. 1–12, ACM, 2016.

[16] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733 – 764, 2003.

[17] E. F. Camacho and C. B. Alba, *Model Predictive Control (Advanced Textbooks in Control and Signal Processing)*. Springer, 2008.

[18] J. A. Momoh, *Adaptive Stochastic Optimization Techniques with Applications*. CRC Press, 2015.

[19] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems.," in *USENIX Annual Technical Conf.*, vol. 8, p. 9, 2010.

[20] B. Peng, M. Hosseini, Z. Hong, R. Farivar, and R. Campbell, "R-storm: Resource-aware scheduling in storm," in *Proc. of Annual Middleware Conf.*, pp. 149–161, ACM, 2015.

[21] R. Gabor, S. Weiss, and A. Mendelson, "Fairness enforcement in switch on event multithreading," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 3, 2007.

[22] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *IEEE/ACM Intl. Symp. on Microarchitecture*, MICRO, (DC), 2007.

[23] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware shared resource management for multi-core systems," *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 141–152, 2011.

[24] R. Gabor, A. Mendelson, and S. Weiss, "Service level agreement for multithreaded processors," *ACM Trans. Archit. Code Optim.*, vol. 6, pp. 6:1–6:33, July 2009.

[25] M. Hoseinyfarahabady, Y. C. Lee, A. Zomaya, Z. Tari, and A. Song, "A model predictive controller for contention-aware resource allocation in virtualized dcs," in *Self-Aware Networks & Quality of Service (MASCOT' 16)*, (London, UK), IEEE, 2016.

[26] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control Theory & Design*. Nob Hill, 2009.

[27] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing env. via lookahead control," vol. 12, pp. 1–15, Springer, 2009.

[28] C. Wen and Y. Mu, "Power & performance management in nonlinear virtualized computing systems via mpc," *PLOS ONE*, vol. 10, no. 7, p. e0134017, 2015.

[29] E. Casalicchio, D. A. Menascé, and A. Aldhalaan, "Autonomic resource provisioning in cloud with availability goals," in *Cloud & Autonomic Computing*, p. 1, ACM, 2013.

[30] M. Morari and J. H. Lee, "MPC: past, present and future," *Computers & Chemical Eng.*, vol. 23, no. 4, pp. 667–682, 1999.

[31] S. Qin and T. A. Badgwell, "A survey of industrial mpc technology," *Control Eng. Practice*, vol. 11, no. 7, pp. 733–764, 2003.

[32] J. A. Rossiter, *Model-Based Predictive Control*. CRC Press, 2003.

[33] P. D. Christofides, R. Scattolini, D. M. de la Pena, and J. Liu, "Distributed MPC: A tutorial review," *Computers & Chemical Eng.*, vol. 51, pp. 21–41, 2013.

[34] J. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2000.

[35] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Power Aware Computing & Systems*, vol. 10, pp. 1–5, CA, 2008.

[36] G. Dosa, "The tight bound of ffd bin-packing algorithm is 11/9," in *Combinatorics, Algorithms, Probabilistic & Experimental Methodologies*, pp. 1–11, Springer, 2007.

[37] G. T. Lakshmanan, Y. G. Rabinovich, and O. Etzion, "A stratified approach for supporting high throughput event processing applications," in *Proc. of ACM Intl. Conf. on Distributed Event-Based Systems*, p. 5, ACM, 2009.

[38] A. Martin, T. Knauth, S. Creutz, D. Becker, S. Weigert, C. Fetzer, and A. Brito, "Low-overhead fault tolerance for high-throughput data processing systems," in *Distributed Computing Systems (ICDCS)*, pp. 689–699, IEEE, 2011.

[39] L. Al Moakar, A. Labrinidis, and P. K. Chrysanthis, "Adaptive class-based scheduling of continuous queries," in *Data Eng. Workshops (ICDEW)*, pp. 289–294, IEEE, 2012.

[40] B. Gedik, S. Schneider, M. Hirzel, and K.-L. Wu, "Elastic scaling for data stream processing," *IEEE Trans. on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1447–1463, 2014.

[41] J. Xu, Z. Chen, J. Tang, and S. Su, "T-storm: Traffic-aware online scheduling in Storm," in *Distributed Computing Systems (ICDCS)*, pp. 535–544, IEEE, 2014.

[42] J. S. van der Veen, B. van der Waaij, E. Lazovik, W. Wijbrandi, and R. J. Meijer, "Dynamically scaling Storm for analysis of streaming data," in *Big Data Computing Service & App.*, pp. 154–161, IEEE, 2015.

An Algorithm Based on Response Time and Traffic Demands to Scale Containers on a Cloud Computing System

Marcelo Cerqueira de Abranches
Departamento de Ciência da Computação
Universidade de Brasília
Brasília, Distrito Federal (61) 3107-6737/3658
Controladoria-Geral da União
Federal Government of Brazil
Brasília, Distrito Federal (61) 2020-6964
Email: marcelo.abranches@cgu.gov.br

Priscila Solis
Departamento de Ciência da Computação
Universidade de Brasília
Brasília, Distrito Federal (5561) 3107-6737/3658
Email: pris@cic.unb.br

***Abstract*—This paper proposes a cloud computing architecture based in containers and on an algorithm that intends to achieve an efficient allocation of processing resources to comply with response time requirements in a Web system. The algorithm is based on the characterization of web requests and on a PID (Proportional - Integral- Derivative) controller. The proposal was evaluated with a real time series obtained from an operational massive web system in a controlled infrastructure. The results show that the proposal achieves the expected response times allocating a lower number of containers than other related proposals.**

I. Introduction

Some of the most challenging and interesting topics on cloud computing environments are auto elasticity algorithms [13] and load balancing procedures. Several recent works address elasticity in cloud computing environments [13], [11], [5] [7]. Elasticity is a key feature in the cloud computing area and is the main characteristic that distinguishes this computing paradigm from the other ones such as grid computing or cluster computing.

The scalability describes the systems ability to reach a certain scale. Is the ability of the system to be enlarged as necessary, mainly to accommodate future growth adding more resources. Elasticity is a dynamic property that allows the system to scale on-demand in an operational system. Elasticity is the ability for clients to quickly request, receive, and release, many resources as needed. The elasticity implies in fluctuations, i.e., the number of resources used by a client may change over time.

The policy to implement elasticity can be manual or automatic. A manual policy means that the user is responsible for monitoring his virtual environment and applications and for performing all elasticity actions. Normally, the cloud provider provides interfaces with the system with this purpose. In automatic policy, the control is done by the cloud system, in accordance with user requirements, normally specified in the Service Level Agreement (SLA). Then, auto elasticy means to automatically adapt the environment, and even optimize resources according to the user demands.

978-1-5090-3216-7/16/$31.00

This work proposes an algorithm and an architecture to promote auto elasticity on a cloud computing environment based on the efficient allocation of resources. Our work is focused on processing power elasticity.

This paper makes the following contributions: first, we propose a cloud computing architecture, integrating several technologies to promote auto elasticity. Secondly, we analyse and characterize the web requests of a massive web system and propose an algorithm that using this typical workload allows to allocate processing resources to comply with QoS requirements, in our case, the response time. And finally, this paper evaluates the proposal in an experimental environment using several scenarios.

This work is organized as follows: section 2 presents the related work. Section 3 contains the literature review and the theoretical concepts used in our proposal. Section 4 describes the tools and technologies used in the proposed architecture. Also this section details the proposed auto scaling algorithm. Section 5 presents the experimental results. Finally, section 6 presents the conclusions and future work of this research.

II. Related Work

The work [11], compares different methods to obtain auto elasticity on a cloud computing environment. These methods can be classified into 2 categories: reactives and predictives. Those techniques are based on machine learning, queueing theory, control theory, temporal series analysis, among others.

The work [5] proposes an algorithm called PRESS to predict CPU loads by extracting consumption patterns and adjust resource allocation. Their approach uses two methods to perform online predictions: the first is based on the use of signal processing (Fast Fourier Transforms -FFT) to extract dominant frequencies. This frequencies are used to generate a time series and different time windows are compared. The Pearson correlation index is generated for various windows. If

it is obtained a Pearson correlation index greater than 0.85, the average value of the resources in each position of the time series is used to generate a forecast to the next window and the virtual machine's resources are adjusted. If a pattern is not identified, they propose an approach that uses a Markov chain with finite number of states to perform the forecast.

Another work that proposes auto elasticity is Haven [13]. This proposal is based on monitoring CPU and memory loads for each virtual machine in a load balancing pool. From CPU and memory thresholds previously established, Haven loads new virtual machines and insert them in the load balancing pool. In addition, the load balancer which is implemented with SDN (Software Defined Network), directs each request to the least loaded member of the pool.

The work [7] proposes and provides a tool called HPA (Horizontal Pod Autoscaler). This tool works by scaling the environment (number of containers) from CPU average thresholds of containers that serve a given application.

Our proposal is different from PRESS, as it does not perform load prediction, but rather, performs resource allocation or deallocation, based on the response time observed from an application behind a load balancer, prior to a workload characterization. Our proposal is reactive since it uses the variation of the average response time of an application to decide using control theory the allocation or deallocation of resources. Also, our proposal is different from PRESS that performs vertical scaling. Our proposal performs horizontal scalability, that is, new instances are allocated behind a load balancer. As our proposal, Haven also performs horizontal scalability but with virtual machines and our proposal uses containers [2].

Another difference of our proposal regarding to HPA is that while HPA performs CPU consumption measurements inside containers, our proposal performs measurements outside of containers, i. e., in the load balancer. This approach has the advantage to watch system performance, regardless of the level of resource utilization of containers. In Section 5 a comparison is made between the HPA and the solution proposed in this paper .

III. Theoretical Concepts and Literature Review

A. Containers

Container is a technology for creation of isolated processing instances and enables virtualization at the operating system level to provide protected processing portions and when running on the same system, they are not aware that are sharing resources as each one has its own network abstraction layer, memory and processes[2].

Containers have a great portability, because they can run on any operating system based on Linux. Virtualization depends on a hypervisor to achieve similar portability. Virtualization via hypervisors consumes more resources than containers. If a container is not performing any task, it is not consuming resources on the server [2]. Besides that, containers are very dynamic to be created and destroyed, as they just have to start or destroy processes in its isolated space.

B. Web Servers and Load Balancers

In our proposal, the infrastructure hosting the web system must be prepared to meet the demand with high performance, scalability and high availability. However there are many challenges to be addressed so that a cluster of distributed servers can function efficiently as if it were a single server. These challenges range from the routing of requests to the members of the cluster, methods for choosing the member to receive the workload and methods to maintain the connection status.[4].

In load balancing at the transport layer, the requests are distributed among the members of the cluster based on informations like IP addresses and ports. The load balancer distributes client connections, which must know the IP address cluster, among the various servers that effectively respond the requests. In this case, as the load balancing process is based on layer 4, the server is selected regardless of the content or the type of request. [1]. When load balancers are in layer 5, the distribution of workloads is based on the contents of the requests.

C. PID Controllers

PID controllers (Proportional - Integral- Derivative) are control algorithms that are widely used in industry. Examples of its applications are temperature control environments, and drone control. PID controllers have three coefficients : proportional , integral and derivative. These coefficients are varied in order to obtain the desired optimum control response for a given process.

A PID controller works in a closed loop system where it is possible to read the current state of a particular variable that is being controlled, and according to its value, an action is performed so that the variable of interest converges and remains on a desired level (even considering external disturbances), for the next iterations of time. [8]

Thus, the PID controller should read the current state of the variable and calculate the output response, by calculating the proportional, integral and derivative components and then adding the three for calculating the control output. The proportional component depends on the difference between the desired value (*setpoint*) and the current value of the variable. This difference is referred to as an error. The integral component adds the error term over time. The derivative response is proportional to the rate of change of the process variable.

In order to produce the necessary adjustments to the system, the PID controllers use gain parameters K_p K_i and K_d that should be adjusted. There are several methods for adjusting these parameters, such as the manual method (guess and check)) and the Ziegler -Nichols method [8].

In the manual method, the gains of each component are adjusted using trial and error. For this, the effects that each parameter causes the controller output must be known. In this method , the terms K_i and K_d are set to zero, and the term K_p is increased until the cycle output starts to oscillate. From there, the term K_i should be slowly raised to reduce the steady error. At this point the term K_d is incremented, in order to decrease the oscillations at the cycle output. The discussion on

other methods of parameter adjustment is beyond the scope of this paper, since we used the manual adjustment method.

IV. The Proposed Solution

Our proposal is based on a cloud computing environment based on containers and a method of auto elasticity to comply with a required system response time. For this purpose, we use a closed loop system with a PID controller, which responds to changes to system response time by increasing or decreasing the number of containers in a load balancing cluster that process web requests.

In the next subsections, we describe the tools that were integrated and the implementation of the algorithm to provide the features that enable our proposal.

A. Docker

Docker started as a project of the PaaS company (Platform as a Service) dotCloud in 2013 [12] proposing to be an integrator and facilitator for adoption of containers in production environments and in large scale. Docker uses kernel features to isolate the containers from the server, creating isolated processes, network and privileges. The limitation and accounting of resources (CPU, memory, disk space and I/O) is made through the use of cgroups. Also the use of the file system is done efficiently because it is based on copy-on-write, which allows changes to a container to be simply a differential update of the previous image.

One of the greatest advantages of Docker is the ability to find, download and start images of containers that were created by other developers very quickly and conveniently.

B. Kubernetes

Kubernetes is a system developed by Google [6], and made available to the community, which aims to manage the life cycle of containers in the nodes of a cluster. Thus, Kubernetes is an orchestrator of containers, being able to schedule the launch of containers between the nodes of a cluster, to do admission control of containers, resource balancing and provides scalability to the environment. Kubernetes also provides features such as service discovery between containers, service publication for access from outside the cluster and load balancing between containers [6].

The infrastructure of a Kubernetes cluster is composed of master nodes that control worker nodes, which run the containers. All settings of the cluster are stored in a distributed configuration repository, called Etcd. PODs are the basic unit within Kubernetes. Containers are grouped in PODs and these generally represent an application. These are created using Replication Controllers which are used to define PODs that can be scaled horizontally. Replication Controllers are also responsible for maintaining the desired number of PODs active in a cluster.

C. Apache Spark, Flume, HAproxy and Redis

Apache Spark is a distributed processing tool, ideal for processing large databases. It was developed by AMPLab (UC Berkeley) and performs data processing in memory by default.

Its basic structure of abstraction are the RDDS (Resilient Distributed DataSets), which are collections of elements that can undergo operations in parallel, making it possible to generate new RDDS from transformations such as map, reduce, filter and join on RDDS [16]. This tool was chosen to integrate the solution because it allows the solution to perform processing of large databases in text format, in a scalable way.

Apache Spark offers an API called Spark Streaming, which allows real-time data processing, through the creation of structures called DStreams (discretized streams), which are sequences of RDDs. The creation of DStreams is made by the StreamingContext class where you can configure the duration of each window of DStreams [16]. In our proposal, the duration of each window was set to 5 seconds and the motivations for this are further detailed in the next section.

In our proposal, the Spark Streaming is used to process the load balancer logs, collecting the response time of each of the requests that the system serves, in real time. This response time information is stored in a time series format on a Redis server, so that it can be used by the auto scaling algorithm proposed in this article.

Spark receives the log entries of the load balancer in text format, using the Flume tool. Flume is a service that aggregates, collects and moves large volumes of data flow. For its operation it creates a source that receives the data of interest. This source is connected to a channel, where the data will travel toward a sink [3].

Therefore, the function of this tool in the solution, is to send, in real time, the load balancer access logs to Spark, through the creation of a source of type Syslog, which receivers the Load Balancer logs, and a memory channel that carries the source data in memory. From there, Spark consumes this data stream through an Avro Sink, which travels over the network.

The load balancer used in the solution is HAproxy, which can act as a layer 4 or 7 load balancer, SSL terminator, reverse proxy and other [17]. Currently, this is the load balancer used by web sites as Reddit, Stack Overflow, Server Fault, Instagram among others, and has been chosen as the cloud load balancer of Red Hat's OpenShift. HAProxy has the following log format:

May 18 06:24:25 10.125.7.229 haproxy[1078]: 10.125.8.252:43839 [18/May/2016:06:24:24.988] cherrypy cherrypy/10.125.7.227 0/0/2/26/28 200 169 - - —- 1/1/1/0/0 0/0 "GET /generate HTTP/1.0"

In the above line, there are informations such as the waiting time in queue at the application server, *http* method and response code, among others. The part with *0/0/2/26/28* contains the information: *Tq '/' Tw '/' Tc '/' Tr '/' Tt*, where *Tr* is the time in milliseconds that the load balancer waits until it receives a complete response of a web request to the server [17]. So this represents the total time of the request processing by the container.

In our proposal, the SparkStreaming processes the log entries and separates the field *Tr* and stores it in the Redis database, in a time series format. SparkStreaming is also responsible for converting the format of the date of each line to the number of seconds from 0 hour of every day, to support the creation of time series.

Redis is used to store the time series, because it can provide low latency both for writing and reading, as it keeps the data as memory structures [14]. The integration of the solution with Redis is made through the use of the Kairos library, which creates a structure for storing time series in databases such as Redis, Mongo, SQL or Cassandra [9]. This library provides features, such as setting the number of entries to keep in the database and the minimum time unit of interest of a series. In the case of this work we keep stored in the series data of the last 600 seconds, which is sufficient for the algorithm operation.

Kairos also allows the calculation of statistical parameters of the series, using configurable time windows. This allows, for example, to compute average response time of an application in the last two minutes. The minimum unit of time set in this solution is 1 second. This allows good flexibility for configuring the time windows for statistical calculations and enables the generation of graphics with good time resolution for monitoring and evaluation of the solution.

D. The Cloud Architecture

The proposed architecture is shown in Figure 1 and described in the PAS pseudocode in this section. The PID controller is used to maintain the average response time of a particular application within a certain threshold. Our proposed architecture, hereby called as PAS (PID based Autoscaler) operates based on the following sequence:

1) Establishment of an average time threshold (setpoint) desired to the system to answer requests. The monitor receives the response time of the requests arriving at the load balancer;
2) The monitor sends the average response time (the average of the last 200 seconds) so that the PAS algorithm calculates the number of containers needed to reach the setpoint. The average of the last 200 seconds was defined because it was found experimentally that this value is adequate since avoids that outlier values influence on the operation of the system.
3) PAS runs algorithm 1 and inform the desired number of containers to Kubernetes. The current number of containers is obtained using a Kubernetes tool called kubectl, and the average response time of the cluster is determined using the time series present in the Redis database.
4) Kubernetes creates, mantains, or removes new containers, and ensures that the environment will remain with the desired number of containers until the next round of the algorithm (in this case, after 10 seconds). This value of 10 seconds was chosen because it was found that it is sufficient for Kubernetes to load new containers.

E. Solution Operation

In order to allow the operation of the algorithm which works with dynamic data received in real time, we use the processing flow shown in Figure 2. Haproxy acts as the load balancer of the solution. Its logs are sent to Flume that puts the data in a memory channel and sends it to Spark Streaming,

Algoritmo 1: PAS

Input: Average response time of the cluster, Current number of containers
Output: Desired number of containers

begin
- Read the desired threshold of average response time of the requests: $t\_ms\_desired$
- Read the current number of containers: $n\_containers\_current$
- Read the average response of the cluster in ms: $t\_ms\_current$
- Calculate the error: $e(t)$ = $t\_ms\_desired$ - $t\_ms\_current$
- Calculate the PID output:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)\delta t + K_d \frac{d}{dt} e(t) \quad (1)$$

- $n\_desired\_containers$=$n\_containers\_currrent$ + $u(t)$

end
return *n_desired_containers*

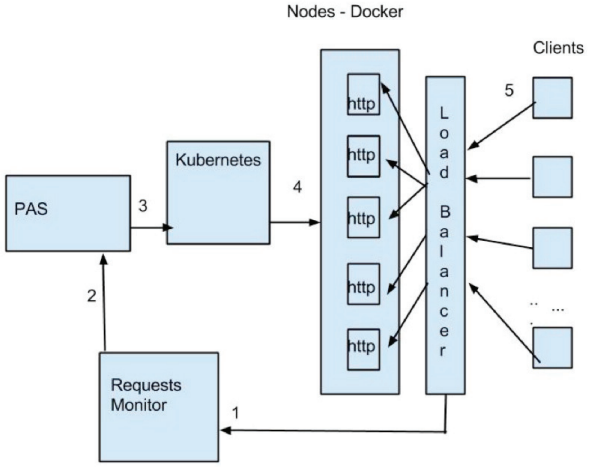


Figure 1. PAS Architecture

thus providing the information necessary for operation of the algorithm.

Haproxy balances the requests in round robin mode between the Docker/Kubernetes nodes hosting the containers. When the Docker/Kubernetes node receives the request, the Kubernetes proxy performs load balancing between the containers of each server.

So, two levels of load balancing are performed, one between the Docker/Kubernetes nodes, where Haproxy performs the load balancing, and other internally within the Docker/Kubernetes nodes where the service Kubernetes proxy performs the load balancing.

We have developed in this research a set of specific codes to customize the interaction between the tools, for example, to generate the time series with system response times and for the creation and destruction of containers, among others.

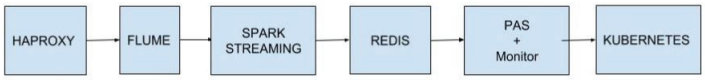


Figure 2. Processing Flow Between the Set of Tools

V. Experimental Results

A. Environment and Evaluation Scenarios

To evaluate the solution we configured a Kubernetes v1.1.2 cluster on the top of Coreos (899.6.0 (2016-02-02)) operating system, virtualised with VMWare ESXi 5.5.0. This environment was configured for solution validation. A production environment could benefit more if the system Coreos was installed directly on physical machines because the virtualization layer would be eliminated.

The cluster was built with the following components: 1 master node (4 vCPUs, 6 GB of RAM) , 1 Etcd node (4 vCPUs, 6 GB of RAM) and 4 worker nodes (4 vCPUs, 6 GB of RAM). Ubuntu 14.04.3 LTS Virtual machines (VMWare ESXi 5.5.0), with the following settings and tools: 1 haproxy 1.5.4 node (4 vCPUs , 4G GB of RAM) , 1 Spark 1.5.2 + Redis 2.8.4 node (2 vCPU 10 GB of RAM) and 1 Flume 1.7.0 + PAS node (2 vCPU , 4 GB RAM)

We defined 4 evaluation scenarios. For the scenarios 1, 2 and 3, we generated an image of a container that runs the web server Cherrypy 5.1.0. We configured a link on this server. The link generates in each request an array of random size between 1,000 and 10,000 elements.

The service publication in Kubernetes was made through the creation of a Replicaction Controller and the configuration of a service of the type NodePort. The HAProxy load balancer was configured to balance requests between the Docker/Kubernetes nodes using the IP addresses of the nodes and ports published by the service of the type NodePort. Each container had its processing and memory resources limited to 18 MB of RAM and 24 millicores of CPU.

Scenario 4 was evaluated with a more elaborate Web system than the arrays generator of scenarios 1, 2 and 3. The evaluation was made using the workload Rubis [15], which is modeled to be a clone of eBay (www.ebay.com). Rubis implements the basic features of ebay: product registration, sale, bidding, browsing products by region (United States) and categories. The installed version of Rubis 1.4.3 was obtained in https://github.com/sguazt/RUBiS.

In the tests we used the PHP version of Rubis and a MySQL 5.5 database. MySQL was installed in a virtual machine with Ubuntu 14.04.1 (16 vCPU and 4 GB of RAM). MySQL has been configured to allow caching of Rubis tables. These high settings of CPU and RAM of the MySQL virtual machine were carried out to ensure that there would be no bottlenecks in access to the application database, since the purpose of the tests is to test Web service auto scaling. The database was populated from the dump obtained in http://download.forge.ow2.org/rubis/rubis_dump.sql.gz.

As in the configuration described for scenarios 1, 2 and 3, the service publication in Kubernetes was made through the creation of a Replicaction Controller and the configuration of a NodePort service. The HAproxy load balancer was configured to balance requests between the Docker/Kubernetes nodes using the IP addresses of the nodes and ports published by the NodePort service. Each container had its processing and memory resources limited to 500 MB of RAM and 160 millicores of CPU.

The PID in the PAS algorithm utilized the following parameters: Kp = 0.016 , Ki = 0.000012 and Kd = 0.096, which was set after several tests with the workloads, adjusting the parameters using the manual method, or *guess and check* , as described in section III-C.

B. Workload

The workload generation was made using "ab" tool (apache bench). In order to generate a load with a realistic profile we collected a set of accesses of the Portal da Transparência (www.transparencia.gov.br), between May/2016 and June/2016. The captured time series in the range of 1 second is the number of accesses on that time interval. The series was characterized with Kettani-Gubner method [10]. The self-similarity and long dependence of the series was confirmed with the Hurst parameter H = 0.87 in the scales of 1 second, 10 seconds and 100 seconds.

A sample of the obtained series can be seen in Figure 3. This series is used to generate in every second, simultaneous requests directed to the IP address of the load balancer which distributes requests to the nodes of the Kubernetes cluster. The workload intensity was set at 3 levels by multiplying the time series by 1, 1.5 and 2, and preserving the same self-similarity index. These loads are referred in the experiments as load_1 , load_1.5 and load_2 .

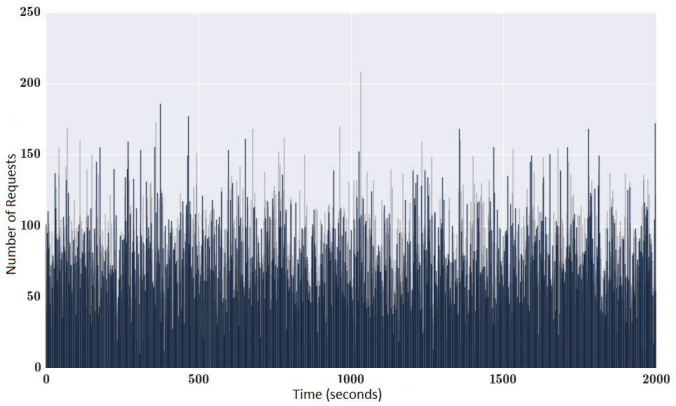


Figure 3. Workload Sample, H=0.87

1) Scenario 1: In this scenario the response time threshold at the load balancer (*setpoint*) was set at 50 ms and we applied load_1 and load_1.5. Figure 4 shows the system response time while under load_1. The graph shows the adjustment caused by the allocation of containers performed by the PAS algorithm and stability achieved close the setpoint of 50 ms.

Figure 5 shows the allocation of containers. The number of containers at the beginning of the experiment was equal to 2. The system allocated the required number of containers so that the average response time shown in Figure 4 reached

the setpoint. At the end of the run there were 26 allocated containers .

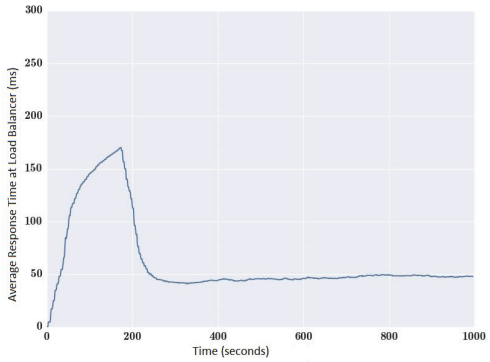


Figure 4. Response Time (ms) x Time (s), load_1, *setpoint* 50 ms

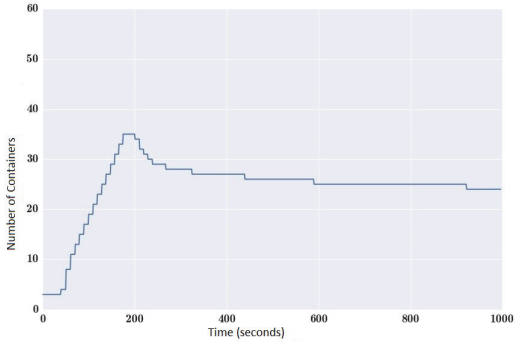


Figure 5. Number of Containers x Time (s), load_1, *setpoint* 50 ms

Figure 6 shows the average response time of the system while under load_1.5 and the setpoint kept at 50 ms. Figure 7 shows the containers allocation during this test. It is worth noting that the container allocation was adjusted to keep the average response time of the system near the setpoint. At the end of the run, 52 containers were allocated. In this case, the system under a greater load, allocated more containers to keep the response time within the defined threshold.

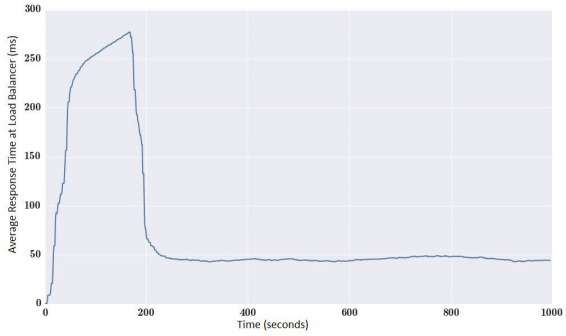


Figure 6. Response Time (ms) x Time (s), load_1.5, *setpoint* 50 ms

2) Scenario 2: In this scenario, the response time threshold in the load balancer (*setpoint*) was set at 50 ms and we applied load_1 for 1000 seconds, then load_1.5 for another 1000 seconds (starting at second 1001), and then we applied load_1 for more 1000 seconds (starting at second 2001). The

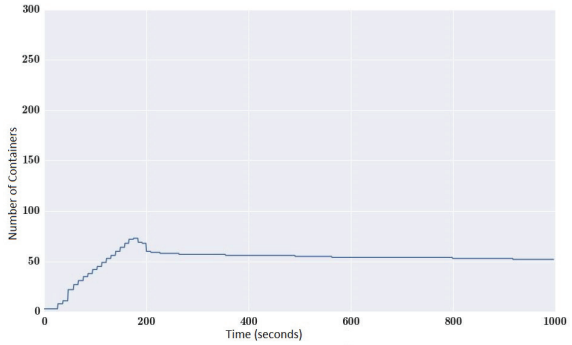


Figure 7. Number of Containers x Time (ms), load_1.5, *setpoint* 50 ms

purpose of this test was to evaluate the behavior of the system during sudden intensity changes.

As can be seen in figure 8 and 13, the system is capable to adjust itself increasing the number of containers when the intensity increases and decreases. It is observed that after exposing the system to load_1.5 at second 1000, the response time remains slightly over 50 ms, and after applying load_1 at second 2000 the response time remains slightly below 50 ms. Better results could be obtained for this case, readjusting the parameters of the PID controller.

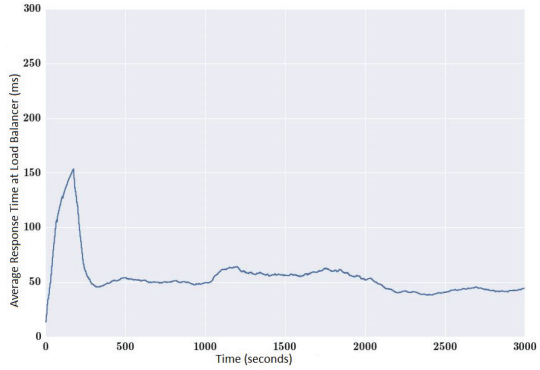


Figure 8. Response Time (ms) x Time (s), Variable load, *setpoint* 50 ms

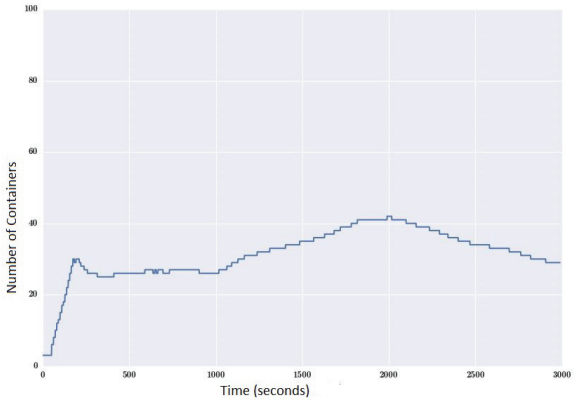


Figure 9. Response Time (ms) x Time (s), Variable load, *setpoint* 50 ms

3) Scenario 3: Scenario 3 compares the proposed algorithm in this paper with the HPA [7].

The comparison follows this procedure: the system configured with the HPA_80 (configured to scale when the average consumption of the containers of a given Replication Controller is above 80 %) is exposed to load_1, load_1.5 and load_2 during 1000 seconds. At the end of the test the average waiting time of the requests at the application layer (customer perspective) is observed.

From these data, it was defined a *setpoint* to use with the algorithm (PAS) for comparing with the response time close to the HPA reference. The results will be compared by checking the average amount of containers allocated during the experiments and the average response times achieved in the customer application layer.

As can be seen in Figure 10 the average response time in the application layer with the HPA algorithm for each load were: 66.18 ms for load_1, 110.12 ms for load_1.5 and 144.01 ms to load_2 .

For comparative purposes, the PAS *setpoints* was configured to deliver an average response time close to those obtained with the HPA. For this, we configured *setpoints* slightly below those observed in the HPA, as the *setpoint* is controlled in the load balancer, so the time measured in the customer application layer should be slightly higher. The values set for *setpoints* are: 50 ms (PAS_50) for the load_1 , 80 ms (PAS_80) for load_1.5 and 100 ms (PAS_100) for load_2 .

Figures 10 and 11 show that for response times near the value obtained by HPA, PAS system allocated less *containers* than HPA. The comparative of the *container* allocation shows that: PAS_50 allocated 44.02 % of which was allocated by the HPA_80, PAS_80 allocated 36.07 % of which was allocated by the HPA_80 to load_1 .5 and PAS_100 allocated 12.72% of which was allocated by the HPA_80 to load_2 .

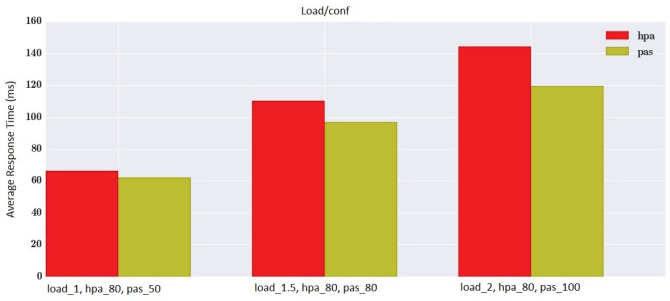


Figure 10. Comparison between HPA and PAS (Average Response Time of the System)

4) Scenario 4: In scenario 4 we evaluated the behavior of the PAS algorithm in the Rubis environment. We used loads_1.5. To generate request variability at every second the accesses to the links is divided as follows: 10 percent home page access, 10 percent of queries to the list of products with random category and random region, 40 percent of visits to random products and 40 percent of queries to random user profiles.

In this test the *setpoint* is set to 50 ms and it is applied load_1.5. Figure 13 shows the container allocation during the test, needed to control the application response time (*setpoint* = 50 ms) for load_1.5, and figure 12, shows the response

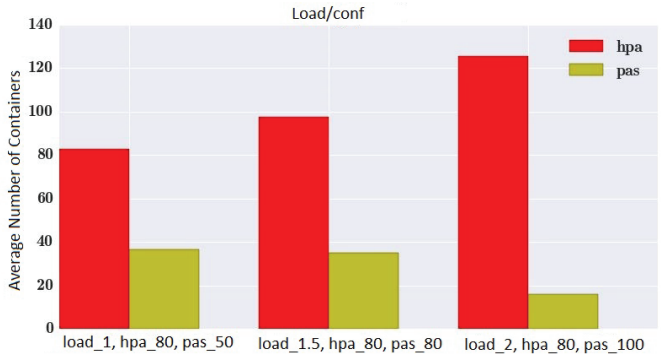


Figure 11. Comparison between HPA and PAS (Average number of containers)

time controlled near the *setpoint*. As can be seen, even with a much more varied workload than the array generator, PAS can control the average response time of the application close to the threshold .

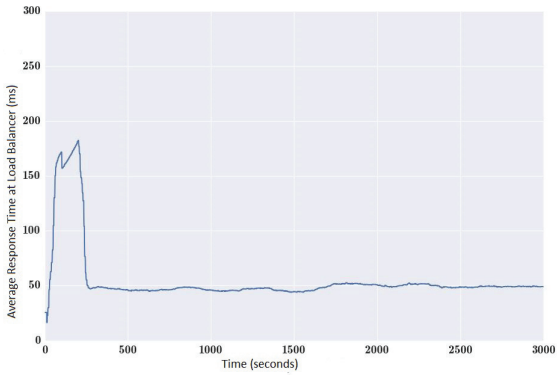


Figure 12. Rubis Response Time (ms) x Time (s), load_1.5, *setpoint* 50 ms

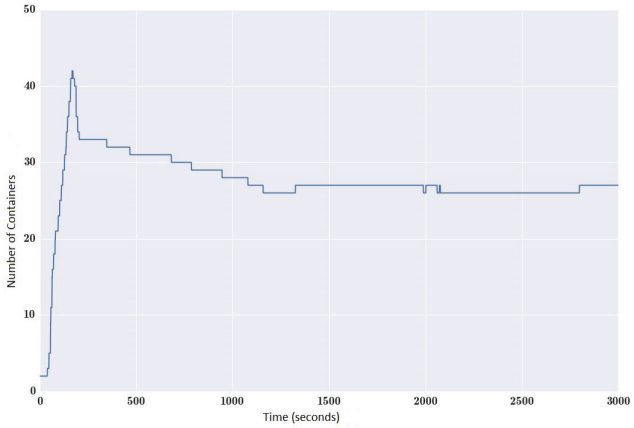


Figure 13. Rubis Number of Containers x Time (s), load_1.5, *setpoint* 50 ms

C. Analysis of Results

The experimental results show that for all the scenarios our proposal is efficient for resource allocation. In the scenario (V-B3), the PAS algorithm optimizes the allocation of the

number of containers to hold the values of *setpoints* within a given threshold. The HPA allocates a greater number of containers to achieve equivalent threshold response times for requests in the application layer. This result shows that the allocation of a larger number of containers can increase the complexity of load balancing time and not necessarily produce better response times.

The obtained results show that the PAS algorithm proposed in this work has the potential to promote an optimization of the number of allocated containers in a cloud computing environment.

The tested scenarios show that the PAS algorithm is a viable alternative to promote auto elasticity to comply with a required response time. Furthermore, performing measurements outside the container, allows PAS to be a generic tool for providing auto elasticity in cloud systems.

VI. Conclusion and Future Works

This paper presented an algorithm based on response time to scale containers on a Cloud Computing system. The proposal defines a cloud computing architecture based on containers and uses a PAS algorithm (PID based Autoscaler) to optimize resource allocation.

The proposal was evaluated in 4 scenarios using different workloads characterized from real world applications. The results shows that our proposal has the potential application to provide auto elasticity in cloud computing systems based on containers. The comparison with the native tool of Kubernetes, the HPA shows a higher efficiency for the PAS proposal.

In future work we intend to improve the PAS algorithm with sophisticated methods for setting the PID parameters. Furthermore the algorithm will be tested with other container orchestrators, such as Mesos and Docker Swarm, to verify that PAS can be a generic tool to provide auto elasticity in cloud computing environments based on containers.

References

[1] Mitchell Anicas. Mitchel Anicas an introduction to haproxy and load balancing concepts. https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts, 2014.

[2] Docker. Docker the definitive guide to docker containers. https://www.Docker.com/sites/default/files/WP-%20Definitive%20Guide%20To%20Containers.pdf, 2016.

[3] Flume. Flume flume user guide. https://flume.apache.org/FlumeUserGuide.html, 2016.

[4] Katja Gilly, Carlos Juiz, and Ramon Puigjaner. An up-to-date survey in web load balancing. *World Wide Web*, 14(2):105–131, 2011.

[5] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE, 2010.

[6] Google. Google container cluster manager from google. https://github.com/kubernetes/kubernetes, 2016.

[7] Google. Google horizontal pod autoscaler. https://github.com/kubernetes/kubernetes/blob/release-1.2/docs/design/horizontal-pod-autoscaler.md, 2016.

[8] National Instruments. National Instruments explicando a teoria pid. http://www.ni.com/white-paper/3782/pt/, 2015.

[9] Kairos. Kairos time series data storage in redis, mongo, sql and cassandra. https://pypi.python.org/pypi/kairos, 2015.

[10] Houssain Kettani, John Gubner, et al. A novel approach to the estimation of the hurst parameter in self-similar traffic. In *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on*, pages 160–165. IEEE, 2002.

[11] Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 12:2012, 2012.

[12] Nick Martin. Nick Martin a brief history of docker containers' overnight success. http://searchservervirtualization.techtarget.com/feature/A-brief-history-of-Docker-Containers-overnight-success, 2015.

[13] Rishabh Poddar, Anilkumar Vishnoi, and Vijay Mann. Haven: Holistic load balancing and auto scaling in the cloud. 2015.

[14] Redis. Redis redis documentation. http://redis.io/documentation, 2015.

[15] Rubis. Rubis rubis: Rice university bidding system. http://rubis.ow2.org/, 2009.

[16] Spark. Spark spark programming guide. https://spark.apache.org/docs/1.5.2/programming-guide.html, 2015.

[17] Willy Tarreau. HAProxy haproxy configuration manual. http://www.haproxy.org/download/1.5/doc/configuration.txt, 2015.

SLA and Profit-aware SaaS Provisioning through Proactive Renegotiation

Aya Omezzine[1,2,3] and Narjes Bellamine Ben Saoud[1]
[1]Univ. Manouba, ENSI, RIADI LR99ES26
Campus Universitaire Manouba, 2010, Tunisie
[2]Université Fédérale Toulouse Midi-Pyrénées, CNRS/LAAS
F-31400 Toulouse, France
Email: aya.omezzine@gmail.com, Narjes.bellamine@ensi.rnu.tn

Saïd Tazi[2,3] and Gene Cooperman[2,4]
[3]Univ. de Toulouse, UT1 Capitole, LAAS
F-31000 Toulouse, France
[4]College of Computer and Information Science
Northeastern University, Boston, MA / USA
Email: tazi@laas.fr, gene@ccs.neu.edu

***Abstract*—Software-as-a-Service (SaaS) providers offer on-demand, highly scalable applications to the end users. To maximize their profit, the providers must make profit-aware scheduling decisions about assigning client requests to virtual resources, while respecting the agreed upon Service-Level Agreement (SLA). Given the highly dynamic nature of the cloud environment, unexpected events may affect the initial scheduling plans, which leads to unanticipated SLA violations. Thus, an unaccounted event may create a lose-lose situation between provider and client. If the SLA is violated the provider must pay the potentially high penalty that is negotiated within the original SLA. But from the client's viewpoint, an SLA violation may cause cancellation of a business-critical job, and no ordinary SLA penalty can compensate for the loss of the client's business. The provider's reputation could also suffers as the number of such SLA violations grows, resulting in loss of future clients. On the contrary of most existing work that assume that once established the SLA cannot be modified, we propose to convert the lose-lose situation into a win-win one through an automated renegotiation mechanism. When an event threatens a lose-lose violation of the SLA, the renegotiation mechanism is launched to establish a new SLA that limits the losses on the two sides. Experiments show that this new approach minimizes the loss in profit of the provider and minimizes the number of cancelled jobs experienced by the client, as compared with enforcing the original SLA.**

***Index Terms*—Cloud computing; SaaS provisioning; Service Level Agreement (SLA); SLA-aware scheduling; Client satisfaction; Automated renegotiation; Decision making strategy;**

I. Introduction

SaaS providers offer highly scalable applications to end users over the Internet. To run their applications, SaaS providers often prefer to rent virtual resources from an Infrastructure-as-a-Service (IaaS) provider instead of in-house hosting. By doing so, they avoid infrastructure maintenance and they can scale their application to serve as many end users as possible. Thus, the end user negotiates with a SaaS provider, while that provider in turn schedules jobs with IaaS providers.

The SaaS application provisioning must satisfy the SLA contract established between the two parties. The SLA contract is a formal representation of the QoS parameters, obligations of the two parties, and provider penalties, that are agreed upon. In order to maximize their profit and to satisfy end users, the SaaS providers use an SLA-aware scheduling algorithm, which efficiently assigns user requests to virtual resources offered by IaaS providers.

978-1-5090-3216-7/16/$31.00

Cloud computing represents a highly dynamic environment (both at the business level and at the resource level). There may be unforeseen events at the resource level such as catastrophic resource failure, or else there may be unexpected events at the business level coming from the need to share rented resources between new clients that compete for immediate execution. These events may result in violation of the original negotiated SLA, since the schedule originally done (based on the initial SLA) can be modified.

Generally if a contract is violated, a penalty is paid and the service is canceled [1], [2]. But if a contract is violated due to circumstances not accounted for in the original SLA negotiation, the two parties may *both* lose badly. For example, consider the situation in which a job is critical to the success of the business. In principle, the client could have insisted on a penalty in the original SLA that is equal to the value of client's business, as compensation for the losses due to the failure of that business-critical job. But this is usually unrealistic, since such a penalty can be larger even than the total assets of the provider. Hence, the client will never be fully compensated, and the provider faces a loss of future clients due to the loss in reputation as the number of violated jobs accumulates.

For these reasons, the provider and the client would normally prefer to renegotiate using a new SLA with a new deadline (i.e., an extension beyond the first deadline), rather than pay a steep penalty and accept the cancellation of a business-critical job. The new SLA will generally include a discount by the provider on the originally agreed-upon price, as a concession by the provider for avoiding the steep penalty envisaged by the original SLA violation.

Most of the literature assumes that once an SLA is established, it cannot be renegotiated [1], [2], [3], [4]. The concept of *renegotiation* has not yet been well studied [5]. There is some work that tries to enhance the WS-Agreement negotiation protocol using renegotiation [6], [7], [8], and others propose general conceptual renegotiation frameworks [9], [10]. However, the term *renegotiation* in this prior literature always refers to a renegotiation phase within the *original SLA*.

In particular, the prior work mentioned above does not propose any decision-making model to guides the renegotiation process toward a satisfactory agreement. In contrast, the current work proposes decision-making strategies in which the negotiators renegotiate based on a concession (lowering) of the original penalty.

The *key novelty of this paper* is to propose an automated renegotiation-based approach when detecting an unexpected event during the SaaS provisioning process. In our approach, the provider proactively renegotiates with the clients whose jobs may be in violation of the SLAs, in order to minimize the loss in profit and in order to assure the continuity of the service. The renegotiation approach is composed of two steps.

1) The first step happens when the SaaS provider detects an unexpected event. Since the provider may not be able to physically continue some jobs with the same scheduling parameters (VM, completion time, etc.), we consider alternative rescheduling options for the provider. The first step consists of the selection of an option for profit-aware rescheduling. In examining the possible scheduling options, the provider chooses an option leading to a minimum loss in profit while also minimizing the number of canceled jobs. (See Section III-C.)
2) At the second step, the SaaS provider triggers a renegotiation with those end users whose jobs may terminate after deadline. The renegotiation consists of an exchange of offers and counter-offers guided by decision-making strategies using a utility model that is based on economics. The strategies followed by the SaaS provider are based on the rescheduling option selected in the first step, above. (See Section IV-A and following.)

The rest of the paper is organized as follows. Section II presents the basis for negotiation, how it is used in cloud computing, and especially the role of renegotiation. Section III and Section IV describe the first step and second step, respectively, of the renegotiation-based SaaS provisioning process. Section V presents experiments to assess the algorithm. Section VI is dedicated to discussing the related work. Conclusions are presented in Section VII.

II. Motivation and background

Cloud computing presents a highly dynamic marketplace for delivering IT services on demand. Each cloud actor has its own interests. In particular, the client aims to obtain the most convenient service at the cheapest price, while the provider aims to maximize its profit and to serve the maximum number of clients. The negotiation between cloud actors is an intuitive way to solve conflicts between client and provider and to reach a satisfactory agreement. As the infrastructure and platforms for the cloud become more complex, we need more automated negotiation to handle that interaction.

Automated negotiation can be split into three issues [11]. 1) The *negotiation protocol* expresses the locutions that may be exchanged between the negotiators and defines the rules of interaction. 2) The *negotiated service* is composed of objects (also called issues) about which the participants negotiate. There are specific service issues, which concern the service type, and there are generic issues such as the price. 3) The *decision-making model for negotiation* defines the decision-making strategies for each actor.

Decision making is composed of two main strategies. The first strategy allows one to evaluate a received offer and to decide whether to accept, reject or propose a counter-offer. This first strategy is generally based on a utility function that measures the degree of satisfaction of a received offer according to the preferences. The second strategy enables one to generate offers or counter-offers at each step.

Automated negotiation in the Cloud is primarily used to establish an SLA between clients and providers. It happens generally in the first phase of the service-provisioning process (before the SLA establishment itself). In this paper, we focus on SaaS application provisioning and especially on compute-intensive applications. Some examples are: scientific data processing, and finance data analysis. In the first phase, in order to maximize the number of clients and minimize the costs of renting sufficient computer resources, the SaaS provider adopts a profit- and SLA-aware scheduling algorithm. The schedule must guarantee that the SLA is met, while also maximizing the profit.

After signing the contract, *unexpected events* may later occur that can impact the current scheduling. To avoid an SLA violation, the provider must take rescheduling actions. However, maybe no rescheduling can meet the previously signed SLA. For example, migrating a job to another VM after failure may delay the completion time beyond the agreed upon deadline. The SLA model generally assumes that once the deadline is violated, the job is automatically cancelled and a penalty is paid.

This is the scenario in which automated renegotiation becomes important, as part of a second phase of service provisioning. This second phase occurs after violation of the original SLA due to an unexpected event. We invoke a new renegotiation phase between the end user and the SaaS provider, in order to minimize the SLA penalty costs and in order to ensure the continuity of service on which the provider's reputation depends.

In what follows, we detail how automated renegotiation can be used to handle unexpected events, as part of a second phase in SaaS application provisioning.

III. Selection of an Option for Profit-aware Rescheduling

When detecting an expected event that alters the initial scheduling, the provider takes rescheduling actions in order to avoid SLA violations to the extent possible. Generally, the provider may have more than one rescheduling option. For this reason, we propose an algorithm for the selection of an option for profit-aware rescheduling. In this section, we model first the unexpected event and the rescheduling option. Then, we present our algorithm for selection of a rescheduling option.

A. Definition of an Unexpected Event

An unexpected event leads to a change in the situation under which the already signed SLAs had originally been negotiated. Indeed, the schedule contracted by the SaaS provider in the first phase may be affected, thereby leading to violation of the original SLA. The unexpected events can be classified into two categories: 1) *resource events*, for example VM failure, failure of the currently executing job, etc. The jobs scheduled on that VM may be affected, and so the initial scheduling may be altered. 2) *business events*, such as a new incoming client needing immediate execution with no additional VMs available from the IaaS provider. Thus, the SaaS provider may choose to execute a new job on an already active VM even though there exist prior jobs (either running, or scheduled but not yet started). An unexpected event can be specified using two parameters: the time at which the event occurs, t_{event}; and the set of resources affected by the event, vm_{ID}. The unexpected event is assumed to be detected just prior to SLA violation through a monitoring module.

B. Definition of the Rescheduling Option

The rescheduling option is composed of potential scheduling actions applied to accepted and scheduled jobs (which may either be running or not yet started). Two examples of rescheduling actions are: (i) the provider may proactively migrate the job to a different computer; and (ii) the provider may invoke periodic checkpointing to protect against catastrophic failure. (The provider can then restart from a previous checkpoint image on a new computer, or even directly migrate to a new computer.) A scheduling action defines when and where to place a job. A scheduling action Ac, applied to a job j, can be defined as $(type, j, estimated\_start\_time, vm_{ID}, compT)$ where: the $type$ denotes the scheduling action type. For example: insert, postpone, cancel/restart, migrate,suspend/restart. The $estimated\_start\_time$ and vm_{ID} define when and where to start the job, respectively. The $compT$ denotes the estimated completion time, and can be calculated based on the information given by scheduling action.

Hence, a rescheduling option, denoted Op, is defined as follows: $Op = \{Ac_j\}$, where $j \in \{\text{rescheduled\_jobs}\}$. Each rescheduling option has as output a list of rescheduled jobs ($resch\_List$) and the list's rescheduling information given by the scheduling action.

C. Algorithm for Selection of a Profit-aware Rescheduling Option

For the selection of a rescheduling option, we consider two metrics: 1) the $lossInProfit_p$, which calculates the SaaS provider loss in profit when choosing rescheduling option p; and 2) the number of potential cancelled jobs when choosing option p, denoted by $nbrJobs_p$.

1) *The $lossInProfit_p$ for the provider:* This include two parameters: a) The $actionCost$ define the cost due to the action. For example, if the action is to migrate the job to a new VM, the cost of the action will be equal to the price of provisioning a new VM. b) The $penaltyCost$, defined as the SLA violation cost, which can be calculated for a job j using the following formula:

$$penaltyCost_j = \begin{cases} 0, \text{if } compT_j \leq respT_j \\ pr_j * delay_j, \text{if } respT_j \leq compT_j \leq dl_j \\ fixedPenalty_j, \text{if } dl_j < compT_j \end{cases} \quad (1)$$

where $respT_j$ is the agreed upon *response time*, and dl_j is the agreed upon *deadline*. The value pr_j indicates the *penalty rate*. The $fixedPenalty_j$ denotes the penalty paid in case of violation.

2) *The number of potential cancelled jobs, $nbrJobs_p$:* This calculates the jobs whose estimated completion time $compT_j$ are greater than the deadline dl_j of the initial SLA.

The proposed algorithm, Algorithm 1, below, takes as input the list of possible rescheduling options that the provider can choose after detecting the unexpected event. The algorithm returns a scheduling option and associated rescheduling information for each job such that the number of cancelled jobs is minimized and the loss in profit is minimized.

For each possible rescheduling option: First, Algorithm 1, below, calculates the estimated completion time $compT_j$ for each job j, based on the action Ac_j applied to this job (line 5). Second, the algorithm calculates the $lossInProfit$ as a sum of the loss in profit for each of the rescheduled jobs (lines 6 and 7). Third, the algorithm selects the potential cancelled jobs whose $compT$ are greater than the agreed upon deadline, calculates $nbrJobs$, and stores the rescheduling information for those jobs in $potentialCancelJobs$ (line 8 to 10). Then, the algorithm stores the option results in $optionsResults$ (line 13). Finally, the algorithm selects the option noted $optionsResults_s$ having the minimum $nbrJobs$ and the minimum $lossInProfit$ (line 14) using the $selectBestOption$ function. We propose to use utility functions in order to select the most convenient option. The utility function of an attribute i with value x can be calculated as follows.

$$\text{Let } U(x_i) = \frac{x_i - x_{worst}}{x_{best} - x_{worst}}, \quad (2)$$

where x_{worst} and x_{best} denote the best and worst value, respectively. **For each possible rescheduling option**: the $selectBestOption$ function calculates the utility values $UtLoss$ and $UtNbrJobs$ for $lossInProfit$ and $nbrJobs$ using equation 2 (line 21 and 22). The worst and best values for the $lossInProfit$ are the maximum and minimum $lossInProfit$ values selected from the $optionResults$, respectively. Likewise, the worst and best values for $nbrJobs$ are the maximum and minimum $nbrJobs$ values selected from the $optionResults$, respectively. Then, the function calculates the option's distance to the best option, which has $UtLoss = 1$ and $UtNbrJobs = 1$ (line 23). Finally, the function returns the option having the minimum distance to the best option (line 24 to 27). The provider will renegotiate based on the results of the rescheduling option selected ($optionsResults_s$)

Algorithm 1 Pseudo-code for selection of rescheduling option

Input: The list of possible rescheduling options
Output: The rescheduling option leading to the min $lossInProfit$ and min $nbrJobs$

1: **for** each $p \in$ possible rescheduling options **do**
2: $lossInProfit_p = 0$
3: $nbrJobs_p = 0$
4: **for** each $j \in resch\_List_p$ **do**
5: $compT_j = getCompT(Ac_j)$
6: $lossInProfit_j = penaltyCost(compT_j) + actionCost(Ac_j)$
7: $lossInProfit_p = lossInProfit_p + lossInProfit_j$
8: **if** $compT_j > dl_j$ **then**
9: $nbrJobs_p++$
10: Add resch info from $resch\_List_j$ to $potentialCancelJobs_p$
11: **else**
12: **continue**
13: Store $lossInProfit_p$,$nbrJobs_p$,$potentialCancelJobs_p$ in $OptionsResults_p$
14: $OptionsResults_s$=selectBestOption($OptionsResults$)
15: **return** $OptionsResults_s$

16:
17: **Function** selectBestOption($OptionsResults$)
18: $minDistance = \sqrt{2}$
19: $optionsResults_s = optionsResults_p$
20: **for** each $p \in OptionsResults$ **do**
21: $UtLoss_p = U(lossInProfit_p)$
22: $UtNbr_p = U(nbrJobs_p)$
23: $Distance_p = \sqrt{(UtLoss_p - 1)^2 + (UtNbr_p - 1)^2}$
24: **if** $Distance_p < minDistance$ **then**
25: $minDistance = Distance_p$
26: $optionsResults_s = optionsResults_p$
27: **return** $optionsResults_s$

IV. The Renegotiation-based Rescheduling Procedure

Once a rescheduling option is selected, the provider will renegotiate with the clients whose jobs may be cancelled by triggering a *renegotiation session* with each client. The values of the renegotiable issues (deadline, compensation) will be guided by the renegotiation decision-making strategy and will be based on the results of the selected rescheduling option. In this section, we present first the overall process for renegotiation. Then we present details about the strategies that will be followed by the provider and the client.

A. The renegotiation overall process

A renegotiation session can be defined as the period covering the time when the interaction between negotiators begins until it stops. The renegotiation session terminates either with an agreement, and in this case the new SLA is applied, or without an agreement, in which case the initial SLA is applied. The different states of the renegotiation session, denoted $renegSessionState$, are: 1) $Active$ (when the two parties are exchanging offers and counter-offers); 2) $Succeeded$ (when the renegotiation session terminates with an agreement if one party accepts the offer received from his opponent); 3) $Failed$ (when the renegotiation session terminates without an agreement). This last situation ($Failed$) occurs when one party rejects the opponent's offer or when the negotiation deadline is reached.

The renegotiation-based rescheduling algorithm, Algorithm 2, takes as input the $potentialCancelJobs$ list (included in the $optionsResults$ returned by Algorithm 1). For each job that may be cancelled, the provider opens a renegotiation session with the client that owns that job. The renegotiation sessions are triggered sequentially. The provider opens a new renegotiation session only if the current one is terminated (lines 3 and 4). If the renegotiation terminates with success, then the SLA is updated to include the new agreed upon deadline and the compensation (lines 5 and 6). If the renegotiation about the job j fails then the provider must update the estimated completion time of the jobs that potentially are rescheduled after job j, in order to avoid the resource wastage due to unused time slots (lines 8 to 10). For that reason, the renegotiation is done sequentially, so that the provider can update the estimated completion time of the rescheduled jobs based on the renegotiation session's output.

Algorithm 2 Pseudo-code for renegotiation-based rescheduling

Input: The list of potential cancelled jobs
Output: The results of each renegotiation session

1: **for** each $j \in potentialCancelJobs$ **do**
2: open renegotiation session j with owner of job j
3: **while** $renegSessionState_j == Active$ **do**
4: **wait**
5: **if** $renegSessionState_j == Succeded$ **then**
6: update the SLA_j
7: **continue**
8: **else if** $renegSessionState_j == Failed$ **then**
9: **for** each $k \in$ rescheduled_jobs after j **do**
10: update $compT_k$ in $potentialCancelJobs_k$

B. The Decision-making Strategies for Renegotiation

During the renegotiation session, the provider and client automatically exchange offers and counter-offers according to their decision-making strategies. The renegotiation strategy should be designed to rapidly achieve agreement, since the participants are generally pressed when renegotiating after an SLA violation. For this reason, we assume that the new deadline proposed by the provider in the first round cannot be modified when exchanging offers and counter-offers. This is because the proposed deadline value is imposed by the rescheduling option selected. So the given deadline value is the best that the provider can offer to the client.

In what follows, we present how the compensation value is evaluated and generated during the renegotiation session.

1) Decision-making by the Provider:

The offer evaluation strategy: The offer evaluation is based on the satisfaction model described in [12]. The utility value of a negotiable attribute i with value x can be calculated using equation 2 where the worst and best values are defined by the negotiator before starting the negotiation as internal preference. In our scenario, based on the SLA model described in equation 1, the values $penaltyCost(deadline)$ and $fixedPenalty$ denote the best and worst values of compensation, respectively.

The acceptance conditions of a received offer from the client during the renegotiation session are: 1) $U(compensation\_received) >= U(compensation\_proposed)$; and 2) $deadline\_received > compT$.

If the offer received from the client does not satisfy the two conditions mentioned above, the provider will propose a counter-offer using the *utility-based offer generation* strategy.

The Strategy for Generation of Utility-based Offers: As mentioned earlier, the proposed new deadline will be equal to the estimated completion time included in $potentialCancelJobs$ list. Given the expected compensation utility for the provider, the compensation value can be generated using equation 2 of Section III-C. The *expected utility* consists of a tradeOff between minimizing the loss in profit and satisfying the client. The expected client utility can vary between 0 and 1. In the special case when the utility is equal to 1, the provider proposes a minimum compensation (the provider's best value) while still managing to relax the deadline. So, in this case the provider prefers minimizing the provider loss over satisfying the client.

And in the special case that the utility is equal to 0, the provider proposes to pay the fixed penalty as compensation while continuing to run the job. So, the provider doesn't minimize the provider loss, but instead satisfies the client by not cancelling his job. The SaaS provider can offer this to the client only because it had obtained additional resources during the rescheduling phase with the IaaS provider, as described in Section III-C. Before renegotiating and according to the provider's internal preferences, the provider has fixed values for *preferred* and *reserved* utility values (upper and lower bounds on the expected utility). Those values are kept secret and are not know by the client. In the first round, the provider generates the initial offer based on the provider's preferred utility. During the later rounds, the provider may back off from its preferred utility until reaching its reserved utility.

2) Decision-making by the Client:

The Strategy for Offer Evaluation: The evaluation is based on the *overall utility* value. The overall utility of a received offer composed of n attributes is calculated as a weighted sum of each single utility using the following equation:

$$U(offer) = \sum_{i=1}^{n} w_i * U(x_i) \quad (3)$$

where w_i is the weight expressing the importance of the attribute i and w_i is in the range $[0, 1]$. For example, for high priority jobs, users may place more importance on the deadline than on compensation. In contrast, for low priority jobs, users may place more importance on the compensation than on the deadline.

The client defines preferred ($preferredUt$) and reserved ($reservedUt$) utility values, as bounds on the overall expected utility. Those clients having urgent business-critical jobs assign low value to the ($reservedUt$). This is because they prefer to accept the job along with a relaxed deadline and a smaller compensation, rather than having the job cancelled. The client preferences are kept secret.

In our scenario, the client accepts an offer only if $U(offer_{received}) \geq reservedUt$. The client rejects an offer if $\exists\, issue\, i, U(x_i) < 0$. Otherwise, the client proposes a counter-offer using the following strategy.

The Strategy for Offer Generation: As mentioned earlier, the client does not change the deadline value proposed by the provider when generating a counter-offer. Since the deadline utility is known (expressed by the provider's initial offer), the compensation utility value can be generated from the expected overall utility using equation 3. As was the case for the provider, the client similarly starts by generating an offer according to the $preferredUt$ value, until reaching the $reservedUt$ value.

V. Evaluation and analysis

A. Experimental settings

To simulate the cloud market and the interaction between the SaaS provider and the final users, we implement a multi-agent system using JAVA and the Java Agent DEvelopment framework (JADE) [13]. Each software agent is acting on behalf of either clients or providers. The agents negotiate through the FIPA iterated contract net protocol, which is a multi-round negotiation protocol [14].

In the SaaS application provisioning process there are two phases: 1) Before the SLA establishment: the provider tries to find a schedule satisfying the client request, and decide whether to accept or reject the request. Once accepted, an SLA is signed between the two parties (between SaaS provider and client). 2) When an unexpected event occurs after the SLA establishment, as we have presented in Section III, there are two steps. The first step happens when the provider initially detects an unexpected event that may alter the initial schedule. The first step deals with choosing an option from several possible rescheduling options. In the second step, the provider triggers a renegotiation session with each user whose SLA may be violated.

Since we are interested in testing and validating the renegotiation approach, we assume in our experiments that:

- The first phase is done according to an existing SLA-aware scheduling algorithm [3]. That algorithm performs more efficiently when evaluated and compared with the reference scheduling algorithms [3]. For each accepted job, the output of the first phase is an SLA with the

required scheduling information. The scheduling information indicates where and when to put the job to satisfy the SLA.

- The first step of the second phase is not explicitly implemented. Instead, we generate an unexpected random event. We implement a rescheduling module simulator that generates the list of potential rescheduled jobs and their estimated completion time given an unexpected event. We assume that the jobs are rescheduled sequentially. The estimated completion for the job running can be generated randomly and for the other jobs using the following formula.

$$compT_j = compT_{jr} + \sum_{k \in \{\text{k between jr et j}\}} procT_{k,l} \quad (4)$$

where $compT_{jr}$ denotes the completion time of the job running jr at t_{event}. And $procT_{k,l}$ denotes the processing time of job k on the VM of type l.
We assume that the rescheduling module simulator chooses the best rescheduling option.

B. Results and Analysis

Our objective is to evaluate the renegotiation-based application provisioning algorithm and to compare it to the basic scenario in which the provider cannot modify the established SLA. For the basic scenario, we assume that the provider tries to execute a rescheduling action (step 1) without any renegotiation. If the SLA is violated the job is cancelled and the SLA penalty is paid. We measure performance using two metrics: 1) the total loss in profit, expressing how much the provider loses when violating an already established SLAs; and 2) the number of cancelled jobs, the number of jobs whose completion time is beyond the agreed upon deadline for the original SLA.

We conduct three types of experiments in which we calculate the loss in profit and the number of cancelled jobs. For these experiments, we assume that each agent (provider or consumer) is able to generate only one offer during the renegotiation session, since the renegotiation must be done in a timely manner. Furthermore, we assume for the expected utility that the agent's reserved utility is equal to the preferred one. So the agents generate one offer according to their expected utility. If the opponent accepts the offer, the renegotiation ends with an agreement. Otherwise the renegotiation fails. This configuration (where preferred utility is equal to the reserved one) is the worst possible configuration in negotiation, since it is the least flexible. By choosing this configuration, we will be sure that for other configurations, our renegotiation algorithm will perform better. Hence, when relaxing the expected utility, there is a greater chance of a request/offer being accepted, and so the number of successful renegotiation sessions will be increased.

For the first and the second experiments, we vary the expected utility for the provider and the client, respectively, while injecting exactly one unexpected event (affecting only one VM). For the third experiment, we vary the number of resources affected by the unexpected event. Note that an event may lead to altering the initial scheduling of more than one VM. For example, a failure may affect many VMs.

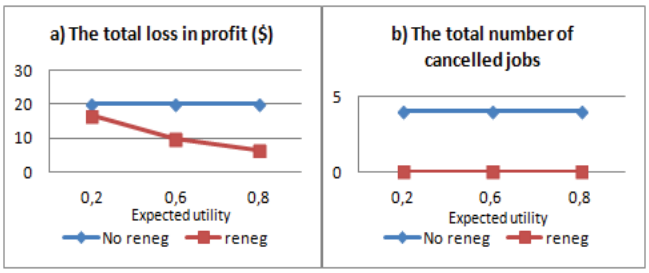


Fig. 1. Impact of provider's expected utility variation

1) Impact when varying the expected utility of the SaaS provider: Figure 1 shows the different values obtained for the loss in profit and the number of cancelled jobs with respect to the provider expected utility. For those experiments, we generate clients and their initial request with expected utility equal to 0.1 (clients with business-critical jobs). We observe that the loss in profit and the number of cancelled jobs using renegotiation is minimized compared to the basic scenario. Without renegotiation, the loss in profit and the number of cancelled jobs are constant. regardless of the value of the provider utility. This is expected, since the provider's strategy for handling unexpected events does not consider the value of the provider utility.

In Figure 1(a), the loss in profit (red curve) is decreasing when the provider's expected utility increase. This is because the utility is related to the compensation paid to the client. The higher the utility, the less is the compensation that is paid, and so the loss in profit is also less. In Figure 1(b), the number of cancelled jobs (red curve) is constant regardless of the value of the provider utility, this is because the client's reserved utility is at the lower limit. This implies that the client will accept any offer from the provider, even if the compensation is not at the upper limit (not at the upper bound for the provider utility). For those clients, a lower utility is nevertheless better than cancelling the job.

In the next experiments, we will vary the clients' expected utility.

2) Impact when varying the expected utility of the clients: Figure 2 shows the different values obtained for the loss in profit and the number of cancelled jobs, with respect to the clients' expected utility. For those experiments, the provider's expected utility is equal to 0.6. We note, as in Figure 1, that the loss in profit and the number of cancelled jobs are constant in the basic scenario, since the basic scenario does not take into account client satisfaction.

With renegotiation, we notice that the loss in profit and the number of cancelled jobs increase when the client expected utility increases. For the users with low utility values, the renegotiation algorithm performs much better than the basic one. But, for users with high utility values, the renegotiation algorithm results are the same as the basic one. So, when

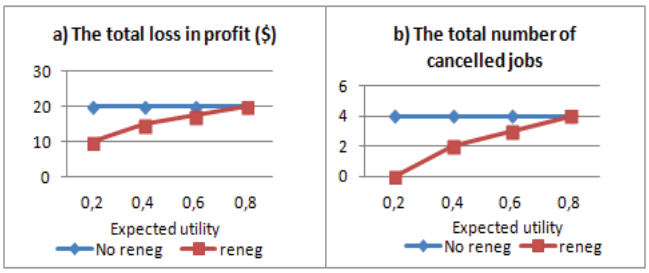


Fig. 2. Impact of variation of clients' expected utility

increasing the clients' expected utility, the renegotiation algorithm performance tends to the performance of the basic algorithm. In contrast, when the expected utility is low, the client has a high-priority business-critical job, and so it accepts any renegotiation offer in order to assure the continuity of its business. In contrast, the client with a high expected utility (i.e., having a less business-critical job) may choose to not accept a renegotiation offer. In this case, the client prefers that the provider should pay the penalty and cancel the job.

3) Impact as the number of resources are varied: Figure 3 shows the different values obtained for the loss in profit and the number of cancelled jobs with respect to the number of affected resources. For those experiments, the provider and the client expected utility are equal to 0.6 and 0.1, respectively.

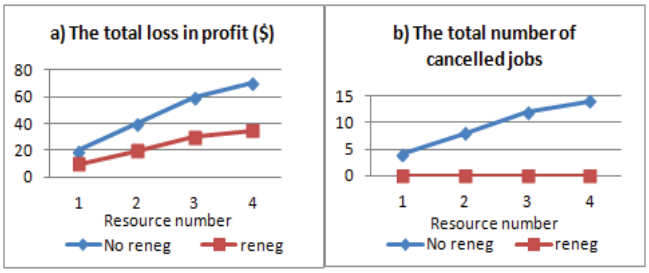


Fig. 3. Impact of variation of number of resources

We notice that the loss in profit (with and without renegotiation) and the number of cancelled jobs (without renegotiation) increase when the number of affected resources increases. Further, when the unexpected event affects many VMs, the number of rescheduled jobs increases which lead to a potentially increased number of cancelled jobs. Consequently, the total loss in profit will increase. In Figure 3(b), the number of cancelled jobs is equal to zero, regardless of the number of resources. This is because, in our configuration, we generate clients whose jobs are highly business-critical. So the clients always accept the renegotiation requests.

For the three experiments, we conclude that: 1) our algorithm's performance exceeds that of the basic algorithm in terms of profit and the number of cancelled jobs when the clients' jobs are highly business-critical (low expected client utility); and 2) our algorithm's performance tends toward the basic algorithm's performance when the clients have jobs that are less business-critical (when the clients' expected utility is high). Thus in the second case, the clients do not accept a renegotiation, and prefer to enforce the initial SLA.

VI. Related Work

Our work is related to SLA-aware Cloud service provisioning. Most of the existing work proposes an approach aiming to guarantee the agreed upon QoS during the service provisioning process. However, there is less work that considers the consequences of SLA violations, and how the service provisioning should be affected by those violations (e.g., the effect on the provider profit and provider reputation).

In [15], Wu et al. propose a negotiation framework that helps both consumers and providers to define QoS parameters values before service provisioning. The proposed framework includes brokers that assist consumers to find SaaS providers satisfying their needs. The provider cost model does not consider the SLA penalties to be assessed in case of violation.

In order to avoid SLA violations and minimize SLA penalties, it is important to design efficient SaaS scheduling algorithms [1], [2], [3], [4]. In [2], Leitner et al. propose a scheduling algorithm that takes as input the incoming job's execution time requests and the current resource load. That algorithm decides for each request whether to launch a new VM or to schedule it on an existing VM. The objective is to minimize the cost of running VMs and to minimize SLA violations. Despite the fact that the provider revenue depends on the budget given by the requests, the authors do not consider this parameter in the scheduling decision. In the same sense as [2], Liu et al. [1] propose a genetic algorithm that aims to maximize revenue by minimizing the costs of rented VMs. This algorithm divides the user's request into sub-tasks, and then tries to find the optimal combination of VMs able to run those sub-tasks without an SLA violation. Although [1], [2] do not consider the client's budget when scheduling, Wu et al. [3] propose admission control strategies that take into account the budget and the deadline to decide whether to accept or reject the client's request. The main goal is to avoid SLA violation and maximize profit. In [4], Wu et al. propose a scheduling algorithm for enterprise-based SaaS application. The algorithm aims not only to minimize the number of rented VMs, but also to maximize the Customer Satisfaction Level (CSL) by considering the consumer's future interest when scheduling his or her initial request.

The works cited above do not consider what to do after an SLA violation, and once established, the SLA cannot be modified. Our work proposes a renegotiation-based approach to handle possible SLA violations. In contrast to cloud service negotiation for SLA establishment, which is well developed, the subject of renegotiation has not yet been well studied. We are interested in work dealing with renegotiating an already signed SLA, in contrast to [16], which considers renegotiation as negotiating a counter-offer (before the SLA establishment). There is some research work that evokes the idea of SLA renegotiation, but without presenting a concrete contribution on how it could be done [5], [17]. That work focusses on showing the importance of adding renegotiation to the

SLA management life-cycle and presents the requirements for doing so. In [9], [10], the authors propose a conceptual framework for renegotiation. Before renegotiation was introduced to the WS-Agreement protocol by the Grid Resource Allocation Agreement Protocol (GRAAP) group [18], many researchers tried to extend the negotiation component of the WS-Agreement standard in order to support renegotiation [6], [7]. Those authors focus on a renegotiation protocol and propose an approach for extending the WS-Agreement standard in order to support renegotiation.

None of the above-mentioned work proposes a decision-making approach for renegotiation. In [8], Sharaf et al. propose a decision-making strategy based on a fuzzy logic decision support system, as part of the AssessGrid project. The proposed strategy enables the evaluation of an offer received during renegotiation. The authors do not provide details on how the offers are generated during the renegotiation.

To the best of our knowledge, no previous decision-making approach for renegotiation handles a SaaS provisioning procedure wherein the SaaS provider is provisioned by a lower-tier IaaS provider. Our work differs from the work above in that we propose a renegotiation process based on SaaS scheduling information. The proposed renegotiation approach aims not only to minimize the loss in profit due to violation, but also to assure the continuity of service.

VII. Conclusion

For scaling purposes, SaaS providers need to rent resources from IaaS providers in order to run their highly scalable applications. In order to maximize their profit and to satisfy clients, a SaaS provider employs an SLA-aware scheduling algorithm that efficiently assigns client requests to rented resources. Since the cloud environment is highly dynamic, unexpected events may occur that alter the originally selected schedule and lead to SLA violations. Most of the literature assumes that once established, an SLA cannot be modified, and when violated the job is automatically cancelled, without first allowing the provider and consumer the option of renegotiation. The provider pays a high penalty and loses reputation, while the consumer may have a business-critical job cancelled.

We have described an SLA renegotiation-based approach to proactively handle such SLA violations. The resulting decision-making model makes possible a win-win situation (ensuring continuity of service and minimizing SLA penalties costs). The decision-making strategies are based on a utility function for the provider and scheduling information generated by the rescheduling option chosen before renegotiation.

In the future, we plan to investigate further the rescheduling options upon detecting an expected event. The impact of these options will be studied in a large-scale environment using a real world application. Finally, we intend that negotiators will be able to automatically choose an appropriate decision-making negotiation strategy based on the situation.

Acknowledgment

This publication is partially supported by the IDEX "Chaire d'attractivité" program of the Université Fédérale Toulouse Midi-Pyrénées under Grant 2014-345.

References

[1] Z. Liu, S. Wang, Q. Sun, H. Zou, and F. Yang, "Cost-aware cloud service request scheduling for saas providers," *The Computer Journal*, p. bxt009, 2013.

[2] P. Leitner, W. Hummer, B. Satzger, C. Inzinger, and S. Dustdar, "Cost-efficient and application sla-aware client side request scheduling in an infrastructure-as-a-service cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 213–220.

[3] L. Wu, S. K. Garg, and R. Buyya, "Sla-based admission control for a software-as-a-service provider in cloud computing environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1280–1299, 2012.

[4] L. Wu, S. K. Garg, S. Versteeg, and R. Buyya, "Sla-based resource provisioning for hosted software-as-a-service applications in cloud computing environments," *IEEE Transactions on services computing*, vol. 7, no. 3, pp. 465–485, 2014.

[5] A. F. M. Hani, I. V. Paputungan, and M. F. Hassan, "Renegotiation in service level agreement management for a cloud-based system," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 51, 2015.

[6] M. Parkin, P. Hasselmeyer, and B. Koller, "An sla re-negotiation protocol," in *Proceedings of the 2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC08), CEUR Workshop Proceedings, ISSN 1613-0073, Volume 411*. Citeseer, 2008.

[7] G. Di Modica, O. Tomarchio, and L. Vita, "Dynamic slas management in service oriented environments," *Journal of Systems and Software*, vol. 82, no. 5, pp. 759–771, 2009.

[8] S. Sharaf and K. Djemame, "Extending ws-agreement to support renegotiation of dynamic grid slas," in *eChallenges e-2010 Conference*. IEEE, 2010, pp. 1–8.

[9] A. F. M. Hani, I. V. Paputungan, and M. F. Hassan, "Service level agreement renegotiation framework for trusted cloud-based system," in *Future Information Technology*. Springer, 2014, pp. 55–61.

[10] W. Mach and E. Schikuta, "A generic negotiation and re-negotiation framework for consumer-provider contracting of web services," in *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2012, pp. 348–351.

[11] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, M. Wooldridge, and C. Sierra, "Automated negotiation: prospects, methods and challenges," *Intern. J. of Group Decision and Negotiation*, vol. 10, no. 2, pp. 199–215, 2001.

[12] X. Zheng, P. Martin, and K. Brohman, "Cloud service negotiation: Concession vs. tradeoff approaches," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 515–522.

[13] Jade Site: Java Agent DEvelopment Framework, http://jade.tilab.com/.

[14] FIPA Interaction Protocols, http://www.fipa.org/repository/ips.php3.

[15] L. Wu, S. K. Garg, R. Buyya, C. Chen, and S. Versteeg, "Automated sla negotiation framework for cloud computing," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 235–244.

[16] A. Galati, K. Djemame, M. Fletcher, M. Jessop, M. Weeks, S. Hickinbotham, and J. McAvoy, "Designing an sla protocol with renegotiation to maximize revenues for the cmac platform," in *Web Information Systems Engineering–WISE 2011 and 2012 Workshops*. Springer, 2013, pp. 105–117.

[17] T. B. Quillinan, K. P. Clark, M. Warnier, F. M. Brazier, and O. Rana, "Negotiation and monitoring of service level agreements," in *Grids and Service-Oriented Architectures for Service Level Agreements*. Springer, 2010, pp. 167–176.

[18] O. Waeldrich, D. Battré, F. Brazier, K. Clark, M. Oey, A. Papaspyrou, P. Wieder, and W. Ziegler, "Ws-agreement negotiation version 1.0," in *Open Grid Forum*, vol. 35, 2011, p. 41.

Modeling Dynamic Location Update Strategies for PCS Networks

Chung-Chin Lu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan
Email: cclu@ee.nthu.edu.tw

Ruey-Cheng Shyu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan

Yung-Chung Wang
Department of Electrical Engineering
National Taipei University of Technology
Taipei, Taiwan
Email: ycwang@ntut.edu.tw

***Abstract*—Location management, accomplished through the backbone network and wireless links, is an important issue in a personal communication system (PCS). In this paper, we present a unified model to assess three dynamic location update strategies, including the distance-based, movement-based and time-based schemes, based on a seven-state Markovian mobility model on a two-dimensional hexagonal cellular topology. In our analysis, performance measures of each strategy can be evaluated efficiently.**

I. Introduction

In a PCS, the backbone network tracks the current location of a user and can therefore route messages to the user regardless of the user's location. In addition to its impact on signaling within the backbone network, a location management strategy influences the expenditure of wireless resources and the power consumption on portable terminals as well. Ideally, the location tracking strategy of each user should depend on user's current mobility behavior and call arrival pattern in order to minimize the consumption of wireless resources.

Recently, per-user based location update strategies, such as time-based, movement-based, and distance-based dynamic strategies, attract many researchers' attention.

Movenment-based location management scheme were analyzed in [1][2] based on a simple one-dimensional Markov walk model. The authors in [3] studied the movenment-based location update and shortest-distance-first selective paging scheme with Poisson call arrivals and general cell residence times. In [4], a simplifed two-dimensional random walk model capturing the movement of mobile users in PCS networks is proposed. In [5], a unified analytic framework for dynamic mobility management of mobile station is proposed by using a noval 2D Markov walk as the mobility model. The author in [6] performed the trade off analysis for the movement-based location update and paging for the wireless mobile networks. In [7], the authors study a dynamic movement-based location update scheme. However, the system topology is usually assumed to be one-dimensional or as a ring and the terminal mobility model is treated as a random walk or as a diffusion model for simplicity. By the way, the analytic models in the literature [1]–[7] just can assess the distance-based or movement-based dynamic location update schemes.

978-1-5090-3216-7/16/$31.00

In this paper, we consider the system to have a two-dimensional hexagonal cellular topology and adopt a Markovian mobility model to capture more practical situations in a real cellular system. A unified analysis is proposed for performance assessment of three dynamic location update schemes, including the distance-based, movement-based and time-based schemes. In addition, several assumptions such as very few incoming calls, adopted in the literature, are relaxed and a more general analysis and comparison of various dynamic update strategies are conducted in this paper.

II. System Formulation

The system model investigated in this paper consists of a cellular topology, a user mobility model and a distribution of arrivals of incoming calls. We assume that the time axis is slotted. We adopt a discrete-parameter seven-state Markovian model to describe the movement behavior of mobile on the two-dimensional hexagonal cellular topology. Finally, the arrivals of incoming calls are modeled as a Bernoulli process.

A. User Mobility Model

A movement will be defined as a user crossing the boundary of two adjacent cells. To describe the user mobility model, we note that the slot time is chosen small enough such that a user can make at most one movement during a slot time.

Among the seven states (called movement states) in the model, the stationary state (S) corresponds to the situation that there is no movement during a time slot and each of the other six states corresponds to a movement from a cell to one of the six adjacent cells on the hexagonal cellular topology. Let $\tilde{U}(t)$ be the movement state of a user in the t-th time slot. The state transition is assumed to be completed at the end of each time slot. The movement process $\tilde{U}(t)$, $t = 0, 1, 2, \ldots$, of a user is modeled as a homogeneous discrete-parameter Markov chain with state space $\mathcal{S} = \{North, NE, NW, South, SE, SW, S\}$ and one-step state-transition probabilities $P_{s_1,s_2} = Pr\{\tilde{U}(t+1) = s_2 \mid \tilde{U}(t) = s_1\}$, $s_1, s_2 \in \mathcal{S}$. These transition probabilities are assumed to be symmetric in each direction as illustrated in Figure 1.

To facilitate the analysis of various dynamic location update strategies in the next section, we shall simplify the description

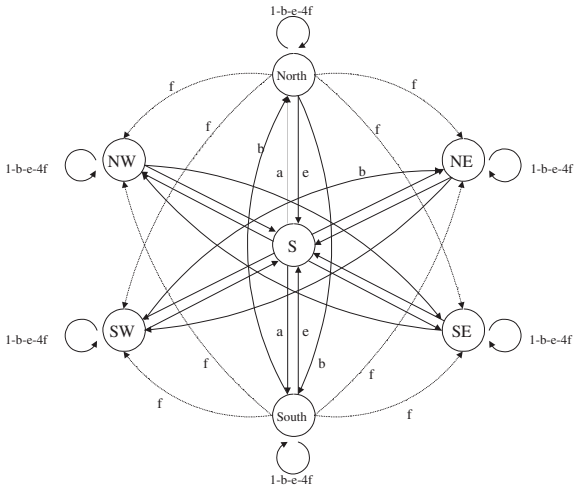


Fig. 1. The state-transition diagram of a seven-state Markovian mobility model.

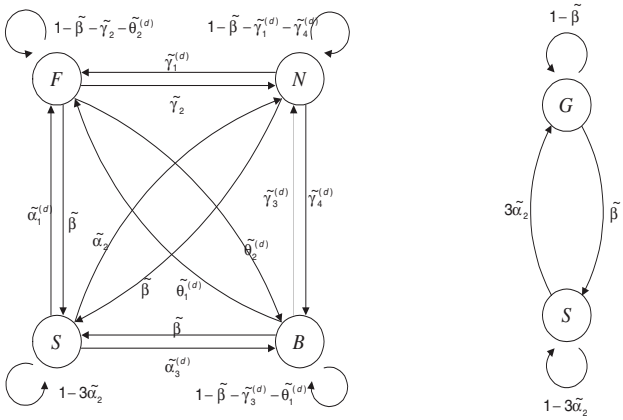


Fig. 2. A schematic representation for (a) distance-dependent transition probabilities of the four-state movement process $\tilde{V}(t)$ and (b) state-transition probabilities of the two-state movement process $\tilde{W}(t)$.

of the movement behavior in two different ways for different purposes. The first is based on a ring structure on the hexagonal cellular topology relative to the center cell. Let $\mathrm{V}(t)$ be another movement process of a user with state space $\{S, F, B, N\}$. $\mathrm{V}(t)$ will be called the four-state movement process, while $\mathrm{U}(t)$ will be called the seven-state movement process. The four-state process $\tilde{V}(t)$ at the t-th time slot is in the stationary state (S) if the seven-state process $\tilde{U}(t)$ is in the stationary state (S). $\tilde{V}(t)$ is in the forward state (F) at t if $\tilde{U}(t)$ is in a movement state such that the user will move up to a cell in the adjacent ring of larger distance. And, $\tilde{V}(t)$ is in the backward state (B) if $\tilde{U}(t)$ is in a movement state such that the user will move down to a cell in the adjacent ring of smaller distance. Finally, $\tilde{V}(t)$ is in the neighboring state (N) if $\tilde{U}(t)$ is in a movement state such that the user will move to an adjacent cell in the same ring.

Now, it is clear that $\tilde{X}(t+1)$ is a function of $\tilde{X}(t)$ and $\tilde{V}(t)$. By induction, $\tilde{X}(t+1)$ is a function of $\tilde{V}(t)$, $\tilde{V}(t-1)$, …, $\tilde{V}(0)$. Unlike the process $\tilde{U}(t)$, the movement process $\tilde{V}(t)$ is not Markovian. The following lemma sheds some light on a way to treat $\tilde{V}(t)$ and its proof is sketched in [8].

Lemma 1: $Pr\{\tilde{V}(t+1) = v_{t+1} \mid \tilde{V}(i), 0 \le i \le t\} = Pr\{\tilde{V}(t+1) = v_{t+1} \mid \tilde{X}(t+1), \tilde{V}(t)\}$. □

The conditional probabilities $Pr\{\tilde{V}(t+1) = v_1 \mid \tilde{V}(t) = v_0, \tilde{X}(t+1) = d\}$, $v_0, v_1 \in \{S, F, B, N\}$ and $d = 0, 1, \ldots$, denoted as $P^{(d)}_{v_0,v_1}$, are called distance-dependent state-transition probabilities of the four-state movement process $\tilde{V}(t)$. Figure 2-(a) illustrates these transition probabilities in a schematic way. The following theorem is also proved in [8].

Theorem 2: The joint distance and movement process $(\tilde{X}(t), \tilde{V}(t))$, $t = 0, 1, 2, \ldots$, *is a homogeneous discrete-parameter Markov chain. And,* $Pr\{\tilde{X}(t+1) = d_{t+1}, \tilde{V}(t+1) = v_{t+1} \mid \tilde{X}(t), \tilde{V}(t)\} = 1_{\{\tilde{X}(t+1)=d_{t+1}\}} Pr\{\tilde{V}(t+1) = v_{t+1} \mid \tilde{X}(t+1), \tilde{V}(t)\}$. □

The above theorem indicates that the one-step transition probabilities of the Markov chain $(\tilde{X}(t), \tilde{V}(t))$ can be determined by the distance-dependent transition probabilities $P^{(d)}_{v_0,v_1}$ of the four-state movement process $\tilde{V}(t)$.

The second way to simplify the description of the movement behavior of a user is to define a third movement process $\tilde{W}(t)$ for a user with state space $\{S, G\}$. The two-state movement process $\tilde{W}(t)$ is in the stationary state (S) if $\tilde{U}(t)$ is in the stationary state S. And, $\tilde{W}(t)$ is in the go state (G) if $\tilde{U}(t)$ is in any non-stationary state. Thus, $\tilde{W}(t)$ is a function of $\tilde{U}(t)$. Furthermore, we have the following theorem, whose proof is given in the Appendix of [8].

Theorem 3: The two-state movement process $\tilde{W}(t)$ *is a discrete-parameter Markov chain.* □

The state-transition diagram of $\tilde{W}(t)$ is shown in Figure 2-(b).

III. Dynamic Location Update Strategies

A. Distance-based Strategy

In distance-based strategy, a location update is initiated if the distance of a user traveled reaches D (in cell-diameters) from the cell the user last updated its location (i.e. the center cell) or after the successful completion of a paging process for an incoming call.

We use a joint distance and movement process $(X(t), V(t))$, $t = 0, 1, \ldots$, to describe the state of a user under the control of distance-based strategy and the influence of incoming call arrival process. $X(t)$ is the distance of the user from the cell the user last updated its location (i.e. the center cell) and takes values from 0 to $D-1$. $V(t)$ is the movement behavior of the user during the t-th time slot and takes values in the space $\{F, S, N, B\}$ of four movement states. Since the incoming call arrival process is a Bernoulli process and the tilded joint distance and movement process $(\tilde{X}(t), \tilde{V}(t))$, $t = 0, 1, \ldots$, is a Markov chain and independent of the incoming call process, the process $(X(t), V(t))$, $t = 0, 1, \ldots$, is indeed a Markov chain. For convenience, we define an ordering among states as

follows: $(0,F) < (0,S) < (1,F) < (0,S) < \cdots < (d,F) < (d,S) < (d,N) < (d,B) < \cdots < (D-1,B)$. Let T be the one-step transition probability matrix of the Markov chain $(X(t),V(t))$ with states indexed in the above order. It can be seen that

$$T = \begin{bmatrix} B_0 & A_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ C_0 & B_1 & A_2 & 0 & 0 & \cdots & 0 & 0 \\ E_0 & C_1 & B_2 & A_3 & 0 & \cdots & 0 & 0 \\ E_0 & 0 & C_2 & B_3 & A_4 & \cdots & 0 & 0 \\ E_0 & 0 & 0 & C_3 & B_4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ E_0 & 0 & 0 & 0 & 0 & \cdots & B_{D-2} & A_{D-1} \\ F_0 & 0 & 0 & 0 & 0 & \cdots & C_{D-2} & B_{D-1} \end{bmatrix}$$

where

$$A_1 = \begin{bmatrix} 1\text{-}\beta\text{-}\gamma_2\text{-}\theta_2^{(1)}\text{-}c & \beta & \gamma_2 & \theta_2^{(1)} \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$A_d = \begin{bmatrix} 1\text{-}\beta\text{-}\gamma_2\text{-}\theta_2^{(d)}\text{-}c & \beta & \gamma_2 & \theta_2^{(d)} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \ \forall 2 \le d \le D-1,$$

$$B_0 = \begin{bmatrix} c(1\text{-}\beta\text{-}c)/(1\text{-}c) & c\beta/(1-c) \\ 3\alpha_2/(1-c) & (1\text{-}3\alpha_2\text{-}c)/(1\text{-}c) \end{bmatrix},$$

$$B_d = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \alpha_1^{(d)} & 1\text{-}3\alpha_2\text{-}c & \alpha_2 & \alpha_3^{(d)} \\ \gamma_1^{(d)} & \beta & 1\text{-}\beta\text{-}\gamma_1^{(d)}\text{-}\gamma_4^{(d)}\text{-}c & \gamma_4^{(d)} \\ 0 & 0 & 0 & 0 \end{bmatrix}, \ \forall 1 \le d \le D\text{-}1,$$

$$C_0 = \begin{bmatrix} c(1-\beta-c)/(1-c) & c\beta/(1-c) \\ 3c\alpha_2/(1-c) & c(1\text{-}3\alpha_2\text{-}c)/(1\text{-}c) \\ c(1-\beta-c)/(1-c) & c\beta/(1-c) \\ (1-\beta-c)/(1-c) & \beta/(1-c) \end{bmatrix},$$

$$C_d = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \theta_1^{(d)} & \beta & \gamma_3^{(d)} & 1\text{-}\beta\text{-}\gamma_3^{(d)}\text{-}\theta_1^{(d)}\text{-}c \end{bmatrix}, \ \forall 1 \le d \le D\text{-}2,$$

$$E_0 = \begin{bmatrix} c(1-\beta-c)/(1-c) & c\beta/(1-c) \\ 3c\alpha_2/(1-c) & c(1\text{-}3\alpha_2\text{-}c)/(1\text{-}c) \\ c(1-\beta-c)/(1-c) & c\beta/(1-c) \\ c(1-\beta-c)/(1-c) & c\beta/(1-c) \end{bmatrix},$$

$$F_0 = \begin{bmatrix} (1-\beta-c)/(1-c) & \beta/(1-c) \\ 3c\alpha_2/(1-c) & c(1\text{-}3\alpha_2\text{-}c)/(1\text{-}c) \\ c(1-\beta-c)/(1-c) & c\beta/(1-c) \\ c(1-\beta-c)/(1-c) & c\beta/(1-c) \end{bmatrix}.$$

It can be seen that the Markov chain $(X(t),V(t))$ is ergodic and has steady-state probability distribution $\pi = (\pi_{0,F}, \pi_{0,S}, \pi_{1,F}, \ldots, \pi_{D-1,B})$ which is the unique non-negative solution of the matrix equation $\pi = \pi T$ with the sum of all components of π to be 1. Now, the averaged location update cost in a time slot is

$$C_u(D) = [(1-c)\pi_{D-1,F} + c]U \tag{1}$$

and the averaged terminal paging cost in a time slot is

$$C_p(D) = c\left\{\Gamma(0)(\pi_{0,F}+\pi_{0,S}) + \sum_{d=1}^{D-1}\Gamma(d)\sum_{v\in\{F,S,N,B\}}\pi_{d,v}\right\}P \tag{2}$$

where $\Gamma(d) = 3d(d+1)+1$ is the total number of cells with distance less than or equal to d from the center cell.

B. Movement-based Strategy

In movement-based strategy, a location update is initiated if a user travels across cell boundary M times since last update. Since the averaged paging cost is directly correlated to the distance, we proliferate the untilded joint distance and movement process $(X(t),V(t))$, $t = 0,1,2,\ldots$, discussed in the last subsection into a three-dimensional process $(Y(t),X(t),V(t))$, by including the counter $Y(t)$ of movements. $Y(t)$ is the number of movements from the time at which a user last updated his location to the current t-th time slot, taking values from 0 to $M-1$. The process $(Y(t),X(t),V(t))$ is again an ergodic Markov chain. Let T be the one-step state-transition probability matrix and π be the steady-state probability vector with component $\pi_{m,d,v}$ for state (m,d,v). We shall represent the matrix equation $\pi = \pi T$ as a system of balanced equations. Since there do not exist any state (m,d,v) with $m < d$, we shall let $\pi_{m,d,v} = 0$ for $m < d$ for convenience. Similarly, we let $\pi_{1,0,S} = \pi_{1,0,F} = 0$ and $\pi_{m,0,N} = \pi_{m,0,B} = 0$ for all $0 \le m \le M-1$. Then for $1 \le d \le m \le M-1$, we have

$$\begin{aligned}
\pi_{m,d,F} &= (1\text{-}\beta\text{-}\gamma_2\text{-}\theta_2^{(d)}\text{-}c)\pi_{m-1,d-1,F} + \alpha_1^{(d)}\pi_{m,d,S} \\
&\quad +\gamma_1^{(d)}\pi_{m-1,d,N} + \theta_1^{(d)}\pi_{m-1,d+1,B}, && \text{(3a)} \\
\pi_{m,d,S} &= \beta\pi_{m-1,d-1,F} + (1\text{-}3\alpha_2\text{-}c)\pi_{m,d,S} + \beta\pi_{m-1,d,N} \\
&\quad +\beta\pi_{m-1,d+1,B}, && \text{(3b)} \\
\pi_{m,d,N} &= \gamma_2\pi_{m-1,d-1,F} + \alpha_2\pi_{m,d,S} && \text{(3c)} \\
&\quad +(1\text{-}\beta\text{-}\gamma_1^{(d)}\text{-}\gamma_4^{(d)}\text{-}c)\pi_{m-1,d,N} + \gamma_3^{(d)}\pi_{m-1,d+1,B}, \\
\pi_{m,d,B} &= \theta_2^{(d)}\pi_{m-1,d-1,F} + \alpha_3^{(d)}\pi_{m,d,S} + \gamma_4^{(d)}\pi_{m-1,d,N} \\
&\quad +(1\text{-}\beta\text{-}\gamma_3^{(d)}\text{-}\theta_1^{(d)}\text{-}c)\pi_{m-1,d+1,B} && \text{(3d)}
\end{aligned}$$

and for $2 \le m \le M-1$,

$$\begin{aligned}
\pi_{m,0,F} &= (1-\beta-c)\pi_{m-1,1,B} + 3\alpha_2\pi_{m,0,S}, \\
\pi_{m,0,S} &= \beta\pi_{m-1,1,B} + (1-3\alpha_2-c)\pi_{m,0,S}
\end{aligned} \tag{3e}$$

and finally

$$\begin{aligned}
\pi_{0,0,F} &= c\left(\frac{1-\beta-c}{1-c}\right)\pi_{0,0,F} + \left(\frac{3\alpha_2}{1-c}\right)\pi_{0,0,S} \\
&\quad +c\left(\frac{3\alpha_2}{1-c}\right)\sum_{m=1}^{M-1}\pi_{m,S} + c\left(\frac{1-\beta-c}{1-c}\right)\sum_{m=1}^{M-2}\pi_{m,G} \\
&\quad +\left(\frac{1-\beta-c}{1-c}\right)\pi_{M-1,G} && \text{(3f)} \\
\pi_{0,0,S} &= c\left(\frac{\beta}{1-c}\right)\pi_{0,0,F} + \left(\frac{1-3\alpha_2-c}{1-c}\right)\pi_{0,0,S} \\
&\quad +c\left(\frac{1-3\alpha_2-c}{1-c}\right)\sum_{m=1}^{M-1}\pi_{m,S} + c\left(\frac{\beta}{1-c}\right)\sum_{m=1}^{M-2}\pi_{m,G} \\
&\quad +\left(\frac{\beta}{1-c}\right)\pi_{M-1,G} && \text{(3g)}
\end{aligned}$$

where

$$\begin{aligned}
\pi_{m,G} &\equiv \textstyle\sum_{d=0}^{m}(\pi_{m,d,F} + \pi_{m,d,N} + \pi_{m,d,B}), \\
\pi_{m,S} &\equiv \textstyle\sum_{d=0}^{m}\pi_{m,d,S}, \ 0 \le m \le M-1.
\end{aligned} \tag{3h}$$

It seems complicated to calculate the steady-state probabilities from the balanced equations. But it can be observed that once $\pi_{0,0,S}$ and $\pi_{0,0,F}$ are obtained, all other steady-state probabilities can be calculated by the balanced equations (3a) – (3e). Now, the averaged terminal paging cost in a time slot is

$$C_p(M) = c \sum_{d=0}^{M-1} w_d \Gamma(d) P \tag{4}$$

where $w_d = \sum_{m=d}^{M-1} \sum_{v \in \{F,S,N,B\}} \pi_{m,d,v}$ and $\Gamma(d) = 3d(d+1)+1$ as defined before.

C. Time-based Strategy

In time-based strategy, a location update is initiated if the time interval during which a user does not update its location reaches T time slots. To describe the behavior of a user under the control of time-based strategy, we define a two-dimensional process $(Z(t), W(t))$, $t = 0, 1, \ldots$, where $Z(t)$ is the number of time slots lapsed from the time at which a user last updated his location to the current t-th time slot, taking values from 0 to $T-1$ and $W(t)$ is the two-state movement behavior of the user during the t-th time slot as described in the last subsection. With the same reasons as for the process $(Y(t), W(t))$, the process $(Z(t), W(t))$, $t = 0, 1, \ldots$, is also an ergodic Markov chain. As in the analysis of averaged paging cost of movement-based strategy, we proliferate the untilded joint distance and movement process $(X(t), V(t))$, $t = 0, 1, 2, \ldots$, into a three-dimensional process $(Z(t), X(t), V(t))$, by including the counter $Z(t)$ of time slots. The process $(Z(t), X(t), V(t))$ is again an ergodic Markov chain. The system of balanced equations to find the steady-state probabilities $\pi_{t,d,v}$ is as follows. Again for convenience, we let $\pi_{t,d,v} = 0$ for any $t < d$ and $\pi_{t,0,N} = \pi_{t,0,B} = 0$ for all $0 \le t \le T-1$. Then for $1 \le d \le t \le T-1$, we have

$$\begin{aligned}
\pi_{t,d,F} &= (1\text{-}\beta\text{-}\gamma_2\text{-}\theta_2^{(d)}\text{-}c)\pi_{t-1,d-1,F} + \alpha_1^{(d)}\pi_{t-1,d,S} \\
&\quad + \gamma_1^{(d)}\pi_{t-1,d,N} + \theta_1^{(d)}\pi_{t-1,d+1,B}, && \text{(5a)} \\
\pi_{t,d,S} &= \beta\pi_{t-1,d-1,F} + (1-3\alpha_2-c)\pi_{t-1,d,S} && \text{(5b)} \\
&\quad + \beta\pi_{t-1,d,N} + \beta\pi_{t-1,d+1,B}, && \text{(5c)} \\
\pi_{t,d,N} &= \gamma_2\pi_{t-1,d-1,F} + \alpha_2\pi_{t-1,d,S} && \text{(5d)} \\
&\quad + (1\text{-}\beta\text{-}\gamma_1^{(d)}\text{-}\gamma_4^{(d)}\text{-}c)\pi_{t-1,d,N} + \gamma_3^{(d)}\pi_{t-1,d+1,B}, \\
\pi_{t,d,B} &= \theta_2^{(d)}\pi_{t-1,d-1,F} + \alpha_3^{(d)}\pi_{t-1,d,S} + \gamma_4^{(d)}\pi_{t-1,d,N} \\
&\quad + (1\text{-}\beta\text{-}\gamma_3^{(d)}\text{-}\theta_1^{(d)}\text{-}c)\pi_{t-1,d+1,B} && \text{(5e)}
\end{aligned}$$

and for $1 \le t \le T-1$,

$$\begin{aligned}
\pi_{t,0,F} &= (1-\beta-c)\pi_{t-1,1,B} + 3\alpha_2\pi_{t-1,0,S}, && \text{(5f)} \\
\pi_{t,0,S} &= \beta\pi_{t-1,1,B} + (1-3\alpha_2-c)\pi_{t-1,0,S} && \text{(5g)}
\end{aligned}$$

and finally

$$\begin{aligned}
\pi_{0,0,F} &= c\left(\frac{1\text{-}\beta\text{-}c}{1-c}\right)\sum_{t=0}^{T-2}\pi_{t,G} + c\left(\frac{3\alpha_2}{1-c}\right)\sum_{t=0}^{T-2}\pi_{t,S} \\
&\quad + \left(\frac{1\text{-}\beta\text{-}c}{1-c}\right)\pi_{T-1,G} + \left(\frac{3\alpha_2}{1-c}\right)\pi_{T-1,S}, && \text{(5h)} \\
\pi_{0,0,S} &= c\left(\frac{\beta}{1\text{-}c}\right)\sum_{t=0}^{T-2}\pi_{t,G} + c\left(\frac{1\text{-}3\alpha_2\text{-}c}{1-c}\right)\sum_{t=0}^{T-2}\pi_{t,S} \\
&\quad + \left(\frac{\beta}{1\text{-}c}\right)\pi_{T-1,G} + \left(\frac{1\text{-}3\alpha_2\text{-}c}{1-c}\right)\pi_{T-1,S} && \text{(5i)}
\end{aligned}$$

with

$$\begin{aligned}
&\pi_{t,G} \equiv \textstyle\sum_{d=0}^{t}(\pi_{t,d,F} + \pi_{t,d,N} + \pi_{t,d,B}), \\
&\pi_{t,S} \equiv \textstyle\sum_{d=0}^{t}\pi_{t,d,S},\ 0 \le t \le T-1.
\end{aligned} \tag{5j}$$

All other steady-state probabilities $\pi_{t,d,v}$ can be calculated by the balanced equations (5a) – (5g). Now, the averaged terminal paging cost in a time slot is

$$C_p(T) = c \sum_{d=0}^{T-1} w_d \Gamma(d) P \tag{6}$$

where $w_d = \sum_{t=d}^{T-1} \sum_{v \in \{F,S,N,B\}} \pi_{t,d,v}$. and $\Gamma(d) = 3d(d+1)+1$ as defined before.

IV. Conclusion

In this paper, we have presented performance modeling of distance-based, movement-based and time-based dynamic location update strategies, based on a seven-state Markovian mobility model on a two-dimensional hexagonal cellular topology. Since our model is based on a two-dimensional cellular system, we believe that a system designer can exploit our results to evaluate the performance of various location tracking strategies to a practical cellular system. Based on the analysis in Section III, a numerical performance comparison is conducted in [8], which shows that with the consideration of cost performance as well as implementation complexity, time-based strategy should be an excellent choice.

References

[1] H. Xie, S. Tabbane, and D. J. Goodman, "Dynamic location area management and performance analysis," *Proc. IEEE VTC'93*, pp. 536–539.

[2] A. Bar-Noy, I. Kessler, and M. Sidi, "Mobile users: To update or not to update?" *Wireless Networks*, vol. 1, no. 2, pp. 175–186, July 1995.

[3] I. Akyildiz, S. M. Ho, and Y. B. Lin, "Movement-based location update and selective paging for PCS networks," *IEEE/ACM Trans. Networking*, vol. 4, pp. 629–638, Aug. 1996.

[4] I. Akyildiz, Y. B. Lin, W. R. Lai, and R. J. Chen, "A new random walk model for PCS networks," *IEEE J. Select. Area Commun.*, vol. 18, no. 7, pp. 1254–1261, July 2000.

[5] C. H. Wu, H. P. Lin, and L. S. Lan, "A new analytic framework for dynamic mobility management of PCS networks," *IEEE Trans. Moble Computing*, vol. 1, no. 3, pp. 208–220, July-Sept. 2002.

[6] Y. Fang, "Movement-based mobility management and trade off analysis for wireless mobile networks," *IEEE Trans. Cmputers*, vol. 52, no. 6, pp. 791–803, June 2003.

[7] J. Li, H. Kamada, and K. Li, "Optimal dynamic mobility management for PCS networks," *IEEE/ACM Trans. Networking*, vol. 8, no. 3, pp. 319–327, June 2000.

[8] C. C. Lu, R. C. Shyu, and Y. C. Wang, "Performance evaluation of dynamic location update strategies for PCS networks," preprint.

Client-Side Monitoring Techniques for Web Sites

Ricardo Filipe and Filipe Araujo

CISUC, Dept. of Informatics Engineering
University of Coimbra
Coimbra, Portugal
rafilipe@dei.uc.pt, filipius@uc.pt

***Abstract*—Ensuring the correct presentation and execution of web sites is a major concern for system developers and administrators. Unfortunately, only end users can determine which resources are available and working properly. For example, some internal or external addresses might be unavailable or unreachable for specific clients, while seemingly available resources, like JavaScript, might run with errors in some browsers. While standard monitoring and analytic tools certainly provide valuable information on web pages, problems might still escape such measures, to reach end web users. To demonstrate the limitations of current tools, we ran an experiment to count web page errors in a sample of 3,000 web sites, including network and JavaScript errors. Our results are significant: as many as 16% of the top 1,000 sites have errors in their own resources; less popular sites have even more. Based on these results, we make a review of three client-side monitoring approaches to mitigate such errors: stand-alone applications, browser extensions and JavaScript snippets with analytic tools. Interestingly, even the latter approach, which requires no software installation, and involves no security changes, can cover a large fraction of existing web errors.**

***Index Terms*—Web monitoring, Client-side monitoring, Analytics.**

I. Introduction

Web site monitoring plays a major role in mitigating the negative consequences of programming errors and network malfunctions. Several studies [1], [2] show the significant impact for companies, when users experience blank pages, missing items, or are unable to interact with the web page. One may consider that the consequences would be minimal, but the true costs of a web page malfunction come from disgruntled customers and the respective impact on the company's reputation. To control failures in web resources, system administrators must keep a watchful eye on a large range of system parameters, like memory occupation, network interface utilization, among an endless number of other metrics, such as page load time. Unfortunately, even with all these metrics — that add complexity and intrusiveness to the system —, clients may experience some problems, due to web page external content, and client specific conditions, such as network glitches or incompatible browser versions. In fact, even sites belonging to the top-50 of the world wide web have errors [3], thus suggesting that expensive monitoring mechanisms cannot provide a completely accurate picture of web page reliability. Another study [4] shows that an earlier detection of failures would reduce the majority of customer complaints. Indeed, customer feedback is a key aspect for web page trustworthiness since some server issues might not produce the same effects in all clients.

978-1-5090-3216-7/16/$31.00

We argue that there are still no effective means to easily detect web page problems. A possible approach is to send browsing data to some analytic tool. These tools may handle problems such as nonexistent pages in the domain, but, since they are oriented to advertising and search engine optimization, they typically neglect correct web page display.

To demonstrate that the web currently suffers from a lack of proper monitoring, in Section II, we describe an experiment with 3,000 sites of the top 1,000,000 web sites of the Alexa ranking [5]. The 3,000 samples cover sites from the range 1-1,000, 10,000-20,000 and also from 100,000-200,000. We used the Chrome web browser from two distinct locations in Europe, to ensure realistic access to web pages, and simulate real user interactions. We collected metrics, such as network errors, broken links or JavaScript problems.

We think that the results we present in Section III are noteworthy: 16% of the top 1,000 sites have errors in their web page resources, being this value higher for less popular sites. This demonstrates that no widespread monitoring tool effectively prevents these errors. Then, in Section IV, we proceed to discuss possible client-side monitoring solutions, to complement available monitoring techniques. We compare the client-side solutions, based on their level of intrusiveness for the client. Unfortunately, as we might expect, approaches that can collect more metrics are also much more intrusive. Nevertheless, even non-intrusive light-weight approaches can cover a significant fraction of web errors. These light-weight approaches have the additional benefit of not compromising client security or increasing the complexity of monitoring.

II. Experimental Settings

Before rendering and displaying an HTML (HyperText Markup Language) page, browsers must first fetch the page from a server, using a URL (Uniform Resource Locator). The browser then goes through the page, to build the DOM (Document Object Model), render the corresponding tree and

TABLE I. SOFTWARE USED AND DISTRIBUTION.

Component	Observations	Version
Selenium	selenium-server-standalone jar	2.45.0
Chrome	browser	48.0.2564.103
Chrome	driver	2.21.371461
Xvfb	xorg-server	1.13.3

display it. The browser might need to download other resources referenced in the main page. To fetch these resources, the browser opens several TCP [6] (Transmission Control Protocol) connections to their respective server, either internal or external to the domain. However, each of these resources is subject to failures that might impact the user experience. To analyze the extension of problems, we inspected 3,000 sites, including the most popular ones, and looked for very specific metrics:

- We decomposed network errors into DNS (Domain Name System) errors, in the phase of name lookup; TCP, if the connection crashes or the server is unreachable; and *other* errors.
- For HTTP errors related to resources, we looked to the range 4xx and 5xx. In our experiments, we do not count how many of these errors exist in a single page, but only whether they exist.
- We count broken links, where the server responds with a 4xx or 5xx HTTP code. Again, we only count the number of web sites that have errors in these ranges.
- We also care for other errors related to resources: fonts, style sheets, images and JavaScript. These errors might originate in the network layer, while processing the script (if applied), or in the cancellation of a resource download, e.g., because a change in the page made it unnecessary.

For the sake of doing an online analysis of the web sites, we used Selenium [7], to emulate clients accessing web pages through browsers. We used the Xvfb virtual display emulator for the client machine. This display performs all graphical functions, without actually needing a real screen, thus allowing Selenium to run without a terminal. We wrote a program in Java that ran in the background, attached to this display emulator. This program used Selenium and Chrome, to access a list of web sites. We used a Linux machine, running in the facilities of our department in Portugal and another instance in Hungary, at MTA SZTAKI's laboratories [8]. Clients running from different locations experienced different network connectivities, thus displaying distinct perspectives for the same web page, like resources inaccessible from only one of the locations. Additionally, since programs ran autonomously, with a time lapse of several hours, they occasionally observed different page errors. Table I lists the software we used and the respective versions.

This program used as input a file that we retrieved from Alexa [5], with the top one million ranking sites. We sequentially analyzed 3,000 sites from this file: from pages 1 to 1,000; then from rank 10,000 to 20,000 with steps of 10 (e.g. rank 10,000; 10,010; 10,020;...) and finally, from 100,000 to 200,000 with steps of 100.

Additionally, and one of the most relevant aspects of our work is that we parse the main HTML page, to get all links accessible to the users through web page interaction. We follow and invoke these links, to check if any HTTP error occurs, with error codes 4xx or 5xx, related to client or server errors, respectively. This information is important, because the availability of the links is tightly connected to the utility of the web page. As we shall see the number of broken links is surprisingly high, even in top web sites.

III. RESULTS

In this section, we present the results of our experiment. The experiment took several days to finish, mostly due to the invocation of links associated with each web page. Tables IIa and IIb present the most significant results we got. To conserve space, we only display the results of our client in Portugal, as the client in Hungary got similar results.

In Table IIa, we analyze the number of web pages with network or HTTP errors. This table displays problems with page resources (HTTP 4xx and 5xx), e.g., some image; connection errors (DNS, TCP and other) and broken links, i.e., links that point to resources outside the page and that exhibit some problem. Connection errors are all related to the main HTML page or one of its resources. As we mentioned before, the numbers in the table refer to the total number of sites where we could observe the problem. This means that, for example, in the first line, first column of Table IIa, the number of HTTP 4xx errors in the top 1,000 sites is 161. I.e., 161 sites have one or more resources that are not accessible and return a 4xx error code.

The number of errors is quite high in general, especially in lower raking sites. Differences between the first and the other two rows of the table are blatant, for most metrics. This is true for internal problems and for external links, including network error conditions, which are also much less frequent in the top ranking sites. Most problems come from the external links that tend to break quite often, either with a 4xx or a 5xx error code (right side of the table). However, internal problems (left-side of the table) are arguably more important, as they might result in visible problems in the page layout. Interestingly, as much as 16% of the top-tier sites may suffer from some form of internal problem. This number is even higher for the lower rankings. The same is true for connectivity errors (center of the table). DNS, TCP and other forms of errors are less frequent in major sites. The HTTP 5xx error codes are the only ones where the frequency of problems seems stable across all rankings. This might be due to an inverse relation between complexity and ranking positions (i.e., more complex pages correspond to lower ranking numbers), but a clear demonstration of such hypothesis requires further study. Overall, these results suggest that top-tier sites either have better network connections, or more server resources, or both. We might say the same about the contents themselves, most

TABLE II. NUMBER OF SITES WITH ERRORS - PORTUGAL

(a) Network and HTTP errors

	HTTP 4xx errors	HTTP 5xx errors	DNS errors	TCP errors	Other Network errors	Broken Links 4xx	Broken Links 5xx
$range_{1000}$	161	62	68	27	96	115	65
$range_{10000}$	251	47	122	38	111	182	42
$range_{100000}$	291	51	113	37	114	193	43

(b) Resource errors and Event averages

	Resource Font errors	Resource style sheet error	Resource image error	Resource JS error	Resource JS External	Resource JS Internal	Resource JS Both	DOM Event	Load Event
$range_{1000}$	11	15	131	153	136	13	4	4517	33
$range_{10000}$	16	16	134	189	136	39	14	6097	24
$range_{100000}$	27	27	143	174	103	62	9	6963	19

likely due to significant advantages in the lifecycle of the web pages (one or more among design, development, testing, deployment, and maintenance).

In Table IIb, we show the number of sites that returned at least one resource error, for different types of resources. Resource problems include errors getting fonts, images or processing JavaScript (left side). Regarding JavaScript errors, we split data into external or internal to the web page domain, or both, if errors exist in internal and external resources. We can see that fonts and style sheet resources do not pose major problems for sites. The main offenders to web page reliability are images and JavaScript. Another interesting result is that top-tier sites have more errors in external JavaScript resources. This is an indication that top pages rely more on standard libraries, normally hosted in another domain.

IV. CLIENT-SIDE MONITORING

In this section, we discuss some solutions that might serve to improve web page reliability. We suggest three different options involving different levels of transparency to the client: a stand-alone approach, a browser extension and a JavaScript snippet.

For a stand-alone application (similar to the one we used in Section II), the possibilities are endless. By having total control of the browser and resorting to a testing framework such as Selenium, developers can test pretty much anything. The major disadvantage of this approach is that a customized stand-alone application is not practical or reasonable to install on the clients. Monitoring a site in this way, would therefore be limited to a handful probes controlled by the site owners.

A second approach would be to install a browser extension to get the most important metrics from the web page interaction and send them to a central monitoring site (we refer to some work using browser plugins in Section V). Extensions could bypass some of the security constraints associated with JavaScript. For example, with extensions, it would be possible to have access to the browser APIs and therefore to network logs. This would enable network error collection (DNS, TCP and others). Additionally, the extension could invoke links associated with a web page. Unfortunately, extensions have two setbacks: firstly, it does not look feasible to convince hundreds or thousands of users to install some browser extension that could raise issues, concerning security and privacy; secondly, different extensions should be developed for each different browser, thus entailing a great effort and cost.

As a third approach, site owners might use JavaScript and AJAX in the web pages they serve, to collect error information. By precluding the need for special software, this would rule out the shortcomings of the previous approaches. Furthermore, this would allow for a very simple integration with analytic tools, like Google Analytics [9]. Naturally, this can only work for resources inside the main page, once the browser loads the JavaScript. Collecting different kinds of errors with JavaScript and AJAX raises a number of challenges, but it is still possible, as we can see in the following list:

1) Networking Errors: with JavaScript, one can only infer DNS or TCP errors, using the Resource Timing API [10], as some browsers add entries in the PerformanceResourceTiming array, for resources with network issues.

2) Internal 4xx Errors: it is possible to customize an HTTP 4xx page for this range of errors. As the user is redirected to this page, administrators will receive an alert.

3) Internal 5xx Errors: the browser may analyze the connection and the response times. If the former is different from zero, while the latter is zero, the browser has an indication that it could not retrieve the resource from the server.

4) External Broken Links: we might invoke links from a proxy, using some AJAX solution [11]. This proxy will then invoke the URL and return the request in JSON, thus not breaking cross-domain security.

5) Resource Errors: regarding JavaScript exceptions and console logs, it is possible to use the `window.addEventListener` for *error* events with the `useCapture` argument set to `true` or use `window.onerror` event. This will retrieve the element or script that originated the error, and not the specific error message. We briefly compare the alternatives in Table III.

TABLE III. COMPARISON OF METHODOLOGIES

	Stand-alone Application	Browser Extension	JavaScript Code
Network Problems	Y	Y	Y Indirectly for resources
Broken Links	Y	Y	Y Proxy
JavaScript Errors	Y	Y	Y
Real-world application	Hard to deploy	Security constraints	Easier to scale and deploy

V. RELATED WORK

We divide previous work on web sites reliability into two categories: (i) methods or platforms that collect client metrics; (ii) studies regarding top sites reliability.

Regarding monitoring platforms, Dasu [12] is a client-based software that gathers metrics from different locations. It is limited by the number of hosts that are online, and discards HTML objects from third-party resources or JavaScript errors. In [13], Flach *et al.* use a browser plugin to analyze sites based on rules. This work only focuses on connectivity issues. In [14], authors propose a collaborative approach to detect performance problems. They use a web browser extension on each client and send all information to a central point, for processing. In [15], authors aim to detect user-visible failures, by analyzing Web logs and users' browsing patterns. However, the client may not react to the visible failure (e.g., by leaving the page, or not refreshing it) making this a major concern.

Regarding the reliability of top web sites, in [16] the goal is to collect web page evolution over time. [17] uses a different approach, implementing a web crawler that gathers HTTP, DNS and TCP connection data, to understand in which layer do most of the user-visible page failures occur. This, however, uses a customized crawler, instead of a common browser. In [18], authors gather network information from 80 sites, and analyze the source of the problems. However, recent studies suggest that this pattern of concurrent accesses can significantly change the results observed [19].

Unlike previous work, we consider a very wide range of sites, using a real web browser.

VI. CONCLUSION

The evidences we collected in this paper support the point of view that monitoring remains as a largely unsolved challenge to this day. Even large companies with vast resources fail to provide impeccable, failure-free, web sites. To mitigate this problem, we argue that web site providers must include client-side observations into their monitoring tools. Despite the trade-offs involved, the least intrusive mechanisms can still detect a large number of errors.

ACKNOWLEDGMENT

This work was partially carried out under the project PTDC/EEI-ESS/1189/2014 — Data Science for Non-Programmers, supported by COMPETE 2020, Portugal 2020-POCI, UE-FEDER and FCT.

REFERENCES

[1] "A study about online transactions, prepared for tealeaf technology inc, oct 2005," http://www-01.ibm.com/software/info/tealeaf/, retrieved: April, 2016.

[2] "Causes of failure in web applications (cmupdl-05-109), dec 2005," http://repository.cmu.edu/cgi/viewcontent.cgi?article=1047&context=pdl, retrieved: April, 2016.

[3] *No time for downtime: It managers feel the heat to prevent outages that can cost millions of dollars*, Internet Week, N.807, 3, Internet Week Std., Apr 2000.

[4] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, ser. USITS'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251460.1251461

[5] "Alexa — top-ranked websites," https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-, retrieved: April, 2016.

[6] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[7] "Selenium browser automation," http://www.seleniumhq.org/, retrieved: April, 2016.

[8] Z. Balaton, P. Kacsuk, N. Podhorszki, and F. Vajda, *From Cluster Monitoring to Grid Monitoring Based on GRM*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 874–881. [Online]. Available: http://dx.doi.org/10.1007/3-540-44681-8_121

[9] "Google analytics solutions," https://analytics.google.com/, retrieved: April, 2016.

[10] "Papers — Resource Timing," https://www.w3.org/TR/2016/WD-resource-timing-20160225/, retrieved: March, 2016.

[11] "Webpage — Yahoo Query Language (YQL)," https://developer.yahoo.com/yql/, retrieved: April, 2016.

[12] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger, "Dasu: Pushing experiments to the internet's edge," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 487–499.

[13] T. Flach, E. Katz-Bassett, and R. Govindan, "Diagnosing slow web page access at the client side," in *Proceedings of the 2013 Workshop on Student Workhop*, ser. CoNEXT Student Workhop '13. New York, NY, USA: ACM, 2013, pp. 59–62. [Online]. Available: http://doi.acm.org/10.1145/2537148.2537160

[14] S. Agarwal, N. Liogkas, P. Mohan, and V. Padmanabhan, "Webprofiler: Cooperative diagnosis of web failures," in *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, Jan 2010, pp. 1–11.

[15] W. Li and I. Gorton, "Analyzing web logs to detect user-visible failures," in *Proceedings of the 2010 Workshop on Managing Systems via Log Analysis and Machine Learning Techniques*, ser. SLAML'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1928991.1929000

[16] "Http archive," http://httparchive.org/, retrieved: April, 2016.

[17] C. Vaz, L. Silva, and A. Dourado, "Detecting user-visible failures in web-sites by using end-to-end fine-grained monitoring: An experimental study," in *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, Aug 2011, pp. 338–341.

[18] V. N. Padmanabhan, S. Ramabhadran, S. Agarwal, and J. Padhye, "A study of end-to-end web access failures," in *Proceedings of CoNEXT*, Lisboa, Portugal, December 2006.

[19] J. Sommers and P. Barford, "An active measurement system for shared environments," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298348

To Route or To Secure: Tradeoffs in ICNs over MANETs

Hasanat Kazmi* Hasnain Lakhani Ashish Gehani
SRI

Rashid Tahir
University of Illinois, Urbana-Champaign

Fareed Zaffar
Lahore University of Management Sciences

***Abstract*—Information-Centric Networks (ICNs) operating over Mobile Ad hoc Networks (MANETs) are challenged by the node churn, evolving topologies, and limited resources of the underlying network. The complex interplay of publishers, subscribers, and brokers brings with it a corresponding set of security concerns, where precisely-defined trust boundaries are needed to guarantee the confidentiality and integrity of all data objects in the ecosystem. Building a practical framework that can service users efficiently requires understanding the motivations and actions of the participants.**

We explore several tradeoffs between efficiency and the security of data objects in such environments, using ICEMAN – a real-wold implementation of an ICN that operates on MANETs. Since our findings are based on an actual system, they have significant implications for building efficient ICNs that have security designed in at the outset (rather than added later when options may be limited). We empirically establish that there is a strong interplay between the need to have more specific information for efficient routing and the need to ensure trust and confidentiality in such a decentralized system.

I. Introduction

Information-Centric Networking (ICN) is an approach for content distribution and retrieval that has drawn considerable attention in recent years. While there are several competing architectures and implementations, the underlying idea is that data is de-coupled from a single location. Network functionality is driven by descriptions of content rather than requests for the content at a specific address, as occurs in traditional source-destination based models. Publishers can advertise descriptions of their content. Subscribers advertise their interests in the hope that they will flow to others that have relevant objects. Data transport and routing decisions are content-aware, driven by matches between interests of nodes and the descriptions of the published content. The network can therefore take advantage of various performance optimizations, such as in-network caching in order to reduce latency and improve bandwidth utilization. However, these useful characteristics of ICNs come with a corresponding set of trust, privacy, and security challenges that need to be addressed.

Mobile Ad hoc Networks (MANETs) are used in environments where nodes can join or leave the network at will. In such high-churn environments, the ability to authenticate nodes in the absence of a single online trusted authority becomes fundamental for the correct and secure operation of the network. In the absence of an authentication scheme, a malicious node can access private objects and generate spurious content to overwhelm the network using resource exhaustion attacks. Similarly, the confidentiality of metadata is an important concern as its breach can lead to privacy compromises. In particular, query, response, and forwarding information can reveal sensitive details about publishers, subscribers, and content [2]. For instance, descriptive information about a data object is usually embedded in the associated metadata. ICN routing algorithms leverage these content descriptions to match objects with subscriber interests for forwarding decisions. However, the information present in the metadata also leaks privacy-sensitive details about the nodes involved in the production and consumption of the data objects.

Several approaches have been suggested in the literature to improve the security, privacy, and confidentiality of ICN-based publish-subscribe systems [5], [4], [11], [12], [1]. However, enhancing security and privacy generally leads to a suboptimal data delivery model with degraded performance. For instance, reducing the number of forwarding options at routers (to enhance security) causes data objects to follow longer paths. This results in reduced data rates and higher network congestion [14]. Additionally, this can also lead to routing *black holes* for data objects – that is, a data object might never be able to find a path to the subscriber [9]. Hence, a detailed investigation is needed to better understand the tradeoffs between the efficiency of the system and the security, privacy, and confidentiality of the actors and data involved. To this end, we present a detailed analysis and results from our measurement- and modeling-based study of security in ICEMAN, SRI International's open source implementation of an ICN for MANETs [15]. Specifically, we explore three tradeoffs: (i) the effect on routing performance when nodes must be authenticated, (ii) how caching policies affect access control performance (since in-network communication is used to retrieve credentials), and (iii) the impact on content delivery rates when stronger privacy protections are utilized for data descriptions and subscriber interests.

The rest of the paper is organized as follows. Section II describes ICEMAN and its privacy-enhancing architecture. Section III explains the tradeoffs we will examine. Our experimental evaluation and results are reported on in Section IV. We conclude in Section V.

*While visiting.

978-1-5090-3216-7/16/ $31.00

II. Background

ICEMAN uses the European Union project Haggle [7] as an integration framework, into which it adds implementations of multiple content dissemination algorithms, proactive and reactive utility-based caching, context-aware network coding, multi-authority node certification, access control, and interest privacy protection.

Participants: ICEMAN has three primary roles for nodes in the network: publishers, subscribers, and brokers. Publisher nodes add content with descriptive tags to the network. Subscriber nodes periodically broadcast node descriptions that include their interests. These descriptions are used by other nodes to identify which content matches a remote node's interests. Broker nodes forward content between publishers and subscribers, based on matches between the content tags and node interests.

Data Objects: Participants in ICEMAN share and receive data in units called *data objects*. Each such object has metadata associated with it, including a set of *tags* that are key-value attributes used to describe the content. Tags in the metadata are used to make forwarding decisions in the network. Content (such as a file) is inserted in a data object as its payload. A data object contains other metadata as well, such as the timestamp of when it was created, and a globally unique identifier, derived by computing a hash of its content and tags.

Data objects can also be exchanged between nodes without any payload. These data objects are announcements from a node to the network – for example, a node can send a *node description* that defines its interests to its neighbors. Nodes periodically retransmit their node descriptions to refresh their neighbors' view of their interests. Each node maintains a knowledge base of other nodes' cached content and interests.

Trust Model: ICEMAN supports the use of multiple concurrent authorities. This enables it to be resilient to failures of individual authorities and partitions of the network. Any node can serve as an authority, as long as other nodes agree to trust it. A node accepts another as an authority by receiving a shared secret from it out-of-band. This serves as the basis for securing messages from nodes to and from the authority. A node can send a Security Data Request ($SDReq$) to an authority to utilize its certification or authorization services. The authority uses a Security Data Response ($SDRes$) to return credentials to a node. These requests and responses are called Security Data Objects (SDOs) and are routed in the same manner as other data objects in the network.

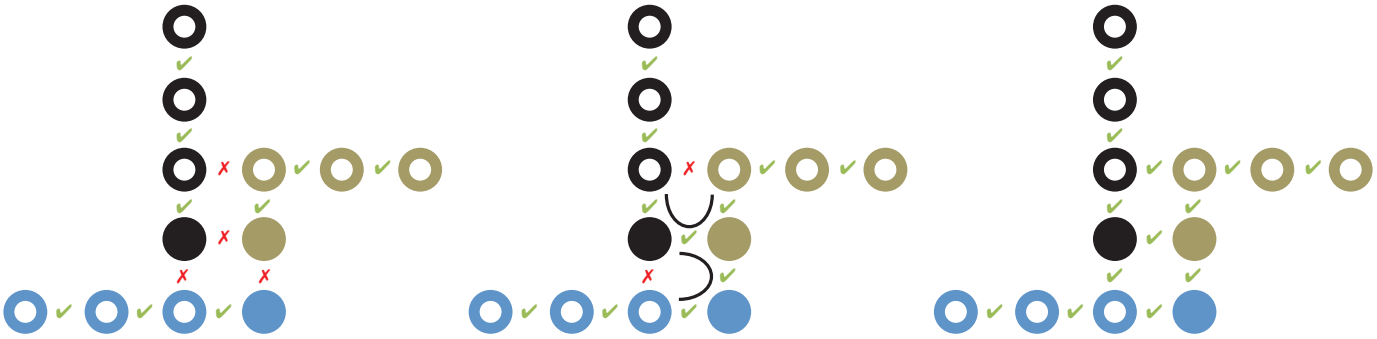
Authenticating Nodes: A node must choose to trust at least one authority in order to start participating in the network. As previously mentioned, the initial trust relationship between a node and an authority is established out-of-band. The resulting shared secret is then used to securely ask the authority to certify the node's identity certificate. The availability of multiple authorities increases the robustness of the system since each node can be certified by any authority.

When a node joins the network, it sends a self-signed identity certificate (in an $SDReq$) to the authority for signing. If the authority is configured to trust that node, it sends a signed version back (in an $SDRes$). Nodes only accept content from their neighbors if they are co-certified – that is, they share at least one trusted authority. Nodes $Alice$ and Bob exchange their certificates when they communicate with each other for the first time. If $C_{Alice,\alpha}$ is the set of certificates that $Alice$ has been issued by the set of authorities α, and $C_{Bob,\beta}$ is the set of certificates that Bob has been issued by set of authorities β, then trust is established if $\alpha \cap \beta \neq \emptyset$. This prevents an untrusted node from injecting content into the network.

Bootstrapping Trust: Nodes can join the network at any point in time and dynamically request that their identity certificates be signed by trusted authorities. If a node $Alice$ tries to join the network and there is only one node Bob in its vicinity, Bob cannot assume that $Alice$ has already interacted with a trusted authority to obtain a signed identity certificate. In particular, it is possible that Bob is the only node that $Alice$ can communicate with at the outset. Consequently, if Bob is not one of $Alice$'s trusted authorities, then Bob must relay communication from $Alice$ to authorities; otherwise, $Alice$ would not be able to get certified. ICEMAN addresses this trust bootstrapping conundrum by granting each node, such as $Alice$, a temporary window during which its $SDReq$ objects will be forwarded even though the node has not yet been co-certified. If $Alice$ receives a response from an authority in this window, it can join the network by including its signed certificate in future interactions. If the window expires without $Alice$ receiving a response, future data objects from it will be ignored by the network.

Content Routing: Data routing decisions are made by comparing a piece of content's tags with the interests of other nodes. Subscribers propagate their node descriptions containing a list of interests to other nodes that they encounter. Similarly, publishers share content with interested nodes that they encounter. Nodes aggregating the interests of others can serve as brokers to facilitate the hop-by-hop transportation of data objects from publishers to subscribers. Content caching in ICEMAN can either be *proactive* or *reactive* [8]. In proactive mode, a node pushes data objects to its neighbors based on the expectation that they will be interested in them in the future. In reactive forwarding, on the other hand, a node only sends a data object if the other node's *interests* and the content *tags* exceed a *matching threshold*.

Among other routing schemes, ICEMAN supports the use of a modified version of the DIsruption REsilient Content Transport (DIRECT) interest-driven content dissemination algorithm [13]. Every node periodically informs its neighbors about its interests by sending a timestamped node description. This is then propagated further in the network so that more nodes can respond if they have matching content cached. When a match occurs, the data object is forwarded along the reverse path – that is, to the neighbor from which the interest was first received. It is worth noting that interests are explicitly listed in a node description, which is then propagated through the network. This introduces the potential for significant dissemination of privacy-sensitive information.



(a) Scenario 1: Intra-group trust. (b) Scenario 2: Inter-group trust. (c) Scenario 3: Security not enabled.

Fig. 1: Three different co-certification scenarios are depicted. Authority nodes are represented with solid circles, while ordinary nodes are represented with rings. Only adjacent nodes are physically close enough to communicate directly.

Attribute-based Encryption: ICEMAN uses discretionary access control to define which nodes can read published content. This is enforced by limiting access to the payload of a data object using the *multi-authority* variant (MA-ABE) [6] of *attribute-based encryption* (ABE) [10]. Since MA-ABE is a type of *ciphertext-policy attribute-based encryption* [3], it embeds the access policy directly into the ciphertext. This is particularly useful in settings where potential subscribers are not known at the time of publication. Every publisher can encrypt each piece of content with a different access control policy. The ability of a node to decrypt a piece of content depends on the set of cryptographic attributes it has received from the authorities. Authorities are expected to issue credentials that correspond to the real-world characteristics of a node (such as its owner's organization, position, or role).

Content Access Control: Publishers encrypt content with an access policy before sending it to a remote node. The policy specifies which nodes can access the data, or more specifically, which combination of attributes are required to gain access. Note that each authority has a unique identifier, which determines the set of attributes for which it can issue encryption and decryption keys. The name of each attribute links it to the authority that issued it. With this approach, each publisher can construct a policy for its data requiring that any party that can decrypt the ciphertext has to possess a set of attributes issued by authorities that the publisher trusts. The publisher needs to know only the attributes that it uses in its policy. Each node requests encryption and decryption attributes from the authorities that it has established trust relationships with. It uses the shared secret key with the authority to establish secure communication for these requests. Nodes may request encryption and decryption attributes either on demand as they are needed to encrypt and decrypt content or through pre-provisioning – that is, requesting encryption and decryption attributes as soon as they join the network.

Metadata Access Control: Similar approaches are used by publishers and subscribers to limit the set of nodes that can act as brokers for their content and interests, respectively. A publisher encrypts each content tag with a policy that specifies which nodes are allowed to serve as brokers. Similarly, subscribers encrypt each interest with a policy that specifies which nodes can serve as brokers on their behalf. It is worth noting the flexibility of this framework, which allows each content tag and subscriber interest to be protected with an independent access policy. When a node receives a data object, it attempts to decrypt as many content tags as it can. Similarly, when it receives a node description, it attempts to decrypt as many interests as it can. Using the decrypted tags and interests, the node attempts to check if there is a sufficient match to forward the data object toward the subscriber.

III. Case Studies

The content dissemination and protection strategies in an ICN can interact in complex ways. We conducted a series of experiments to measure the effects of authorization, privacy, and caching policies on the efficiency and usability of ICEMAN. Below we report on three tradeoffs that we identified empirically:

Impact of Authentication on Routing: If all the nodes in a network act fairly and all the edges have the same bandwidth, the shortest path between any two nodes would be the optimal path for communication. However, as previously mentioned, a node trusts another node only if there is at least one authority that has certified both of them. If two nodes do not have such a trust relationship, the direct path between them cannot be used for any content exchange. This can lead to an increase in latency as the next optimal path may require more hops. Furthermore, this phenomenon can also partition the network graph into disconnected components, where no communication is possible between subsets of nodes. Greater trust increases routing efficiency at the cost of leaving the system increasingly vulnerable.

Impact of Caching on Authorization: ICEMAN uses encryption to ensure the confidentiality of data and to limit the nodes that can serve as brokers for content and interests. The efficiency with which a node can check for matches between a content's tags and a subscriber's interests depends on the extent to which the node can decrypt them. This may require the node to obtain cryptographic attributes from authorities, using SDOs. However, this process is complicated

by the fact that these objects are subject to caching policies at intermediate nodes between a requester and an authority. As storage pressure increases at a node, the SDOs may be evicted, adversely affecting authorization efficiency of remote nodes.

Impact of Privacy on Delivery Rate: An ICN router has access to content tags and interests in plaintext. ICEMAN addresses the privacy concerns of publishers and subscribers by allowing them to scope which nodes have access to the tags and interests, respectively. As more restrictive access policies are utilized, the privacy of publishers and subscribers increases. However, this also limits the nodes that can serve as brokers, bringing a concomitant reduction in routing robustness and efficiency.

IV. Evaluation

To ensure the repeatability of our experimental evaluation, we used the U.S. Naval Research Laboratory's Common Open Research Emulator (CORE 4.3) and Extendable Mobile Ad-hoc Network Emulator (EMANE 0.7.3) frameworks. Each ICN node's entire user-space code is run unmodified in a separate Linux (lightweight virtualization) *container* provided by CORE. The creation and movement of ICN nodes, publication of content, and subscription to interests, are orchestrated with scripts on each node, using CORE and EMANE programming interfaces.

A. Authentication / Routing Tradeoff

1) Goal – Understanding the baseline: Our first set of experiments was conducted to measure the effect of different configurations of trust among nodes on the time it takes to successfully deliver published data objects to interested subscribers. Figure 1 shows the physical arrangement of nodes in the experiment. Nodes are arranged in three groups, where each group has four nodes with one of the nodes an authority node (depicted with a solid circle). Nodes within a group trust each other as their identity certificates have been attested by the same authority.

Experimental Setup: We considered three scenarios:

1) *Intra-group trust* is established (as shown in Figure 1a). This case deals with the situation where there can be no data object exchanged across groups as no two nodes from different groups trust each other (as they are not co-certified).
2) *Inter-group trust* is established by letting all authority nodes in all groups co-certify each other. As shown in Figure 1b, this allows communication between groups but a data object may not follow the shortest path to a destination node.
3) *Ubiquitous trust* is shown in Figure 1c, where security is not enabled. Nodes are no longer required to be co-certified to exchange data objects. Hence, exchanges happen along the most optimal path permitted by the routing protocol.

Application Workload: In this experiment, 15 data objects were published and 31 subscriptions were issued. These publications and subscriptions were distributed among all groups and a total of 348 data objects could be delivered in the network (with multiple published data objects containing the same tags). All data objects were published and interests subscribed to within first 30 seconds of nodes initializing. Each data object had a payload that 512 KB in size.

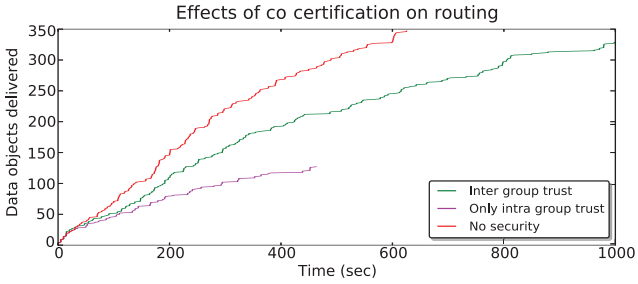


Fig. 2: Data objects delivered as a function of time. Individual plots correspond to different co-certification configurations.

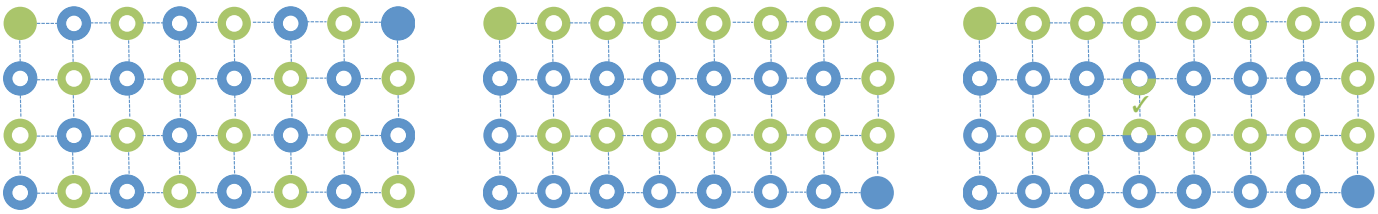
Results: Figure 2 shows the effect of various security policies on the performance of the data delivery. Lack of inter-group trust severely hampers data object dissemination across groups. This causes total network delivery to reduce significantly. An intermediate configuration allows inter-group communication but forces data objects to follow longer paths. This causes slower data delivery. Furthermore, a small fraction of data objects remain undelivered at the end of the experiment. Finally, a ubiquitous trust environment (where security is entirely disabled) removes any barriers for data object flow between nodes. This case is characterized by complete and fastest data object delivery across the network.

We can infer from this that if an ICN is designed for a public domain (such as use by first responders) where the primary focus is complete access to all data objects, the ICN should allow communication without the restriction of co-certification. On the other hand, enterprise and private ICNs, where data confidentiality is a higher priority, should either have a single authority or redundant authorities that certify all the nodes. For an option that allows a better balance between utility and security, please see the next section.

2) Goal – Understanding the benefit of bridge nodes: A node that has been co-certified by two or more authorities is trusted by the corresponding co-certification groups. It is therefore able to bridge the groups by forwarding data objects from one group to the others. We empirically study the effect of the presence of such bridge nodes on content delivery performance.

Experimental Setup: We considered the following five scenarios:

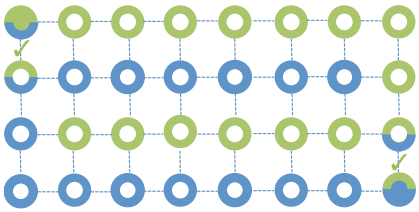
1) *Untrusted neighbors* are depicted in Figure 3a. In this scenario, each node is a member of one of two distinct trust groups. Each group has been separately co-certified



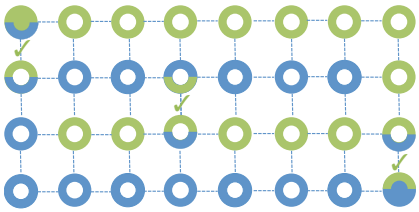
(a) Each node's neighbors are untrusted.

(b) Longest linear distinct trust groups.

(c) Single bridge between two trust groups.



(d) Two bridges between two trust groups.



(e) Three bridges between two groups.

Fig. 3: Co-certification creates trust groups. Five different configurations are shown here. Solid circles are authorities. Rings are ordinary nodes. Only adjacent nodes are close enough to communicate.

by a different authority. The nodes are arranged so that no two neighbors are from the same group. Hence, every node is surrounded by untrusted nodes.

2) *Longest linear groups* are shown in Figure 3b. Nodes in this setting are arranged to maximize the number of hops a data object can travel while remaining within a single trust group. The configuration is referred to as *maximum piping*.
3) *Single bridge* is a refinement of the previous case. The difference is the presence of a single bridge node, as depicted in Figure 3c.
4) *Two bridges* are illustrated in Figure 3d. The second bridge reduces (on average) the number of hops a data object must traverse to be able to cross from one trust group to another.
5) *Three bridges* are shown in Figure 3e. As expected, the availability of a third bridge further reduces the intra-group number of hops a data object must traverse before crossing into another trust group.

Application Workload: To facilitate a consistent comparison with the baseline, the same data object publication and interest subscription patterns were used as those described in Section IV-A1.

Results: Figure 4 reports the data delivery achieved (as a function of time) when different node trust configurations are utilized. As expected, disabling security completely results in the best data delivery (in the absence of any adversaries). To limit the attack surface of the ICN, the use of co-certification is recommended (to ensure outsiders cannot inject content into the system). In the baseline case we saw this impose a high negative impact on performance.

Our results here show that we can achieve both high performance and low risk through the judicious use of sufficient

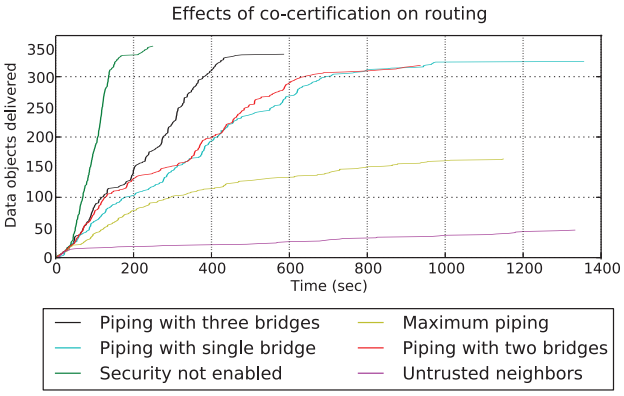


Fig. 4: Data objects delivered as a function of time. Individual plots correspond to different co-certification configurations.

bridge nodes. In particular, when three bridge nodes were used, we saw data delivery performance approaching the case when security was disabled. As the number of bridge nodes decreases to two and then to one, we continue to observe all the data objects delivered but over longer timespans.

In the case of maximum piping, data objects can only travel within a single trust group. Consequently, the maximum number of deliveries is limited by what can be retrieved without objects crossing over from the other group. Similarly, in the case that all neighbors are untrusted, the only objects delivered are from a node to itself. This occurs if one application on the node publishes a piece of content while the interests of another application match the content's tags.

Our findings provide a prescription for designers of ICNs with multiple trust domains. In this setting, it is critical to introduce enough bridge nodes in the architecture. While it

is well understood that increasing the number of such nodes is important for reliability, our results show the direct and significant effect on improving data delivery rates as well.

B. Authorization / Caching Tradeoff

1) Goal – Understanding the effect of caching policies on distributed access control: In the absence of dedicated control channels for security metadata, an ICN must utilize the underlying data forwarding infrastructure to send security-related requests and responses – that is, the SDOs described in Section II. Since access control in a distributed system is implemented through the use of SDOs from ordinary nodes to authorities and back, the caching policy at intermediate brokers determines the speed with which authorization requests complete.

Experimental Setup: We studied this question in a setting with the hourglass topology depicted in Figure 5. It contains two authorities α_1 and α_2 at opposite corners of the network. They are responsible for issuing encryption and decryption attributes to nodes that request them (and are authorized according to the authorities' configurations). The node β is the narrow waist of the hourglass. It separates the two sides of the network where α_1 and α_2 are located, respectively.

The bottleneck at node β facilitates controlled analysis of authorization performance as a function of the caching policy (at β). We consider the canonical case where a data object is published by node n_i with an access policy that contains an attribute from authority α_1. This data object has tags that match the interests of a node n_j on the opposite side of the network. Consequently, when n_j attempts to access the content, it will send a security data request that must traverse node β en route to the authority α_1. Similarly, the security data response from α_1 to n_j must go through β.

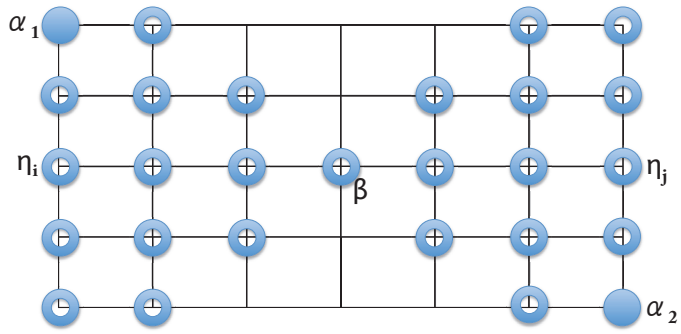


Fig. 5: Node β serves as a bridge between the groups of nodes on its left and right, credentialed by authorities α_1 and α_2, respectively.

We compared four representative caching strategies to understand their effect on authorization performance:

1) *Ubiquitous SDO prioritization.* All nodes, including bridges, use caching policies that give SDOs highest priority, minimizing the chance that they will be dropped at an intermediate broker en route to or from an authority.
2) *Only bridges are SDO-agnostic.* All nodes, except bridges, use caching policies that prioritize SDOs. This configuration models the case where bridges minimize per-object analysis to be able maximize throughput. In this setting, bridges will not distinguish between SDOs and other data objects.
3) *Only bridges prioritize SDOs.* In practice, the importance and low volume of SDOs may argue for bridges to be configured with caching policies that prioritize SDOs. This scenario assumes this but examines what happens when the remaining nodes do not perform similar prioritization.
4) *No SDO prioritization.* The final case provides a baseline for comparison. This scenario represents the ICN performance if it does not build in any cognizance of the effect of caching on authorization.

Application Workload: To saturate the cache at the bridge β, 8 data objects were published at the node. Each of these had content tags that matched interest subscriptions from the bridge. The 13 nodes on each side of the bridge each publish a single data object, for a total of 2 x 13. Each is encrypted with an access policy that contains a unique attribute that can be obtained from the authority that is reachable without crossing the bridge. Each node expresses an interest that matches an object that must traverse the bridge to arrive at the subscriber. Further, subscriptions are distributed across the network to balance the delivery overhead at all nodes.

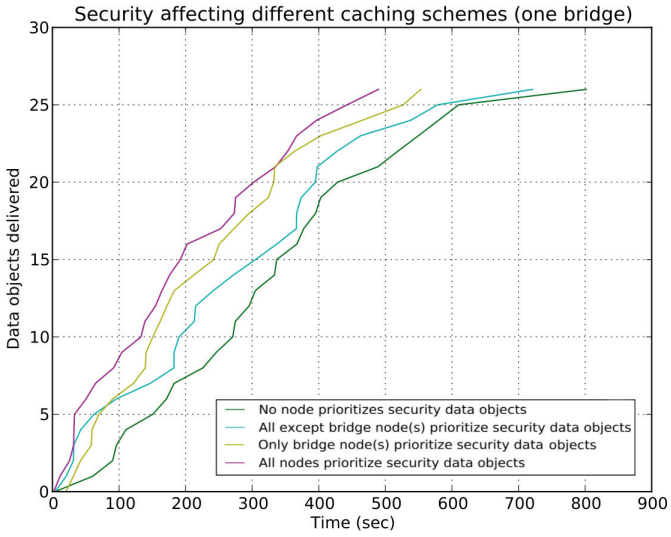


Fig. 6: Access control performance depends on the prioritization of SDOs in caching policies at nodes.

Results: Figure 6 describes how many objects have been delivered as a function of time. The four plots correspond to the different caching policy configurations described above. Recall that every data object's delivery is predicated on the completion of an authorization operation (that retrieves a cryptographic attribute from an authority on the opposite side of the bridge). So these plots are reporting on the effect of the caching policies on access control performance.

When all the nodes in the system prioritize SDOs, authorization and consequently data object delivery, completes fastest. When none of the nodes do so, performance is the worst. Interestingly, the prioritization of SDOs at the single bridge node does more to improve performance than such implementing this caching policy at all the remaining nodes in the system. This derives from the fact that the bridge is the bottleneck in communication.

In addition to affecting the average performance, caching policies also impact the variance in the performance. Configurations with policies that result in worse performance also result in higher standard deviations, as seen by the error bars in Figure 7.

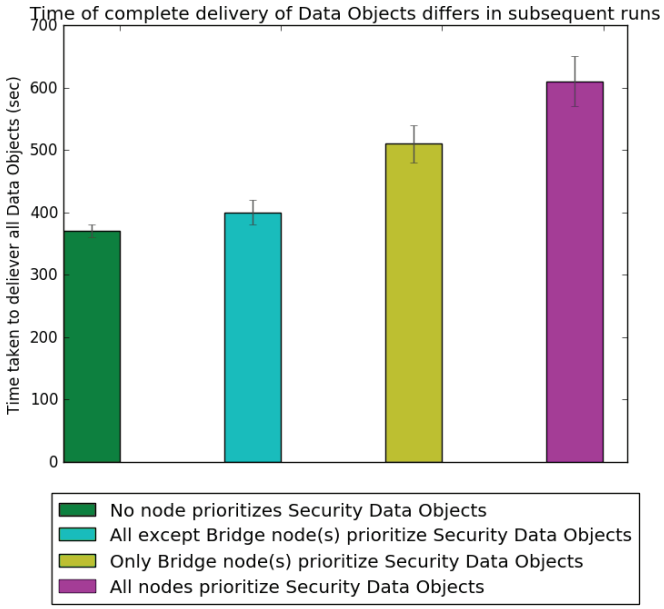


Fig. 7: Caching policies that result in longer times for access control requests to complete have the side-effect of larger variations in performance as well.

C. Privacy / Delivery Tradeoff

1) Goal – Pricing privacy primitives: ICEMAN protects the privacy of publishers and subscribers by allowing them to encrypt their content tags and interests, respectively. This encryption is effected using MA-ABE policies that scope the set of nodes that will have access to the tags and interests. Coarser policies result in lower privacy. Finer-grained policies protect the privacy of publishers / subscribers. To understand the price being paid for privacy, we measured the associated cryptographic costs.

Experimental Setup: We constructed the following micro-benchmarks:

1) *Encryption* measures the time for symmetric encryption of a tag or interest.
2) *Decryption* measures the time for symmetric decryption of a tag or interest.
3) *Capability generation* measures the time to encrypt a symmetric key with an MA-ABE access policy.
4) *Capability usage* measures the time to decrypt a symmetric key encrypted under an MA-ABE policy.

Workload: The set of micro-benchmarks was run, while varying the number of attributes used in the MA-ABE access policy. More attributes correspond to a more specific policy, which results in reduced privacy leakage.

Results: Figure 8 reports the increasing costs of more specific access policies. A runtime cost is introduced that results in each node taking longer to be able to access content tags and subscriber interests. The cost is commensurate with the number of attributes utilized.

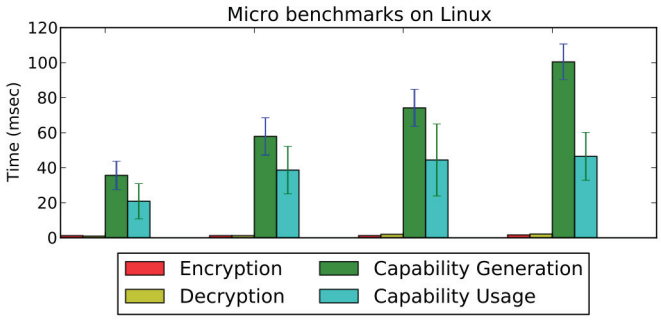


Fig. 8: As the number MA-ABE attributes used increases, the capability generation and usage times grow.

2) Goal – Pricing privacy-sensitive routing: A publisher can preserve their own privacy by limiting the nodes in the ICN that can see the tags they label their content with. Similarly, subscribers can maintain their privacy by scoping which nodes can see their interests. The consequence is that only some nodes can serve as brokers. To understand the impact these privacy protections have on the data delivery rates in the ICN, we conducted the following study.

Experimental Setup: The setting for this analysis was the 4 x 4 grid of nodes depicted in Figure 9. i, j, and k denote three groups of subscriber nodes. The nodes in i are also publishers. In this context, we studied the following three scenarios:

1) *Circumference routing* where the tags encrypted by nodes from group i can only be decrypted by nodes from groups i, j, and k.
2) *Universal routing* where all nodes have sufficient decryption attributes to be able to match and forward data objects.
3) *Promiscuous routing* where privacy protections are disabled.

Application Workload: Within the first 20 seconds after node initialization, they publish content or subscribe to specific interests. Each publisher shares 4 data objects, while subscribers express interests that match the tags on these objects. A maximum of 160 data objects can be delivered in this setting.

Results: If security is disabled, promiscuous routing results. In particular, both the tags of all content and the interests

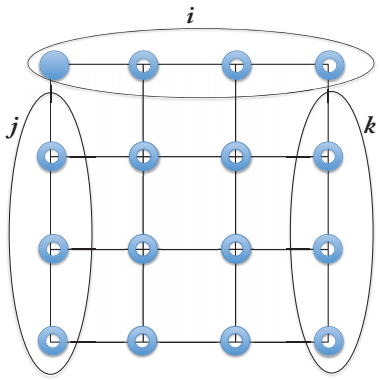


Fig. 9: Three levels of routing privacy protection were studied using this arrangement of nodes. They differed in which nodes could serve as brokers and the cost for doing so.

of subscribers are exposed for all intermediate nodes to see. This mode of operation provides no privacy protection but does result in the fastest delivery of content. Note that all data 160 objects are delivered by the 50 second mark, as seen in Figure 10.

In the case of universal routing, publishers apply protection to content tags, subscribers do the same for interests, and all intermediate nodes are provided with attributes that allow them to access content tags and interests. While privacy protections are used, all nodes are trusted enough to be able to serve as brokers. This allows multiple paths for content to flow from each publisher to each subscriber. The result is that data objects are all delivered by the 70 second mark. The reduction in speed is the result of brokers needing to decrypt tags and interests to perform matches.

Finally, circumference routing occurs when nodes outside groups i, j, and k are not trusted to access the private tags and interests of publishers and subscribers, respectively. The untrusted nodes cannot serve as brokers, preventing content from flowing through them. Consequently, content must be "piped" through a longer path. This results in the data objects taking close to 100 seconds to be delivered. The cost of maintaining tag and interest privacy is the slower delivery time.

V. Conclusion

Using the SRI's open source ICEMAN implementation of an ICN that operates over a MANET, we examined three trade-offs between maintaining the privacy of content publishers and subscribers on the one hand, and the performance of routing and content delivery on the other. In particular, we empirically demonstrated that (i) adding authentication to prevent outsiders from injecting content in the ICN comes at a cost for routing performance, (ii) using the data plane of an ICN for security-related communication requires corresponding cache policy prioritization, and (iii) the use of more privacy-sensitive content tags and subscriber interests increases the delivery time of affected content. We also make specific prescriptions for ICN architectures, including the use of sufficient bridge nodes, specific cache policy priorities, and relaxing privacy protection of metadata when possible.

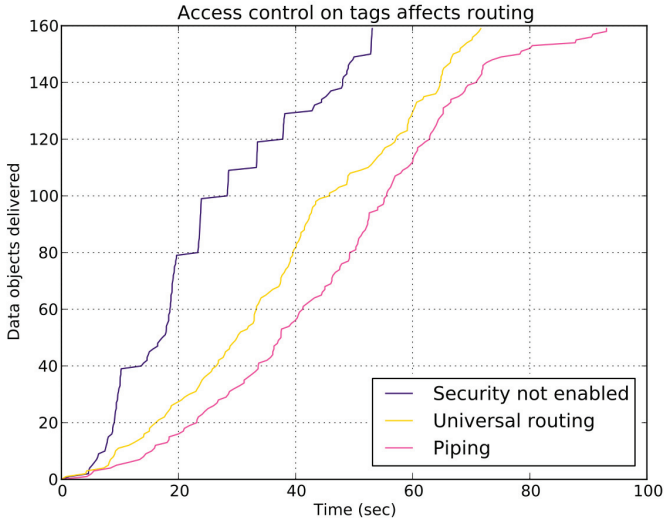


Fig. 10: Access control on tags and attributes slows down content delivery.

References

[1] Alexander Afanasyev, Priya Mahadevan, Ilya Moiseenko, Ersin Uzun, and Lixia Zhang, Interest flooding attack and countermeasures in Named Data Networking, *12th IFIP Networking Conference*, 2013.

[2] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman, A survey of information-centric networking, *IEEE Communications Magazine*, Vol. 50(7), 2012.

[3] John Bethencourt, Amit Sahai, and Brent Waters, Ciphertext-policy attribute-based encryption, *28th IEEE Symposium on Security and Privacy*, 2006.

[4] Mihaela Ion, Jianqing Zhang, and Eve Schooler, Toward content-centric privacy in ICN: Attribute-based encryption and routing, *3rd ACM SIGCOMM Workshop on Information-Centric Networking*, 2013.

[5] Jun Kuriharay, Ersin Uzun, and Christopher Wood, An encryption-based access control framework for content-centric networking, *14th IFIP Networking Conference*, 2015.

[6] Allison Lewko and Brent Waters, Decentralizing attribute-based encryption, *30th International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2011.

[7] Erik Nordstrom, Christian Rohner, and Per Gunningberg, Haggle: Opportunistic mobile content sharing using search, *Computer Communications*, Vol. 48, Elsevier, 2014.

[8] Soon Oh, Davide Lau, and Mario Gerla, CCN in tactical and emergency MANETs, *3rd IFIP Wireless Days Conference*, 2010.

[9] Mariana Raykova, Hasnain Lakhani, Hasanat Kazmi, and Ashish Gehani, Decentralized authorization and privacy-enhanced routing for ICNs, *31st Annual Computer Security Applications Conference*, 2015.

[10] Amit Sahai and Brent Waters, Fuzzy identity-based encryption, *24th International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2005.

[11] Abdullatif Shikfa, Melek Onen, and Refik Molva, Bootstrapping security associations in opportunistic networks, *6th International Workshop on Mobile Peer-to-Peer Computing*, 2010.

[12] Abdullatif Shikfa, Melek Onen, and Refik Molva, Privacy and confidentiality in context-based and epidemic forwarding, *Computer Communications*, Elsevier, Vol. 33(13), 2010.

[13] Ignacio Solis and J. J. Garcia-Luna-Aceves, Robust content dissemination in disrupted environments, *3rd ACM Workshop on Challenged Networks*, 2008.

[14] Reza Tourani, Travis Mick, Satyajayant Misra, and Gaurav Panwar, Security, privacy, and access control in Information-Centric Networking: A survey, *arXiv:1603.03409*, 2016.

[15] Samuel Wood, James Mathewson, Joshua Joy, Mark-Oliver Stehr, Minyoung Kim, Ashish Gehani, Mario Gerla, Hamid Sadjadpour, and J.J. Garcia-Luna-Aceves, ICEMAN: A system for efficient, robust and secure situational awareness at the network edge, *32nd IEEE Military Communications Conference*, 2013.

Confidential and Authenticated Communications in a Large Fixed-Wing UAV Swarm

Richard B. Thompson and Preetha Thulasiraman
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California 93940, USA
{rbthomps & pthulas1}@nps.edu

***Abstract* – Large Unmanned Aerial Vehicle (UAV) swarms are a nascent technology promising useful military and civilian solutions to logistical problems. Securing data communications within the swarm is essential to accomplishing swarm objectives. The Naval Postgraduate School has successfully demonstrated the launch, flight and landing of 50 UAVs. The communications architecture to support a UAV swarm is unique. This paper details the practical challenges of creating a secure communications channel in the swarm. The Advanced Encryption Standard (AES) was chosen as one of the encryption algorithms for testing as it is authorized by the National Security Agency (NSA). Various modes of AES, including Galois/Counter Mode (GCM) and Counter with Cipher Block Chaining Message Authentication Code (CCM), were analyzed within the swarm architecture. The impact of these authenticated encryption algorithms on network capacity and processor performance is presented. In addition to AES, ChaCha20-Poly1305, another type of authenticated encryption scheme was studied. It was found to be the better solution for securing the swarm if classified data is not being handled or created.**

I. Introduction

With the advent of ever cheaper hardware and further development of unmanned systems technology, the ability to field large swarms of unmanned aerial vehicles (UAVs) has become a reality. On August 27, 2015 the Naval Postgraduate School (NPS) set a world record by autonomously launching, flying and landing 50 UAVs concurrently [1]. The UAV swarm can be controlled by a single operator, guiding it to perform a specific behavior. Specific behaviors include area search, point intercept, ordered transit and mass ordered landing.

A. Research Motivations and Contributions

Currently there is no security architecture built into the UAV swarm at NPS. This stems from various factors, including cost and how UAV performance will be affected by computationally expensive encryption operations. However, the inability to solve the security problem has and will continue to have severe consequences for swarm deployment [2][3]. In the last several years, researchers have studied the various cyber security threats that UAV swarms face. These threats have been identified, along with extensive risk assessments [4][5]. While there is a consensus in the research community that UAV swarms are vulnerable to cybersecurity attacks, there has been little movement in identifying appropriate algorithms to facilitate a security architecture for UAV swarm communications. As large UAV swarms are just coming into existence, examples of security implementations in practice do not exist. This is especially true for large swarms. A swarm that is unable to operate securely is almost entirely useless in any military or civilian application.

In this paper, we focus on the impact of communication security on the swarm. This includes both encryption and authentication. Authenticated encryption (AE) is designed to simultaneously protect both a message's privacy and authenticity. For classified information communications, we study the Advanced Encryption Standard (AES). AES has been approved and adopted by the National Security Agency (NSA) as the official cryptographic module for the transmission of SECRET and TOP SECRET information. We implement four modes of AES: Counter with Cipher Block Chaining Message Authentication Code (CCM), Galois/Counter Mode (GCM), Synthetic Initialization Vector (SIV) and EAX. In addition, we also implemented ChaCha20-Poly1305, an unstandardized AE algorithm. This is used as a baseline for securing unclassified swarm communications. We present results that show the impact of these algorithms on network throughput and execution time.

It must be noted that this is a work in progress. The results we present in this paper reflect limited scale experiments run on the UAV software. We are in the process of building a full network simulation scenario using Network Simulator 3 (NS3) in which we will run experiments using a number of UAVs. These simulations will be concluded in the coming months.

The remainder of this paper is organized as follows: In Section II, we present how the swarm is designed and operates, with specific detail on the communications architecture. Section III contains a discussion on communication security, and how it is to be implemented in the swarm. Various AE techniques will be presented for consideration. Section IV presents network traffic analysis, including the impact of AE on network throughput. In Section V we perform a comparison of AE techniques on the ODroid processor. The percentage of time spent on cryptographic operations for each AE technique is calculated. In Section VI we conclude with a description of ongoing work.

II. Swarm System Architecture

A. Concept of Operations

The swarm architecture and its detailed operation can be found in [1] and are summarized here. UAVs launch at regular intervals of about 15 seconds and transit to a waiting area where they await a command from the swarm controller. The swarm controller has a set of predefined behaviors to choose from. After performing the set of defined behaviors, a command is sent to land. The swarm will then sort itself and land in an orderly fashion.

Each UAV was built from low-cost commercially available components. Fig 1. shows a picture of the NPS UAV.

Fig 1. Picture of NPS UAV, designed to be low cost yet capable. The figure is from [1].

The swarm communicates with an ALFA AWUS036NEH Long Range Wi-Fi Radio and processes information on an ODroid U3 computer with Ubuntu Linux 14.04. The ODroid computer has a Samsung Exynos4412 Prime Cortex-A9 Quad Core 1.7 GHz with 1 MB L2 cache. All software is coded in Python. Each UAV also possesses a remote control (RC) link and a 900MHz serial link, but are used for emergency and not necessary for swarm operation. In a secure setting they would be turned off.

B. Communications Architecture

The swarm communicates using 802.11n in ad hoc mode. The Wi-Fi radio has a single spatial stream operating on a 20 MHz channel, allowing for a maximum data rate of 72.2 Mbps. Each message is broadcast to each of the other UAVs. There is no expectation of privacy from any of the other UAVs. It is single hop, so there is no routing. All messages use UDP on top of IPv4, which by definition is connectionless. There is a small subset of messages that do receive acknowledgements, but that is built on top of UDP with a custom protocol. Fig. 2 shows the information pathways to and from a UAV.

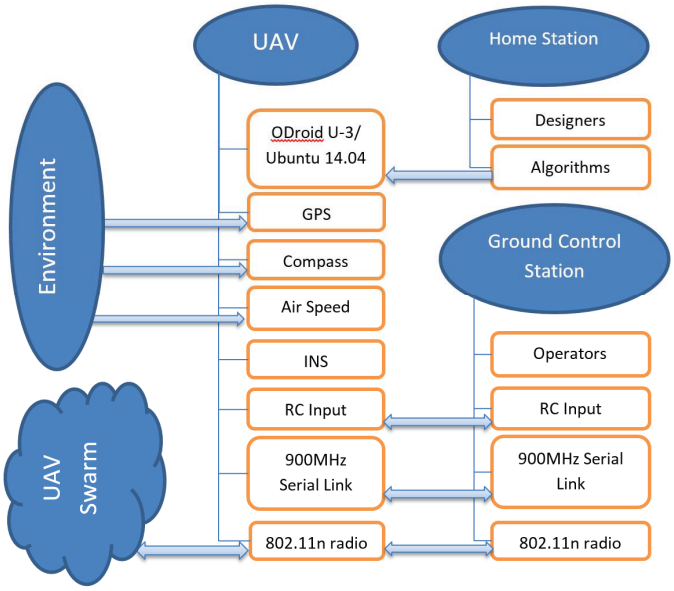


Fig 2. Illustration of UAV hardware configuration, detailing communication paths between entities. This figure was adapted from [3].

Increasing the reliability by using TCP, implementing a routing protocol, and communicating with direct links instead of broadcast would significantly increase latency and congestion on the network to the point that it becomes intolerable with a large swarm.

A side effect of the chosen architecture is the need to be tolerant of lossy communications. Messages must adhere to two principles. They must be stateless and idempotent. Stateless means a message cannot rely on a different message having been received. Idempotent means any message received may only change the state of the UAV once and only once. Duplicate messages do not further change the UAV state [6].

Table I lists the components that make up a message with typical lengths.

TABLE I
Message Contents

Section	Typical Length (bytes)
Preamble	15
802.11 Header	34
IPv4 Header	20
UDP Header	8
Autonomous Capability Suite (ACS) Header	16
Encryption Overhead	Varies by Algorithm
Payload	Varies by Behavior

Any security implementation should not fundamentally alter or impose changes on the defined communication structure. Doing so would limit the range of behaviors available and increase the complexity.

III. Communications Security Architecture

A. Communication Threats

Due to the fact that swarm communications are broadcast in every direction, any receiver within range of the UAV is able to collect communications. In order to prevent tactical information from being discovered, encryption is a necessary requirement. Due to the ease of message capture, and open nature of communications, the swarm is particularly vulnerable to replay attacks. As such, strong authentication is also an absolute necessity.

B. Cryptographic Methods

There are two broad categories for providing encryption and authentication in the swarm: symmetrically and asymmetrically. With asymmetric encryption, a public key is used to encrypt and a private key is used to decrypt. This would create a different communication channel between each UAV. In addition, to provide authentication, the system would require a secure central repository for public keys. That repository would either be a UAV or a ground station.

Asymmetric cryptography is not an option for swarm communications. It is a potential option for initially keying or for inflight rekeying, but for regular communications it would permanently impose an undesirable structure on the swarm.

Symmetric encryption is faster and works within the current swarm architecture. In this construct the same key would be preloaded before flight into each UAV. The UAV would encrypt and decrypt with this same key.

C. Authenticated Encryption Alternatives

There are only two options authorized for the encryption of classified data, Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) [7]. It is well established that AES is faster, more secure and efficient [8] and should be the chosen method for classified operations. Within AES, there are two AE modes available, GCM and Counter with CCM. GCM has been a part of the NSA Suite B since 2007 [7]. It has the benefit of being both efficient and parallelizable [9]. While both CCM and GCM provide secure solutions, GCM has a reputation for being faster [10].

SIV and EAX are highly specialized AE techniques designed for specific problems. Initialization Vectors (IVs) are used by cryptographic algorithms to ensure duplicate messages produce unique ciphertexts. For most algorithms, including CCM and GCM, improper selection of the IV has catastrophic results. SIV was designed to be tolerant of IV misuse, but by sacrificing speed [11].

EAX is another AES mode that was built to improve upon certain features of CCM. It can use arbitrary length IVs, and it does not need to know the message length in advance before beginning the algorithm. It also is a two pass mode (i.e., one pass to achieve privacy and the other pass for authenticity), and as such was not designed for speed [12]. SIV and EAX are currently under consideration for use in a classified environment and were included in the ODroid processor performance analysis [13].

The ChaCha20 algorithm for encryption and Poly1305 for authentication have become a popular alternative in industry for performing AE [14]. In 2014, Google replaced GCM on its Android phones with ChaCha20-Poly1305, believing it to be more secure and showing it to be significantly faster in software implementations [15]. ChaCha20-Poly1305 was designed to be fast in software on generic computer architectures by minimizing hardware intensive operations such as matrix multiplication [16]. While not approved for classified data, it was included in the analysis to provide both a baseline and an option for secure communications when the swarm is not performing classified operations.

The AES algorithms were implemented using the Python library PyCryptodomex 3.4.2. This library is written in Python where possible, except for the pieces critical to performance, which were written in C [17]. ChaCha20-Poly1305 was implemented using the Python library PyNaCl 1.0.1. The library is also written in Python but is a wrapper around the libsodium library, which is written in C.

PyCryptodomex allows for IV sizes of between 7 and 13 bytes for CCM. For CCM, the chosen IV size was 13 bytes and for all other AES based algorithms it was 16 bytes as recommended by [18]. The IV size for PyNaCl is required to be 24 bytes [19].

Message Authentication Codes (MACs) are bytes attached to each message used to verify the authenticity of a message. The MAC size for each algorithm was 16 bytes [18][19].

D. Appropriate Layer for Applying Security

In most applications there is a choice of whether to place security at the application layer or at the transport layer. Due to the connectionless nature of the swarm, it is possible to place security at an even lower level, just above the Physical Layer. Every message that is sent on the swarm channel is received by every other entity, with no expectation of routing. Therefore, the entire frame can be encrypted, including MAC addresses. This would prevent an adversary from gathering potentially critical message source information. Doing so, however, would require specialized hardware and/or software and is not possible with the current swarm configuration.

Security at the Transport Layer is also problematic. WPA2 with authentication on Ubuntu 14.04 is not supported for a connectionless ad hoc network without a central access point or centralized authentication server [20]. The ALFA Wi-Fi radio provides WPA2 encryption but only with 128 bit keys and lacks authentication that would work with the connectionless swarm configuration [21].

Security at the Application Layer is easily implementable, and provides privacy and authenticity. For the purposes of this study, security was implemented at the Application Layer on the ODroid processor. Implementation at the Application Layer will allow us to determine the impact of security on communications within the swarm.

IV. Effects of Authenticated Encryption on the Network Throughput

A. Simulation Description

To determine the effects of security on the network it is necessary to get a sense of the amount and type of traffic that is being passed.

Three instances of UAVs were created using multi-UAV simulation in the loop (SITL) software, commanded and flown with actual flight software as described in [22]. The messages passed between UAVs and ground station were the same as though the UAVs had been flying. For each message on the channel, the message type, size, time of transmission and sending UAV was recorded. A larger software swarm would have been preferred, but hardware limitations resulted in inaccurate results when the number of UAVs grew larger than three. Section IV-C discusses how the results obtained from a three UAV swarm are relevant to larger swarms.

The swarm has the following behaviors that it can perform:

- **Line Formation:** UAVs form into a line and fly to a designated location or flight pattern.
- **Swarm Search:** UAVs cooperatively search a specified area.
- **Greedy Shooter:** UAVs find the closest enemy UAV and tag it as being shot.
- **PN Interceptor:** Command given to one UAV to intercept another.
- **Eager Altitude Sort:** UAVs are sorted by altitude. Missing information is requested from other UAVs. Responses to requests are given about itself and other missing UAVs.
- **Lazy Altitude Sort:** Similar to Eager Altitude Sort except only missing information about itself is broadcast.
- **Independent Transit:** All UAVs in a subswarm transit separately to a geographic position.
- **Sequence Land:** UAVs land in an orderly fashion.

A software application called *Swarm Commander* is used to command the swarm to perform these behaviors during actual operations. In the simulation, *Swarm Commander* was used to execute each of the behaviors in the order presented above. When parameters were required, the default values were selected.

B. Simulation Results

There was an average of 13.02 messages per UAV transmitted in any given second. The average number of messages per second for a three UAV swarm was 39.07, with a standard deviation of 6.618 and a high of 52. The average unencrypted message size was 141.61 bytes with a standard deviation of 12.54. Fig. 3 illustrates the throughput of the 3 UAV swarm over the time of the experiment. As can be seen, throughput is fairly constant, and it is difficult to distinguish when certain behaviors are occurring.

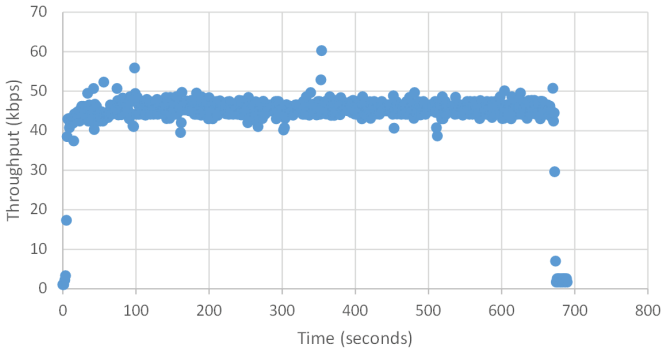


Fig 3. Total throughput on a 3 UAV Swarm channel without AE

With a 3 UAV swarm, Table II shows the breakdown of occurrence by message type.

TABLE II
Occurrence of Message Type

Message Type	Occurrence
Pose	74.38%
Flight Status	14.38%
Heartbeat	9.48%
Other	1.76%

Traffic is dominated by just three types of messages: Pose, Flight Status and Heartbeat. These messages are used by the UAVs to update each other and the ground station with telemetry and health information. They are sent out at regular intervals regardless of swarm size.

Fig. 4 shows the same throughput as Fig. 3, but with Pose, Flight Status and Heartbeat messages removed. Throughput is much less, and areas where behaviors occur are somewhat distinguishable.

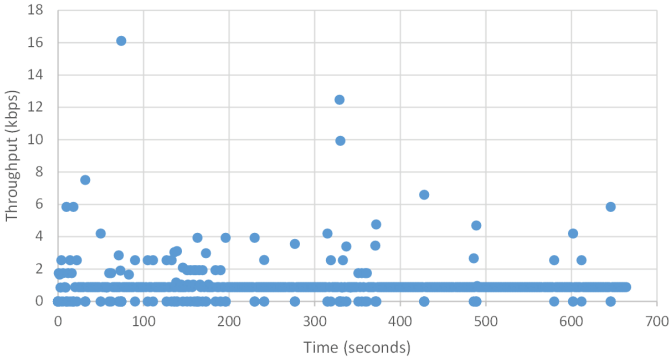


Fig 4. Total throughput on a 3 UAV Swarm channel with Pose, Flight Status and Heartbeat messages removed

Table III shows the effects of AE on swarm throughput. ChaCha20-Poly1305 had the greatest overhead. Note that the bytes per message overhead incurred by each algorithm is constant, regardless of message length. Thus as message length increases, the overhead as a percentage of message length due to cryptographic operations decreases.

TABLE III
EFFECTS OF CRYPTOGRAPHIC OPERATIONS ON THROUGHPUT

Cryptographic Algorithm	Average Throughput (kbps)	Maximum Throughput (kbps)	Average Overhead Incurred (%)
None	44.23	60.28	0
CCM	53.29	71.64	20.4
GCM	54.29	72.82	22.5
ChaCha20-Poly1305	56.73	76.96	28.2

C. *Extending Results to Larger Swarms*

Knowing how these results apply to larger swarm sizes is essential for an accurate understanding of the effects of AE on network throughput. To determine the effects, it is necessary to analyze each message and determine how it depends on the size of the swarm. There are two ways in which a message can be dependent on swarm size: length dependent and frequency dependent. A message is length dependent if a particular message changes length as a function of swarm size. A message is frequency dependent if how often a message is sent depends on swarm size.

With complete message independence of swarm size, growth rate of network traffic is $O(n)$ where n is the number of UAVs in the swarm. In a behavior where messages grow by a constant amount with each additional UAV, growth rate of network traffic would be $O(n^2)$. In a behavior where the frequency of messages increases at a constant rate with each additional UAV, the growth rate of network traffic would also be $O(n^2)$. A behavior where both message frequency and length grow by a constant amount with each additional UAV would be $O(n^3)$.

If all messages are independent of swarm size, Fig. 5 shows how average throughput will increase as the swarm size grows. In this scenario, even in a 100 UAV swarm, and 30% cryptographic overhead, the swarm does not approach the 72.2 Mbps throughput ceiling.

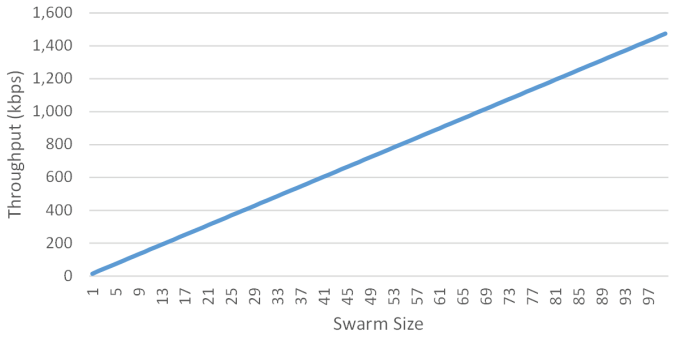


Fig 5. Average total throughput as a function of swarm size where messages do not depend on swarm size (no AE)

If an additional packet is sent by each additional UAV per second, and each of those messages grows by 10 bytes for each UAV, we get the results seen in Fig. 6. With larger swarms the channel becomes overwhelmed, even without cryptographic overhead.

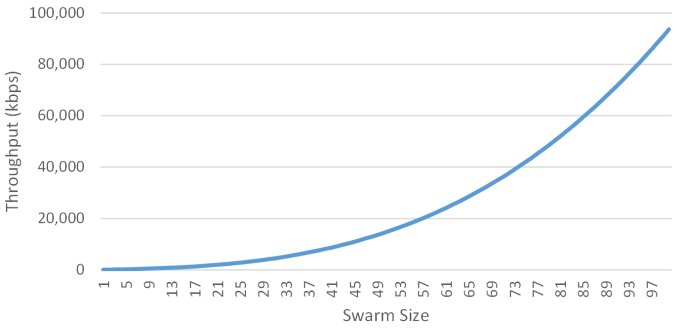


Fig 6. Average total throughput as a function of swarm size where messages have both length and frequency dependence on swarm size (no AE)

Fig. 5 and Fig. 6 give a reasonable lower and upper bound on network traffic. To determine the likelihood of the two scenarios, a closer examination of each behavior was undertaken. Table IV shows how each behavior affects message dependence on swarm size.

TABLE IV
MESSAGE DEPENDENCE ON SWARM SIZE

Behavior/ Message Type	Length Dependence	Frequency Dependence
Flight Status	No	No
Pose	No	No
Heartbeat	No	No
Eager Altitude Sort	Yes	Yes
Greedy Shooter	No	Yes
Independent Transit	No	No
Lazy Altitude Sort	No	Yes
Line Formation	Yes	Yes
PN Interceptor	No	No
Sequence Landing	No	No
Swarm Search	No	Yes

Flight Status, Pose and Heartbeat messages are sent out at frequencies of 10Hz, 2Hz and 2Hz respectively, regardless of swarm size.

The Eager Altitude Sort and Line Formation behaviors make use of a consensus sort algorithm. This algorithm requests a message from each UAV from which it lacks information. A larger swarm increases the likelihood of missing information. In addition, response messages from any UAV includes information from any other UAV whose information was also requested, thus message lengths will also increase. Designers recognized this and limited the frequency of the messages to 4Hz. Thus, the frequency dependence has an upper bound.

In the Greedy Shooter behavior, the frequency dependence is very weak and probably undetectable. As the swarms grow, the density of UAVs also grows creating more shooting opportunities, and thus more kill reports.

In the Lazy Altitude Sort behavior, the consensus sort algorithm is also used with the difference being any UAV whose information is requested will only respond with information about itself. Thus there is no length dependence. The frequency dependence is again limited to 4Hz.

In the Swarm Search behavior, a lead UAV is designated at the commencement of the behavior. The lead UAV proceeds to assign search areas to each of the other UAVs in

its subswarm. There is frequency dependence, but only from the point of view of the lead UAV.

The worst case for network traffic would be during a behavior using the consensus sort algorithm where each UAV requested information from every other UAV. If an additional 4 messages were sent from each UAV per second, each UAV grew 10 bytes for each UAV in the swarm, and a 30% overhead was added on for worst case cryptography, the resulting throughput would look similar to that shown in Fig. 7.

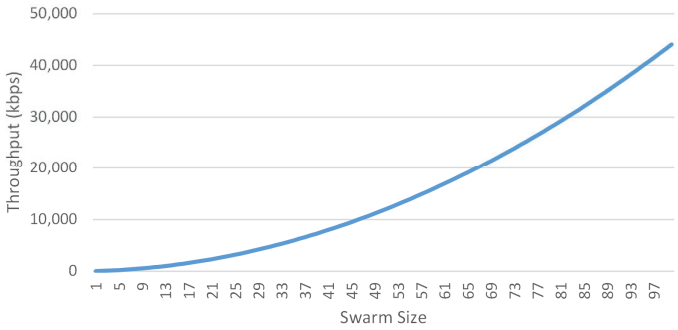


Fig 7. Expected throughput of a worst case scenario behavior, with worst case cryptographic overhead included

Even with the worst case behavior, and using the largest cryptographic overhead, the swarm communications channel is left with some operational margin.

IV. Effects of Authenticated Encryption on the ODroid Computer

A. Experiment Description

To completely understand the impact of AE on the swarm, it is necessary to determine if the ODroid processor is capable of sustaining the cryptographic overhead incurred. To make this determination, GCM, CCM, SIV, EAX and ChaCha20-Poly1305 algorithms were implemented and executed on the ODroid hardware.

AE was performed on messages of sizes ranging from 8 to 32,768 bytes and then decrypted and authenticated. For each message size, this step was repeated 10,000 times and averaged. Each message was randomized on each pass to create as cold a cache as possible.

The experiment was repeated with key sizes of 128, 192 and 256 bits. ChaCha20-Poly1305 and SIV do not provide functionality for key sizes of 128 and 192 bits.

B. Experiment Results

Figs 8, 9 and 10 show the results for key sizes of 128, 192, and 256 bits, respectively.

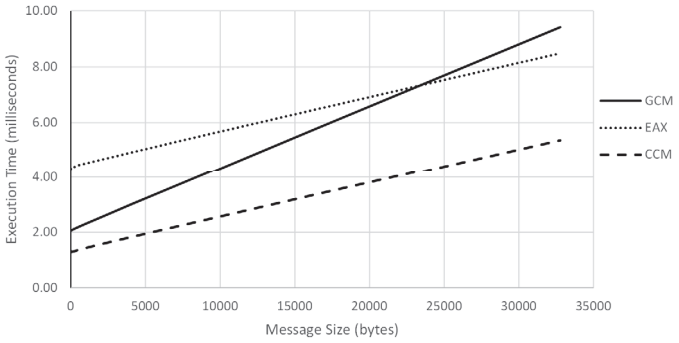


Fig 8. ODroid performance while executing cryptographic operations with 128 bit key sizes

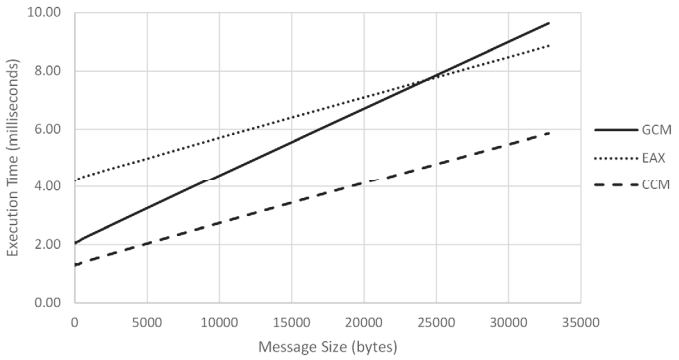


Fig 9. ODroid performance while executing cryptographic operations with 192 bit key sizes

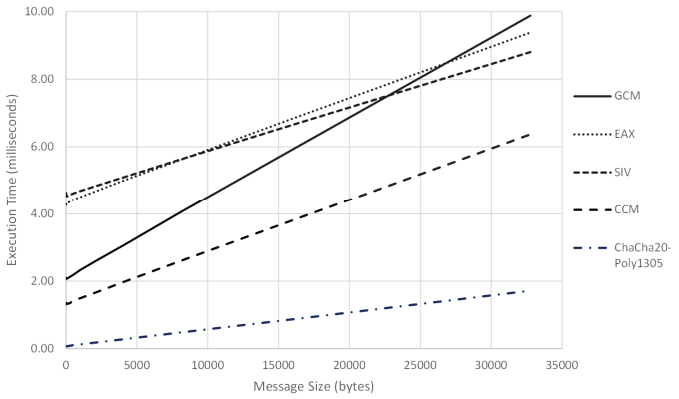


Fig 10. ODroid performance while executing cryptographic operations with 256 bit key sizes

As was expected, the best performance for any key size was ChaCha20-Poly1305. EAX and SIV had the poorest overall results. This poor performance was expected given their specialized nature.

According to the results, CCM outperformed GCM. As stated earlier, the biggest advantage of GCM is its ability to be parallelizable. GCM did not perform better than CCM in this situation for the following reason: GCM performs best when software is tailored to the hardware, making use of parallel processors [9][23]. It does not appear that the PyCryptodomex library makes full use of the parallelizable nature of GCM, and it certainly was not designed specifically

for the ODroid computer. These results are consistent with the results found in [12].

Another interesting result is how little key size affects speed of execution, especially with smaller messages. The average length of an unencrypted message was 141.61 bytes. Table V lists the average time (in milliseconds) to execute each iteration of an average length message.

TABLE V
CRYPTOGRAPHIC TIMES FOR AVERAGE SIZED MESSAGE IN MILLISECONDS

	128 Bit Key	192 Bit Key	256 Bit Key
CCM	1.30	1.31	1.32
ChaCha20-Poly1305			.0781
EAX	4.40	4.28	4.33
GCM	2.10	2.08	2.09
SIV			4.54

C. ODroid Performance under Various Network Loads

From the channel throughput analysis in Section IV, we can predict the burden that cryptography will place on the ODroid computer. Given that the average unencrypted message size was 141.61 bytes, and the average number of messages per second per UAV was 13.02, an average cryptographic load for a given swarm size on the ODroid computer can be estimated.

Assuming messages do not exhibit any dependence on swarm size (as laid out in Fig. 5) and assuming an average case throughput of 13.02 messages per UAV per second, the percentage of time out of each second spent conducting cryptographic operations is displayed in Fig. 11.

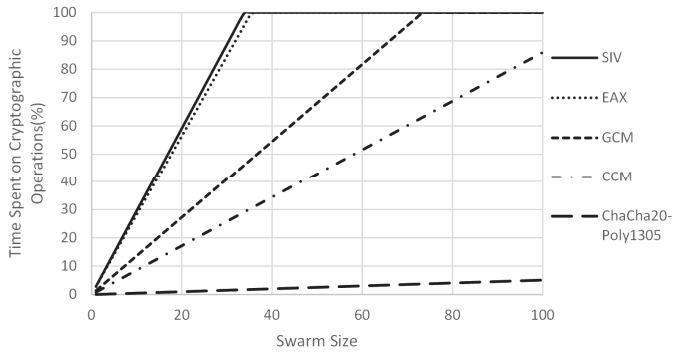


Fig 11. Average case percentage of time spent on cryptographic operations with 256 bit key

SIV and EAX would not be able to support a 50 UAV swarm, as the ODroid would be spending 100% of its time on cryptographic operations and still not be able to keep up with the traffic load. GCM and CCM might be able to manage an average case load in a 50 UAV swarm, but it would be consuming a significant amount of the processing capacity, leaving little to other processes. Fig. 11 shows the worst case scenario as defined by Fig 6. Using a smaller key size does not improve performance by any meaningful amount.

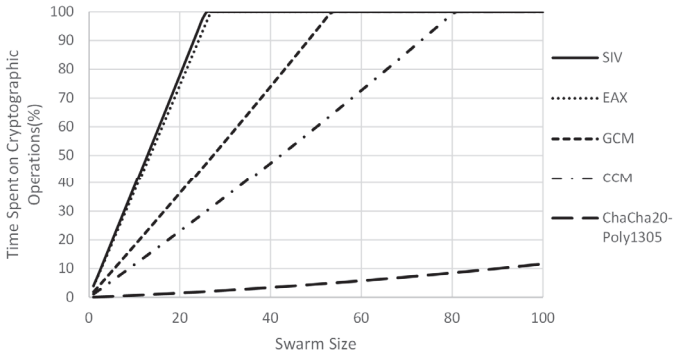


Fig 12. Worst case percentage of time spent on cryptographic operations with 256 bit key

Clearly SIV, EAX, GCM and CCM are not an option. It does appear that ChCha20-Poly1305 would be successful in providing AE for the swarm.

D. Mitigations for Classified Information

Currently the swarm at NPS is used for academic purposes. It does not gather or create classified information and thus does not require the use of an AES based algorithm. In the event that classified information was gathered, the following mitigations could be used to enable the use of either GCM or CCM:

- Upgrade to a more powerful processor.
- Use an Application Specific Integrated Circuit (ASIC).
- Tailor the AES algorithm to the ODroid processor.
- Only use AE on command data.

Performing AE only on command data would give access to information about the state of the swarm and its current location, and might not be tolerable. It also opens the risk of an adversary planting false telemetry data and surreptitiously changing the state of the swarm. It would, however, prevent the taking over of the swarm by the sending of direct commands. Fig.13 shows the impact on swarm performance in the worst case network channel scenario presented in Fig. 7 when only command data is encrypted.

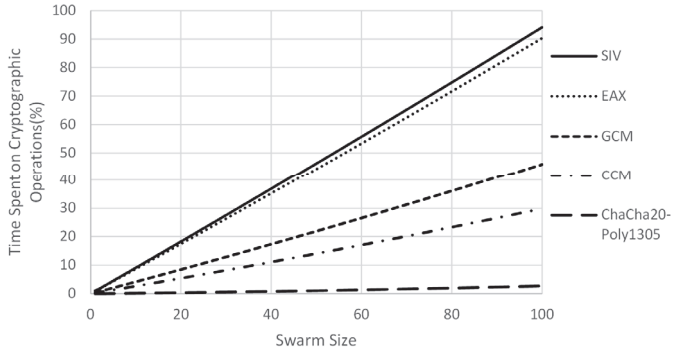


Fig 13. Worst case percentage of time spent on cryptographic operations only on command data with 256 bit key

With this mitigation, SIV and EAX still fall short of acceptability on large swarms. GCM and CCM perform

tolerably, and ChaCha20-Poly1305 again proves the superior method.

V. Conclusions and Ongoing Work

In the current swarm configuration and architecture, analysis indicates that performing AE with GCM, CCM, SIV or EAX is not feasible. GCM and CCM are only feasible by accepting risk. The best choice by far is ChaCha20-Poly1305, and it should be the AE choice in any scenario where classified data is not being handled or created.

Future work is needed to fully and accurately determine the true effect of AE on swarm communications. The following course of action is either planned or in progress:

- An NS3 model of the channel is being built. This will allow a more in depth understanding of how much capacity is actually available. It will be able to model how distance, packet collision avoidance and modulation affect throughput.
- An experiment has been designed to bombard an ODroid with typical traffic through an 802.11n connection. The amount of traffic will be increased until a saturation point is discovered. This will help determine how much processor margin is available for cryptographic operations.
- Power consumption due to AE could prove significant. An experiment is planned to measure the impact AE has on battery life.

Operating a swarm without communication security poses too great a risk in almost any operation. Completion of this work is essential for the ultimate success of UAV swarms in any operational environment.

Acknowledgement

This work was funded and sponsored by the Consortium for Robotics and Unmanned Systems Education and Research (CRUSER)-Office of Naval Research (ONR).

References

[1] T. H. Chung et al., "Live-Fly, Large-Scale Field Experimentation for Large Numbers of Fixed-Wing UAVs," in *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, 2016.

[2] M.S. Faughnan et al., "Risk Analysis of Unmanned Aerial Vehicle Hijacking and Methods of its Detection," in *Proc. of IEEE Systems and Information Engineering Design Symposium*, 2013, pp. 145-150.

[3] K. Hartmann and C. Steup, "The Vulnerability of UAVs to Cyber Attacks-An Approach to Risk Assessments," in *Proc. of IEEE 5th International Conference on Cyber Conflict*, 2013, pp. 78-100.

[4] K. Mansfield et al., "Unmanned Aerial Vehicle Smart Device Ground Control Station Cyber Security Threat Model," in *Proc. of IEEE International Conference on Technologies for Homeland Security*, 2013, pp. 722-728.

[5] A. Javaid, W. Sun and M. Alam, "UAVSim: A Simulation Testbed for Unmanned Aerial Vehicle Network Cyber Security Analysis," in *Proc. of IEEE Globecom Workshop-Wireless Networking and Control for Unmanned Autonomous Vehicles*, 2013, pp.1432-1436.

[6] M. Clement, private communication, Dec. 2015.

[7] A. Vassilev, "Annex A: Approved Security Functions for FIPS PUB 140-2, Security Requirements for Cryptographic Modules," National Institute of Standards and Technology, Gaithersburg, Maryland, 2016.

[8] H. O. Alanazi, et al., "New Comparative Study between DES, 3DES and AES within Nine Factors," *Journal of Computing*, vol. 2, no. 3, pp. 152-157, 2010.

[9] M. Dworkin, "NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," National Institute of Standards and Technology, Gaithersburg, Maryland, 2007.

[10] D. A. McGrew and J. Viega, "The Security and Performance of Galois/Counter Mode (GCM) of Operation," in *Progress in Cryptology - INDOCRYPT 2004*, Chennai, India, 2005.

[11] P. Rogaway and T. A. Shrimpton, "A Provable-Security Treatment of the Key-Wrap Problem." in *Advances in Cryptology EUROCRYPT2006*, St. Petersburg, Russia. 2006, vol. 4004 of LNCS, pp.373-390.

[12] Švenda, P. (2016) "Basic comparison of Modes for Authenticated-Encryption (IAPM, XCBC, OCB, CCM, EAX, CWC, GCM, PCFB, CS)." [Online]. Available: https://www.fi.muni.cz/~xsvenda/docs/AE_comparison_ipics04.pdf

[13] Modes Development. (2016, Mar. 24). National Institute of Standards and Technology. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html#01

[14] N. Sullivan. (2015, Feb. 23). Do the ChaCha: better mobile performance with cryptography [Online]. Available: https:// blog.cloudflare.com/do-the-chacha-better-mobile-performance-with-cryptography/

[15] E. Bursztein. (2014, Apr. 24). Speeding up and strengthening HTTPS connections for Chrome on Android [Online]. Available: https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html

[16] D. Bernstein. "The Salsa20 family of stream ciphers" in *New Stream Cipher Designs*, 2008, pp. 84–97.

[17] pycryptodomex 3.4.1 (n.d.) Cryptographic Library for Python. Python Package Index. [Online]. Available: https://pypi.python.org/pypi/pycryptodomex/3.4.2. Accessed Aug. 8, 2016.

[18] Module AES. (2016 Feb. 7) Pycryptodome.org. [Online]. Available: http://legrandin.github.io/pycryptodome/Doc/3.4/Crypto.Cipher.AES-module.html#MODE_CCM

[19] D. J. Bernstein et al. "The security impact of a new cryptographic library." in *International Conference on Cryptology and Information Security in Latin America*, Santiago, Chile, 2012, pp.159-176.

[20] WifiDocs/Adhoc. (2011, May 23). Ubuntu Documentation. [Online]. Available: https://help.ubuntu.com/community/WifiDocs/Adhoc

[21] AWUS036NEH. (n.d.) ALFA Network. [Online]. Available: https://www.alfa.com.tw/products_show.php?pc=34&ps=22. Accessed Aug. 11, 2016.

[22] M. A. Day et al., "Multi-UAV Software Systems and Simulation Architecture," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Denver, Colorado, 2015.

[23] K. Thompson, "Bad APIs Cause Bugs," in *Zero Bugs and Program Faster*, 1st ed. Seattle: Kate Thompson Books, 2015, ch. 29, pp. 61-64.

A Back-end Offload Architecture for Security of Resource-constrained Networks

Jiyong Han and Daeyoung Kim
School of Computing
Korea Advanced Institute of Science and Technology, KAIST
291 Daehak-ro, Yuseong-gu, Daejeon, Korea
Email: {jyhanzz, kimd}@kaist.ac.kr

***Abstract*—Recent years have seen the development of successful internet of things (IoT) technologies based on an IP-enabled 6LoWPAN, which enables the extensive message exchange of information generated from various applications such as health-care, smart home, and factory automation. Because IoT devices are closely related to the human life, they generally handle critical data which must be protected from a malicious adversary. However, a majority of IoT devices are resource-constrained in terms of memory and computational ability, so that they cannot provide heavy security protocols and authentication. To protect connections over constrained networks, we introduce a back-end offload architecture which offloads the processing of a security protocol to a specified back-end offloader. The offloader assists constrained devices back-end by handling the packets of handshake procedure and encrypted application data. The load balancing of offloaders and the protection of offload messages are also provided in the design. The proposed architecture allows an extremely constrained device to establish a secure session by utilizing high-level authentications such as public key infrastructure or certificate, without the burden of deploying heavy security modules. This research would be advantageous for incompetent nodes to support security and reduce cost at the same time.**

I. Introduction

The proliferation of internet of things (IoT) technologies enables billions of objects to be connected online and to exchange information over an IP-enabled 6LoWPAN network which serves various applications such as health-care, smart home, factory automation, even military. IoT devices are expected to be closely related to the human life, which handles critical data generated from such services. Thus the security of resource-constrained networks should be counted as an essential requirement, notwithstanding the fact that a majority of devices participating the networks are mostly not able to handle security connections due to their limited memory and computation resources.

In order to support security to constrained devices, several studies have explored lightweight solutions, centering around datagram transport layer security (DTLS) [1] which is the de facto security protocol for IoT. However, DTLS was originally designed to protect traditional Internet services for resource-rich devices. Thus, deploying DTLS as is in constrained devices incurs considerable overheads. To overcome this problem, recent studies have focused on the new designs of authentication architecture [2][3][4], security protocol optimization [5][6], and protocol profiling for constrained networks [7][8].

978-1-5090-3216-7/16/$31.00

Although much work has been done to date, more practical and network-wide solutions need to be further studied to enable the reliable establishment of secure connections, even if a network device is extremely constrained to have a security protocol inside. In addition, constrained devices should be able to employ high-level authentications such as public key infrastructure (PKI) and certificate.

The purpose of this research is to provide security to constrained nodes even if they cannot run heavy security protocols with their own specification. To this end, we introduce a back-end offload architecture which offloads the processing of a security protocol to a specified *back-end offloader (BeO)*. It relieves constrained devices from managing security protocols directly and also from authentication. The proposed BeO assists a constrained device back-end by handling security contexts instead of it, from handshake negotiation to the encryption and decryption of application data. The communication between a constrained device and BeO is protected safely with a symmetry secret key generated. Trust association manager (TAM) controls BeOs and their allocations based on the proposed load balancing scheme. With the aids of BeO, heavy authentications can also be provided. This architecture eventually allows constrained devices to not only establish a simple secure connection but also be provided high-level authentications and the load balancing of security processing.

Despite the difficulties on the use of security protocols in constrained networks, security should be provided to protect critical information related to our life. The proposed back-end offload architecture would provide an alternative to the problem of security supports in resource-constrained networks.

The rest of the paper is organized as follows: We introduce the background in section II and explain a back-end offload architecture in section III. Section IV discusses security considerations and finally, section V concludes the paper.

II. Background

We here provide a brief explanation of the security protocol and limitations on providing security in constrained networks.

A. Resource-constrained Networks

IoT generally refers the network of heterogeneous nodes from constrained devices such as small sensors which mainly

focus on single task (e.g. monitoring) to comparatively unconstrained devices like smartphones and tablets. Compared to traditional network nodes, resource-constrained nodes have difficulties to provide a seamless communication with other nodes of plenty resources. The IETF standard document [9] specifies the terminology of a resource-constrained network and the resource standard in terms of memory and energy limitation. It defines the constrained node (CN) and the constrained-node networks (CNN). Table I represents the memory constraints of CNs. Note that CNs have maximum 250 kilobytes of code memory (e.g. Flash) and 50 kilobytes of read and write memory (e.g. RAM). This paper mainly aims at C0 and C1 devices which cannot deploy a security protocol due to the lack of resources.

B. Datagram Transport Layer Security (DTLS)

DTLS is a security protocol for UDP datagram which succeeds the major characteristics from traditional transport layer security (TLS) [10]. This protocol is originally designed to protect web application services; however, it is now actively used for IoT devices which mainly have constrained resources. Handshake procedure is similar with the one of TLS as shown in figure 1. To provide the same features of the connection-based protocol (TLS) to the connection-less protocol (DTLS), DTLS explicitly uses a sequence number field. Several optional messages (with a star mark) in handshake are used to negotiate ciphersuite parameters in PKI and certificate based authentication.

C. Limitations on Providing Security

Due to the limitations of CNs explained in II-A, they usually have a little space for upper-layer applications even though they load a lightweight operating system such as Contiki OS [1] or TinyOS[2]. The OS required around 100 kilobytes of ROM and 15 kilobytes of RAM in our implementation on the 16-bit MSP430 micro-controller platform. As for DTLS, it incurred additional ROM and RAM overheads about 16 kilobytes and 5 kilobytes [11]. When it comes to the use of software-based cryptography modules, about 20 kilobytes of ROM is additionally consumed. PKI and certificate based authentications also require additional overhead. For the PKI library, 16 kilobytes or more ROM is needed [12] and for the certificate based authentication, transmission and parsing become costly jobs. Because of these constraints, C0, C1 even C2 devices cannot deploy security protocols easily. Low-class devices cannot save the space for security protocols, and even high-class level devices cannot run protocols without considering performance degradation. However, it is still important that C0 and C1 devices should be able to provide security.

III. Architecture

In this chapter, we propose the overall design of the back-end offload architecture. We first introduce the overview, then explain the details.

[1]Contiki: [Online]. Available: http://www.contiki-os.org
[2][Online]. Available: http://www.tinyos.net

TABLE I
Classes of Constrained Devices

Name	Code Size (e.g., Flash)	Data Size(e.g., RAM)
Class 0, C0	<<100 KiB	<<10 KiB
Class 1, C1	~ 100 KiB	~ 10 KiB
Class 2, C2	~ 250 KiB	~ 50 KiB

* KiB = 1024 bytes

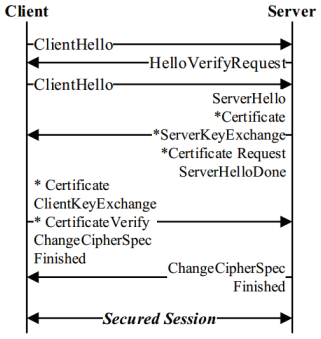


Fig. 1. DTLS handshake procedure. Messages marked with * are optional.

A. Overview

The communication scenario can be summarized as follows.

- A CN (e.g. level C0, C1) communicates with a remote end-point with an IP address (both IPv4 and IPv6).
- A remote end-point can be any nodes either constrained or not, and located either inside or outside local network.
- A remote end-point can request a CN to obtain information (e.g. value of deployed sensors) by using application protocols such as CoAP [13] or LWM2M [14].
- A CN can report its status to a remote end-point by itself (likely to be periodic).

The proposed architecture shown in figure 2 provides a secure communication in above scenario even when CNs cannot provide security. The proposed architecture enables DTLS by back-end offloading, so that a CN can pretend to be a powerful host with a security protocol, without the real implementation in it. A CN has no ability to interpret secure packets (i.e., DTLS packets); thus, it forwards packets to a *back-end offloader (BeO)* which has ability. A BeO, assigned to the CN beforehand, handles these packets instead of the CN and forwards the result back. The communication data between BeO and the CN is protected by a symmetry key generated using a nonce provided from the trust association manager (TAM) beforehand. With this procedure, the CN can finally establish a secure connection with a remote host whenever it cannot deploy a security protocol inside it.

Note that TAM and BeOs are also in the same network with CNs. They have the IEEE 802.15.4 network stacks and communicate with CNs. Our architecture allows multiple BeOs in one CNN to guarantee many CNs can be provided the security feature at the same time. Offloading requests of many CNs are distributed to multiple BeOs and it is performed by

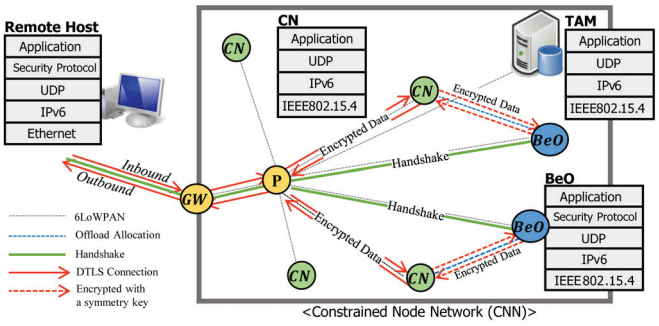


Fig. 2. Back-end offload Architecture for Security. (CN) is a constrained node, (BeO) is a back-end offloader, (P) is a pan coordinator, and (GW) is a gateway. Two CNs in the middle have finished the handshake procedure and are now offloading the application data. All transactions are encrypted.

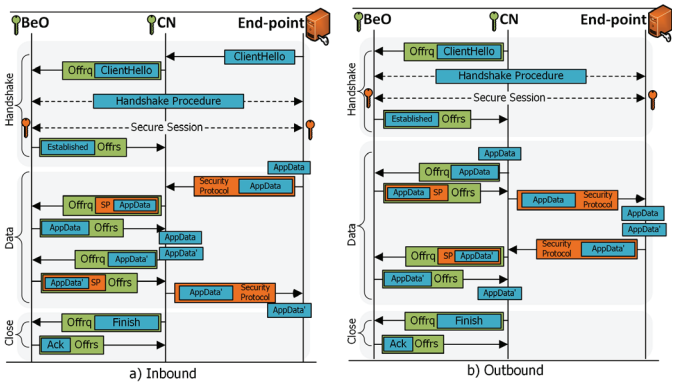


Fig. 3. Back-end Offloading Procedure. a) inbound and b) outbound. *Offrq* means offload request, and *Offrs* is offload response. *SP* stands for security protocol. All transactions are encrypted.

the proposed load balancing scheme. For overall supports, we introduce a trust association manager and discuss its roles in the next subsection.

B. Trust Association Manager

To manage BeOs effectively, we propose the trust association manager (TAM) which in charge of providing secret keys between BeOs and CNs. TAM is under the direct supervision of a local CNN administrator and should be deployed at least one in a CNN. TAM is specially required to have sufficient resources to be able to manage secret keys and the load balancing of BeOs.

When a CN which needs security assists joins a network, it automatically registers itself to TAM, then TAM enrolls the node id to a list of joined CNs. For this procedure, it is required that the CN and TAM should share a master key to protect messages between them. This key can be imprinted from factory or securely deployed [15] from TAM. After this, TAM finds a suitable BeO for the CN, and assigns it by transmitting a nonce. This nonce is used for a symmetry key generation between the CN and BeO. Using this key, the CN offloads its security context to BeO in a secure manner and can be provided protected information from a remote end-point. In case that a CN cannot start a bootstrapping function, a local CNN administrator can register and assign a BeO manually.

C. Back-end Offloading Procedure

In our back-end offloading architecture, we assume that there should be at least one BeO in a local CNN and BeO should not be resource-constrained. For the handling of offload packets, a CN and BeO employ a tiny application which takes charge of bootstrapping and exchanging offload request and response. Note that all offload request and response data are encrypted by a symmetry key generated beforehand to guarantee security between a CN and BeO.

There are two network cases how a CN has a secure session between a remote end-point, *inbound* and *outbound*. In the inbound case, a remote end-point first asks a CN to retrieve the information of sensor values. In the case of the outbound, a CN reports its status (mainly periodically) to a remote end-point by itself. As shown in figure 3-a), when a CN receives a *ClientHello* from an end-point, it firstly distinguishes the packet whether it is from a security protocol by inspecting the incoming port. Then, the relevant packet is packed in a offload request message (Offrq) and forwarded to BeO. BeO establishes a DTLS connection with the end-point using an IP address of the CN and let the CN knows the result with the *Established* message within a offload response message (Offrs). On the other hand, a CN initiates a secure session with a remote end-point as shown in figure 3-b). It first sends the Offrq message to BeO with a flag that indicates outbound, and encloses an address information of the remote end-point. Then BeO establishes a DTLS connection instead of the CN, and notifies the handshake finish to the CN.

After a secure session is established, both communication parties can exchange their application data safely under the protection of a security protocol. Similar to handshake offloading, when a CN receives an encrypted application data, it forwards it to the allocated BeO and waits for the decrypted data. Likewise, when a CN wants to send an application data, it transmits the data to BeO in the Offrq message. Then BeO constructs a security protocol packet and forwards the packet to the CN in the Offrs message. The CN simply removes the Offrs and sends the protocol packet to the end-point.

When a CN or an end-point explicitly notifies a session termination, the CN lets BeO knows the termination by transmitting *Finish* message in the Offrq. When there is no explicit session termination, BeO waits until the session timeout reaches so that it can withdraw resources back and avoid security dangers.

Since the communication from a CN to a remote end-point is secured thoroughly, an adversary cannot eavesdrop plaintexts between them. During whole procedure, a remote end-point has no knowledge of BeO and only can assume that a CN is directly communicating with it over a security protocol (e.g. DTLS). Also a CN can initiate a secure session without notifying the existence of BeO to the outside of a network. Note that all offload-related transactions are encrypted by a

generated symmetry key.

D. End-point Authentication

CNs follow ciphersuites which BeO supports. Thus they can employ high-level authentications such as public key infrastructure (PKI) or certificate by letting BeO handles them instead. Pre-shared key (PSK) is the most lightweight and common authentication; however, a shared secret key has to be provided to both entities beforehand, which restricts the access of various end-points. Moreover, a shared key is used only for one client-server pair; thus, it is not efficient in the case of many pairs.

Meanwhile, PKI or certificate based authentication is mainly considered for resource-rich devices due to big implementation size and high requirements for processing, despite its convenience of key provision and guaranteed secret (e.g. perfect forward secrecy (PFS)). In the proposed architecture, BeO can assist a CN to employ PKI or certificate based authentication. Thus a CN can be provided upgraded security properties even if it has limited capability.

E. Initiating Offloaders

We have pointed that BeO should be deployed at least one in a CNN. Any nodes which have capability of managing many secure sessions at the same time can be considered to be BeO. A BeO candidate bootstraps by sending its node id and the device specification to TAM. TAM identifies the node and records it in the list of BeO candidates. Similar to CNs, a master key between TAM is also provided to BeOs. The device information sent to TAM is used for the decision of the qualified offloader for a CNN. The priority of metrics to become a BeO can also be adjusted. Until when TAM determines that there are sufficient BeOs in a network, it keeps selecting BeOs for the network. A network administrator can also designate a BeO node manually without using the function of TAM. When there are multiple candidates, TAM can initiate multiple offloaders in one CNN. It mainly occurs when there are many offload demands which one BeO cannot handle connections reliably (e.g. low response time).

We assume that a pan coordinator does not act as BeO even though there is a benefit which lightens the burden of deploying another resourceful node (BeO) in the network. A pan coordinator already has many jobs as a main controller, such as device association, dissociation, assigning addresses, and synchronization; thus, it is hard to entrust security operations to a coordinator. Moreover, it is vulnerable to an outside attacks because a pan coordinator is more likely to be shown. On the other hand, BeOs are basically hidden devices which never disclose their network addresses to outside.

F. Load Balancing Offloaders

The designation of BeOs to CNs can be changed dynamically by the load balancer module of TAM when the previously allocated BeO is found to be out of resources. The change of network topology caused by the mobility of CNs is also considered as a metric (hop distance). For this, TAM continuously monitors BeOs and determines whether they are available or not. The load of a BeO is computed as line 3 in algorithm 1. Hop distance between the BeO_k and each connected CN_i is represented as h_i. The number of connected CNs with the BeO_k is $numCN$, and the traffic load of the CN_i is represented as $load_i$. A weight factor γ_i ($\forall\gamma \in [0,1]$) is multiplied to h_i to reflect the network condition between the BeO_k and each CN_i in real-time. If the quality of a network is good (in terms of low delay, high packet delivery rate, etc.), γ is set low. If not, γ should be set high to reflect bad condition. Each γ is computed by BeO using various condition parameters. For example, if a network is sensitive to packet delivery rate (PDR), PDR parameter should primarily be reflected to the computation of γ. Based on computed loads of BeOs, TAM derives the average load as line 5 in algorithm 1 and determines whether the BeO is in *idle, normal*, or *overload* status. The balancer computes two load boundaries, low and high, which are derived using pre-determined parameters α and β beforehand by administrator. Then the BeO is flagged as shown in line 6-14 in algorithm 1.

Algorithm 1 The Load Balancing of BeOs

```
1:  procedure BALANCER(numBeO, α, β)
2:      for every BeOs k do
3:          load_BeOk ← (Σ_{i=1}^{numCN} γ_i h_i × load_i) / capacity_BeOk
4:      end for
5:      load_avg ← (Σ_{k=1}^{numBeO} load_k) / numBeO
6:      for every BeOs k do
7:          if load_BeOk ≤ α × load_avg then
8:              flag_k ← 0                          ▷ idle
9:          else if load_BeOk ≥ β × load_avg then
10:             flag_k ← -1                         ▷ overload
11:         else
12:             flag_k ← 1                          ▷ normal
13:         end if
14:     end for
15: end procedure
```

When a new CN joins a network, TAM assigns the best suitable BeO with the lowest score, among BeOs only with *idle* or *normal* flags. The score is computed as a product of the load of BeO and the hop distance between the BeO and the new CN.

In the case of *overload*, TAM replaces the BeO with a new available BeO (*idle* or *normal*) and notifies to CNs. An overloaded BeO is not reallocated until its flag has changed to others and CNs ask the new BeO to handle its security context instead of former one. Proposed load balancing scheme for the back-end offload architecture mitigates the load concentration to one BeO and improves the overall network performance.

G. Session Management

BeO can have multiple connections from several CNs. To manage these, BeO maintains the lists of connected CNs and their sessions. This information is used in offloading until the session terminates.

When BeO is overloaded and thus replaced, ongoing sessions will not be terminated by force, rather BeO waits until the session time-out reaches, claiming the sticky session. The migration of sessions into *idle* or *normal* BeO might be effective; however, its overhead would become more burden.

When a session finally expires after the application data exchange, BeO should remove all information related to the session only except when a node wants to cache certificate information [16] or when it uses the session resumption extension [17] to accelerate handshake performance.

There can be harmful sessions with end-points. If an end-point is proved to be malicious, BeO stops offloading and instructs a CN to drop packets from the host. In this process, the CN blacklists the malicious host and reject a connection from it in the future. The blacklist can also be provided to CNs beforehand by administrator. In that case, CNs should block all connections listed.

IV. Security Considerations

The Security of an Offloader If BeO is compromised, it can cause the serious malfunction and also can affect related CNs. Then, TAM should immediately deprive the qualification of BeO from the compromised node and notify CNs to dismiss connections. All stored information related to certificate caching and live sessions should be removed. After the compromised BeO has been recovered, it can apply as a candidate of BeO again. Note that a new symmetry secret key should be regenerated.

The Security of TAM When TAM is compromised, CNs cannot be provided a proper security assist, which deactivates the key functionality of the proposed architecture. In this case, TAM should be able to recover itself from attackers by resetting the device, otherwise it should be quickly replaced by an administrator. All registered BeOs and provided keys should be initialized. Note that TAM should always be monitored.

The Security of End-points There are also risks in end-points. When a CN is attacked, it must be isolated from other nodes and BeO. All live connections should be terminated. A recovered CN can rejoin the network and register itself to TAM. In the case of a remote end-point, we require that it must be protected and managed regularly by a remote operator and should not store expired sessions or secret information. Particularly, the end-point vulnerability is not only limited to our architecture, but also relevant to general networks.

V. Conclusion

The main purpose of this study was to suggest a feasible security architecture in resource-constrained networks, even when CNs cannot support security protocols. We presented the back-end offload architecture which allows a CN to establish a secure session between a remote end-point, without the burden of deploying a security protocol. This architecture introduces the back-end offloader (BeO) and trust association manager (TAM). Also it takes account of the load balancing of offloaders, reflecting the current load, hop distance, and network conditions. The proposed architecture would be advantageous for incompetent CNs to have security with low cost. This study has taken a step in the direction of defining the architecture itself; thus, extensive future experiments should evaluate the performance of the proposed architecture and determine its feasibility in a real world IoT system.

Acknowledgment

This work was partly supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.R01261610020001002, Development of agro-livestock cloud and application service for balanced production, transparent distribution and safe consumption based on GS1) and supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2016-H8601-16-1007) supervised by the IITP.

References

[1] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," 2012.

[2] R. Hummen, H. Shafagh, S. Raza, T. Voig, and K. Wehrle, "Delegation-based authentication and authorization for the ip-based internet of things," in *2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, June 2014.

[3] R. Hummen, J. H. Ziegeldorf, H. Shafagh, S. Raza, and K. Wehrle, "Towards viable certificate-based authentication for the internet of things," in *Proceedings of the 2Nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, ser. HotWiSec '13. New York, NY, USA: ACM, 2013.

[4] S. Gerdes, O. Bergmann, and C. Bormann, "Delegated coap authentication and authorization framework (dcaf)," *IETF draftgerdes-core-dcaf-authorize-02*, 2014.

[5] S. Raza, D. Trabalza, and T. Voigt, "6lowpan compressed dtls for coap," in *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, May 2012.

[6] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure coap for the internet of things," *Sensors Journal, IEEE*, vol. 13, no. 10, 2013.

[7] T. Fossati and H. Tschofenig, "TLS/DTLS Profiles for the Internet of Things," Internet Engineering Task Force, Internet-Draft draft-ietf-dice-profile-17, Oct. 2015, work in Progress.

[8] S. Gerdes, L. Seitz, G. Selander, and D. C. Bormann, "An architecture for authorization in constrained environments," IETF, Internet-Draft draft-ietf-ace-actors-03, Mar. 2016, work in Progress.

[9] C. Bormann, M. Ersue, and A. Keranen, "Terminology for constrained-node networks," *Internet Engineering Task Force (IETF), RFC*, vol. 7228, 2014.

[10] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.

[11] J. Han, M. Ha, and D. Kim, "Practical security analysis for the constrained node networks: Focusing on the dtls protocol," in *Internet of Things (IOT), 2015 5th International Conference on the*, Oct 2015.

[12] M. Sethi, J. Arkko, A. Kernen, and H.-M. Back, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks," Internet Engineering Task Force, Internet-Draft draft-aks-lwig-crypto-sensors-00, Oct. 2015, work in Progress.

[13] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," 2014.

[14] "Oma lightweight m2m," 2016. [Online]. Available: http://openmobilealliance.org/about-oma/work-program/m2m-enablers

[15] C. Kuo, M. Luk, R. Negi, and A. Perrig, "Message-in-a-bottle: User-friendly and secure key deployment for sensor nodes," in *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, ser. SenSys '07. New York, NY, USA: ACM, 2007.

[16] S. Santesson and H. Tschofenig, "Transport layer security (tls) cached information extension," *Transport*, 2015.

[17] R.Hummen, H.Wirtz, J.H.Ziegeldorf, J.Hiller, and K.Wehrle, "Tailoring end-to-end ip security protocols to the internet of things," in *21st IEEE International Conference on Network Protocols(ICNP)*, Oct 2013.

MANETs Monitoring with a Distributed Hybrid Architecture

Jose Alvarez and Stephane Maag
SAMOVAR, Telecom SudParis, Université Paris-Saclay
9 Rue Charles Fourier, 91000, Evry, FR
{*jose_alfredo.alvarez_aldana,*
stephane.maag}*@telecom-sudparis.eu*

Fatiha Zaïdi
LRI-CNRS, Université Paris Sud, Université Paris-Saclay
15 Rue Georges Clemenceau, 91400, Orsay, FR
Fatiha.Zaidi@lri.fr

***Abstract*—Monitoring techniques have been deeply studied in wired networks using gossip and hierarchical approaches. However, when applied to a MANET, several problematics arise. We present a hybrid distributed monitoring architecture for MANETs. We get inspired of gossip-based and hierarchical-based algorithms for query dissemination and data aggregation. We define gossip-based mechanisms that help our virtual hierarchical topology to complete the data aggregation, and then ensure the stability and robustness of our approach in dynamic environments. We propose a fully distributed monitoring protocol that ease the nodes communications. We evaluate our approach by using NS3 and Docker.**

1. Introduction

Network monitoring have been deeply studied in P2P, DTN and others using gossip-based or hierarchical-based approaches. However, when it is applied to a wireless mobile ad hoc network (MANET), new problematics arise mainly due to the absence of a centralized administration, the inherent MANETs properties and the node mobility. Some approaches propose a coordinator, nevertheless, due to energy efficiency, infrastructure or other parameters, these solutions are not always applicable.

While studying the monitoring of a network, the most common and intuitive approach is to define a central node as a coordinator for storage and processing of the observations. This is notably proposed by [1], where the author surveys the different communication mechanisms. These centralized architectures might be efficient for certain type of topologies, but become critical when considering dynamic topologies. This is why there has been a lot of efforts on decentralized monitoring. Gossip-based approaches show an extraordinary robustness and stability in dynamic scenarios and changing topologies. Nonetheless, depending on the scalability, the cost and performance can be impacted. On the other side, hierarchical approaches show an efficient performance, cost and scalability, although the robustness and stability may decrease in dynamic scenarios. This shows that the two major categories perform very good under different characteristics, requirements and constraints of a network [7].

The main contribution of this paper is the proposal of a hybrid algorithm for decentralized monitoring of MANETs. We define an architecture combining gossip-based and hierarchical-based algorithms for query dissemination and data aggregation. We perform the gossip-based approach to disseminate the query and in the process to build a virtual hierarchical topology (VHT) for a time window. Once the query is disseminated through all the network, with the support of the VHT, a hierarchical-based aggregation takes place. The second contribution of this paper is the definition of a monitoring protocol that aims at helping a decentralized monitoring process. Our expectation is not just to provide a structure but also a mathematical background for further model checking and testing. Our protocol has been successfully assessed using NS3 and Docker.

The remaining of our paper is as it follows. In Section 2, we present our hybrid algorithm. In Section 3, we present our implementation, with a semi-formal support for our protocol. Next, in Section 4, we present some interesting related works from which we got inspired. Finally, we conclude and give some perspectives in Section 5.

2. Hybrid Monitoring Approach

Network monitoring can be described as "A number of observers making observations, and wish to work together to compute a function of the combination of all their observations" [1]. The goal is that all the nodes in the network compute a value $t \mapsto f(t)$ [$\mathbb{R}^{+*} \rightarrow X$, X being the domain targeted by f] in a given instant of time t in a collaborative way. For our purposes, f is a linear and non-complex function (e.g., the average CPU).

Our hybrid algorithm architecture consists in two network states, the "query state" and the "aggregate state". The idea is to combine a gossip approach and a hierarchical approach to achieve the monitoring of a property of the network. The communication between the nodes to achieve the monitoring of the network will be achieved through a package previously defined. The idea is that a start node will start the monitoring process by propagating a monitoring query in a gossip approach. The approach chosen will be described as epidemic. Each hop, the nodes will exchange information creating a virtual hierarchical topology (VHT) which will be valid only during the monitoring process. Then based on this topology, the nodes will start aggregating

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

the information by sending their results to the parent node. Once the aggregation is done and has reached the start node, there will be a global view of the measured property and the VHT will no longer be usable. If the process starts again, a new VHT will be derived. The purpose is to establish a VHT during one time window, duration of the monitoring process, to ease the analysis and the global view of the property.

2.1. Hybrid Architecture

2.1.1. Query State. The query state refers to the process of propagating in an epidemic way the monitoring packet. This state goal is to disseminate the query and the VHT layout to allow the nodes in the network to do an accurate and efficient aggregation in the next state. This query will be forwarded in an epidemic approach to the nodes in the relay set. The packet is explained in depth in Section 3, containing the query itself but also the information to generate the VHT. This will communicate all the network information to create the VHT, which is the foundation of the following state of the network. This process will go on until a node on the edge of the network is reached.

Along this state, there are some specific challenges to discuss. (i) The first challenge is if a node receives more than one monitoring packet once it is already in a monitoring state. For this, the node will take the first monitoring packet and will discard all the subsequent monitoring packets. (ii) The second challenge is, what if the propagation of the query is interrupted by a node that remains in a cyclic state. For this, we introduce a timeout for the packet to avoid these problems. The idea is to provide a mechanism to avoid loops in the communications. For this problem, the timeout will be triggered and once reached, this node will start the aggregation process by sending its result to the parent node. (iii) The third challenge is the broadcast of the packet itself. Due to the nature of the simple epidemic dissemination approach, a packet will be forwarded to the next hop of nodes but also to the parent node. We decided that this will work as an acknowledgment of the child node to the parent node. This way, the parent node will receive n acknowledgments and he will know how many packets he should wait for before changing to the aggregate state.

2.1.2. Aggregate State. Once the data is disseminated up to the edge of the network, the edge nodes will change from query state to aggregate state and will start sending recursively their information to their parent up to the start node. This process will be an aggregation of all the data of a node and his children in order to collect the monitored values. The aggregation will be computed in a hierarchical manner with a combination when required of a gossip approach. A node will compute based on his own observations the result of the function $f(x)$ that received from the query state. This information will be aggregated with the same child nodes information. In the edge nodes cases, where this state starts, it will be done only with information from themselves.

Along this state, there are some specific challenges to discuss. (i) The first challenge is when a parent node and

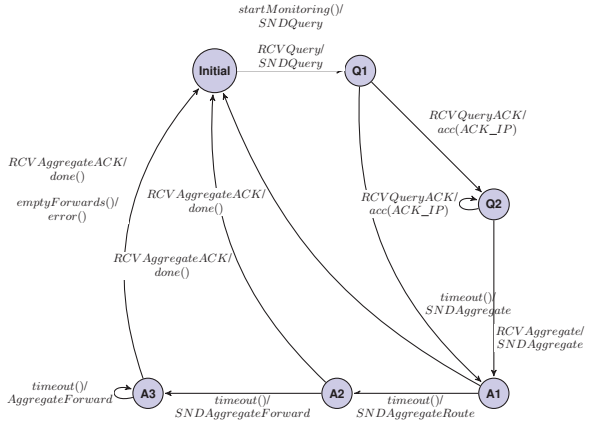


Figure 1: State machine definition of our protocol

a corresponding child node goes out of range from when they first met. When the child node sends an aggregate type of message and receives no acknowledgment it will trigger a forward packet to the corresponding node. For this, we will rely on the routing protocol of the network. This makes our approach dependent on the routing layer of the network and we will consider our own opportunistic routing mechanism in forthcoming works. (ii) The second challenge is when a parent node is off line. For this, we propose that in the query state, a set of nodes are communicated to every child node for them to have an alternative path. Since the child node will have the relay set of the parent node, he will fall back into one of these nodes to send the information. Since it is a hierarchical approach, the parent will send the information about his parents in the VHT. (iii) The third challenge is when a node receives a grandchild node aggregate information. For this, the node will assume that the child node is off line and that he will be aggregating that information. Given that the node does not know the information of how many grandchild will send information, he will also rely on the timeout before he sends his own aggregate information. For every grandchild packet he receives, he will restart the timeout to give time for additional packages. If the timeout is reached, he will continue with his aggregation process.

3. Experiments

3.1. Protocol Definition

The protocol definition, depicted in Figure 1, shows the expected behavior of the protocol to support as base ground for the hybrid monitoring architecture. The set of states is $Q = (Initial, Q1, Q2, A1, A3, A3)$. Where $Initial$ is the initial state. The states $Q1$ and $Q2$ refer to the query states of the network. And the states $A1$, $A2$ and $A3$ refer to the aggregate states of the network. The internal operations of the automaton are startMonitoring(), acc(IP), timeout(), done() and error(). The startMonitoring() refers to the process of starting the monitoring. The acc(IP) refers to the process of the node of accumulating the IP of the acknowledgment

messages source. This is used to identify while the query is propagating if there are child nodes available for a given node. If a node does not receive any acknowledgment, he will continue the monitoring process by using a timeout. The timeout() refers to the process of counting time since the last package received. The done() refers to the restart of the state of the node. Meanwhile, error() refers to the process of not being able to send a message, which if it happens, it means that the node itself is out of the network range or a major outage is happening with the network. The input and output operations of the automaton are determined by sending (SND) and receiving (RCV) messages. The possible messages to be sent or received are the query, query ack, aggregate, aggregate ack, agregate route and aggregate forward. The query message refers to the query itself and the base ground of the query state. For simplicity purposes, in the automaton, there is a distinction between the query and the query ack message. But in reality, they are meant to be the same package but received by a different node. This is discussed in Section 2.1.1. The aggregate messages refer to the aggregation process and the same principle applies as the query messages. The aggregate ack message is an aggregate message but received by a different node. Then we also have two extra messages which are the aggregate route and aggregate forward. The aggregate route message refers to the process of routing a message through the network to the corresponding parent node in the VHT. As explained in Section 2.1.2, the idea is to make the hierarchical aggregation more robust through the addition of a gossip routing approach to route the package to the corresponding root node on the fly. And finally, the aggregate forward message, which in the case that the parent node is not found, probably because the parent node went offline due to an outage or something similar. In this case, the message will be forwarded to one of the nodes defined in the relay set, which will be populated by the grandparents and siblings.

3.2. Packet Definition

In order for the communication to be successful, we need to define the monitoring packet. The packet will work equally in both states of the network, query and aggregate states, but different information will be sent depending on the state containing a set of common properties. It needs to contain some basic information in order to be useful for the following nodes and hops. The definition of such packet will be done using json. For each state of the nodes, there will be a set of properties transmitted. There will be a set of global properties that will always be transmitted. These global properties are: 1) Type: the type of message being sent, the set of values is listed in 3.1. 2) Parent: the IP address of the parent node. 3) Source: the IP address of the node sending the message. 4) Timeout: the timeout value in milliseconds. For the query state the properties transmitted are: 1) Function: the function f to compute. 2) Relay Set: the list of IPs for alternative paths, with at most three items. For the aggregate state the properties transmitted are: 1) Result: the result of the aggregation of the function f.

TABLE 1: Scenario 1 & 2 parameters

	Scenario 1	**Scenario 2**
Number of nodes	10, 20, 25, 40, 50, 60, 75, 80 and 100	25
Network Space	500x500, 800x800 and 1,000x1,000	500x500
Network Positioning	Grid (100m apart)	Random
Running time	80s (init time 60s)	80s (init time 60s)
Emulation times	200	40
Mobility	-	RWP
Mobility Speed	-	2m/s and 5m/s

2) Destination: the destination IP that should be the parent IP for most of the cases, unless the parent node is off line, then it will be a relay set IP. 3) Observations: the number of aggregated observations. The json definition of the complete package is the following:

```
{"type": "<type>", "parent": "<parent IP>", "source": "<source IP>",
 "timeout": <ms>,
 "query":{ "function": "<f(t)>", "relaySet": ["<IP list relay set>"] },
 "aggregate":{ "outcome": "<monitoring result>", "destination": <destination IP>,
  "observations": <number of observations> } }
```

3.3. Results

We evaluate our proposal using an emulator built in-house based on DOCKEMU [9]. This emulator is a combination between Docker and NS3, which allows to conduct highly scalable, replicable and robust experiments. The testbed consisted in an implementation of the protocol in the language Go, that was deployed on our emulator. The idea was to determine the convergence time, by which we mean the time it took from the moment that the monitoring started by the root node, to the moment that the root node was able to return a verdict. We defined two scenarios, one scenario with no mobility and another with low mobility. For this study, we are testing the implementation without the mobility support. This means we do not consider states $A2$ and $A3$ of Figure 1. Scenario 1 and scenario 2 consider the parameters of Table 1. For both scenarios the Mac Protocol is 802.11a with a data rate of 54Mbps. Each node had a range of ≈125m. Scenario 1 was designed to test the convergence time, the scenario 2 to prove that due to the high performance of the algorithm, we may monitor in a mobile environment without the mobility support.

The emulator was running on top of an Amazon EC2 t2.large instance and Ubuntu 16.04. Versions in use were Docker 1.12.1, NS3.25 and Go 1.6.2. The containers were running as a base Ubuntu 16.04 LTS and IPv4.

3.3.1. Scenario 1. For our first scenario, we collected the convergence time using a different root node as a starting point in each run. We decided to use different root node selected randomly to prove that it will work independently of who the root node is. The results are summarized in Figure 2. We can point out that there is a clear relationship between the number of nodes and the time it takes to converge. With 50 nodes to 100 nodes, the average value seems to stabilize in around ≈2.15s. About the number of packets sent, we empirically assumed that for the static environments, the number would be twice the number of nodes. This can be deducted because everyone will send their query message

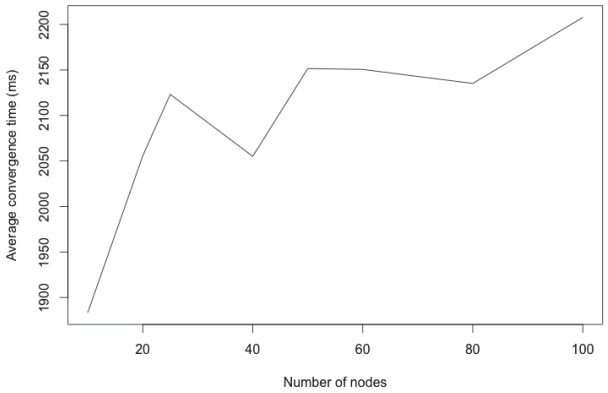


Figure 2: Scenario 1 convergence results

TABLE 2: Scenario 2 results

	Speed 2m/s	Speed 5m/s
Average convergence time (ms)	2018.06	2123.63
Average observations (# nodes)	22.68	21.25
Success rate	0.8	0.4

one time and their aggregate message one time as well. The variability of the nodes will occur when the mobility support is added. For N nodes, the messages sent are $2*N$ for static environments. The average message size is $\approx$151 bytes.

3.3.2. Scenario 2. For the second scenario, we collected the convergence time but also the amount of observations collected by the root node at the end of each run. The results are summarized in Table 2. We can observe that the nodes converge about the same amount of time that they do in a static environment. We observed that it would converge but without all the possible observations on the network. And on top of that, between the more the speed of the nodes the lower the success rate would be. By success rate, we define if the monitoring process was able to converge. The algorithm has proved in static environments that is capable of converging really fast, even though is not using the mobility support, suggesting promising future results.

4. Related Works

MANET monitoring has been studied for many objectives like their performances [2], to test them [3], their security [4] and more recently their energetic efficiency [5].

Gossipico [10] is an algorithm to calculate the average, the sum or the count of node values in a large dynamic network. The foundation of the algorithm is through two parts: count and beacon. The combination of these two mechanisms provides the advantage for counting the nodes inside a network in an efficient and quick way. Mobi-G [8] is designed for urban outdoor areas with a focus on pedestrian that moves around. The idea is to create the global view of an attribute incorporating all the nodes in the network. Nevertheless the accuracy decreases for an increasing spatial network size. On the hierarchical categorization, we can mention BlockTree [6], which is a fully decentralized location-aware monitoring mechanism for MANETs. The idea is to divide the network in proximity-based clusters, which are arranged hierarchical. This approach scales with the spatial network size and provides accurate results. In [7], the main key points in architectural description for decentralized monitoring mechanisms are depicted. However, it is difficult to determine a better performer since both perform better in diverse scenarios and workloads.

5. Conclusions

We have presented in this paper a hybrid algorithm for monitoring decentralized networks that consists on the combination of gossip-based and hierarchical-based algorithms. The gossip-based approach is applied to disseminate the query and the hierarchical approach is applied to aggregate the data. Besides, with the help of a time-based hierarchical approach, the computation of a global property is achieved. We designed a scalable and configurable testbed using NS3 and Docker, based on DOCKEMU [9]. Our methodology and results seem promising for a wide set of scenarios.

As future works, we intend to study the selection of the root node. It could be based on location, energy, computing power and other parameters, or to be an autonomous process, proactively or reactively, or a manual process. Besides we plan to introduce the mobility support and enhance the testbeds to this specific cases. We also intend to consider more complex functions in monitoring the MANETs interoperability. For this we need to define an optimal solution to propagate a more complex function through our query mechanism.

References

[1] G. Cormode. The continuous distributed monitoring model. *ACM SIGMOD Record*, 2013.

[2] A. Mehrotra, A. Saxena, and M. Tolani. Performance comparison of different routing protocols for traffic monitoring application. *International Journal of Computer Applications*, 92(4), 2014.

[3] K. Merouane, C. Grepet, and S. Maag. A methodology for interoperability testing of a manet routing protocol. In *3rd Int. Conference on Wireless and Mobile Communications*, pages 5–5, 2007.

[4] A. Nadeem and M. P. Howarth. A survey of manet intrusion detection & prevention approaches for network layer attacks. *IEEE communications surveys & tutorials*, 15(4):2027–2045, 2013.

[5] S. Palaniappan and K. Chellan. Energy-efficient stable routing using qos monitoring agents in manet. *EURASIP Journal on Wireless Communications and Networking*, 2015(1):1, 2015.

[6] D. Stingl, C. Gross, L. Nobach, R. Steinmetz, and D. Hausheer. Blocktree: Location-aware decentralized monitoring in mobile ad hoc networks. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 373–381. IEEE, 2013.

[7] D. Stingl, C. Gross, K. Saller, S. Kaune, and R. Steinmetz. Benchmarking decentralized monitoring mechanisms in peer-to-peer systems. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 193–204. ACM, 2012.

[8] D. Stingl, R. Retz, B. Richerzhagen, C. Gross, and R. Steinmetz. Mobi-g: Gossip-based monitoring in manets. In *IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9, 2014.

[9] M. A. To, M. Cano, and P. Biba. Dockemu–a network emulation tool. In *IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pages 593–598, 2015.

[10] R. Van De Bovenkamp, F. Kuipers, and P. Van Mieghem. Gossip-based counting in dynamic networks. In *International Conference on Research in Networking*, pages 404–417. Springer, 2012.

Secure Complex Monitoring Event Processing

Mehdi Bentounsi
LIPADE, Université Paris Descartes
Sorbonne Paris Cité, France
mehdi.bentounsi@parisdescartes.fr

Salima Benbernou
LIPADE, Université Paris Descartes
Sorbonne Paris Cité, France
salima.benbernou@parisdescartes.fr

***Abstract*—In this paper, we present EMaaS, for Event Management as a Service, a multi-tenant cloud service to outsource the management of monitoring events in the cloud. Furthermore, we provide a secure multi-party computation (SMC) protocol for complex monitoring event processing. The protocol is integrated to EMaaS and uses some properties of encryption schemes. Finally, we evaluate the protocol security, and then discuss its implementation.**

***Index Terms*—Cloud Computing, Monitoring, Multi-Party Computation, Encryption Schemes, Security by Design.**

I. Introduction

Business processes (BPs) consist of companies' core business and are significant source of revenues. In such setting, *Business Process Management Systems* (BPMSs) are software platforms that support the definition, execution and tracking of BPs [1]. Furthermore, BPMSs give health information about the BPs they support at the runtime with the aim of helping companies to *(i)* optimize and improve Quality of Service (QoS) of their BPs, *(ii)* alert them in case of faults, and *(iii)* identify the root causes of incidents.

BPs are implemented on top of IT infrastructures and are highly dependent on their operations. Therefore, IT monitoring software are integrated to BPMSs to recover and process monitoring events generated by monitoring agents deployed in infrastructures. From operational perspective, exploiting monitoring events is time consuming and often requires human operators to identify their origins and interpret their significances. Thereby, a new generation of IT monitoring software appeared with Nagios [2]. The goal was to limit number of monitoring events by adding preprocessing filters at the monitoring agent level. Thus, metrics are simply compared to thresholds before generating monitoring events, e.g., "`cpu utilization is above 80%`".

Complex monitoring event processing (CMEP) is done to have a higher level of knowledge on incidents. Indeed, more complex faults, including several parameters at once, can happen in infrastructures, e.g., "`cpu utilization is above 80%`" in two separate servers which affects several applications. Consequently, deployed agents in each server can not detect the faults by analysing metrics separately. For this purpose, a global correlation using a set of metrics and predefined rules was proposed [3], [4], and several *Event Management Software* (EMS) coming out on the market. We can mention: IBM Tivoli Netcool/OMNIbus and BMC Event Manager. However, the main obstacle to the broad adoption of such systems by SMEs remains high capital expenditure (CAPEX) and operational expenditure (OPEX).

978-1-5090-3216-7/16/$31.00 ©2016 IEEE

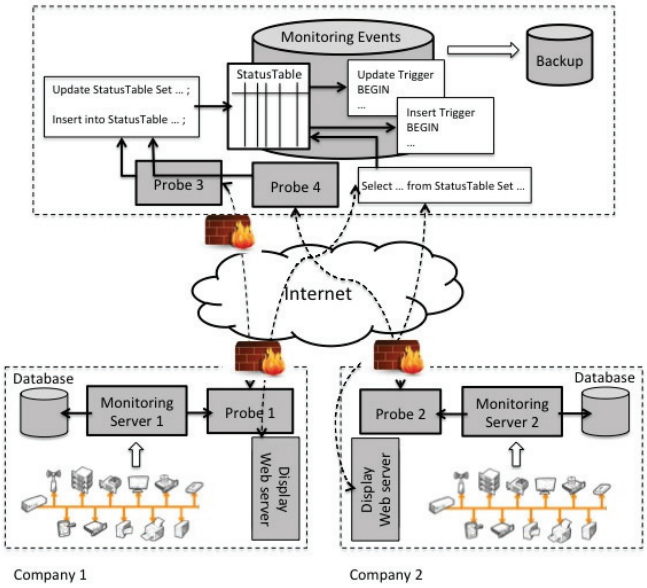


Fig. 1. EMaaS Global Architecture.

EMaaS [5], [6], for *Event Management as a Service*, is a cloud service which offers a sharing instance of EMS between several SMEs in order to reduce CAPEX, and the outsourcing of the operation to a specialized company in order to control OPEX. As depicted in Figure 1, EMaaS is considered as a multi-party cloud system [7], which consists in three parts: *(i)* a public cloud platform, i.e., *process curator*, providing a remote IT infrastructure to host the cloud service; *(ii)* a service provider company that implements and operates the EMS instances, i.e, *process provider*; and *(iii)* multiple organizations outsourcing their EMS to the cloud, i.e, *process consumers*.

Basically, the security architecture of EMaaS is based on the security perimeter of both process curator and consumers, and the security of communication channel. Consequently, this conventional security architecture ensures the security against only external attacks. However, in a multi-party cloud system as EMaaS, computations could occur between fully trusted partners, partially trusted partners, or even between competitors (see [8] for more details). In such context, when

two or more parties want to conduct computations based on their private inputs, but neither party is willing to disclose its own input to anybody else. This problem is referred to as Secure Multi-party Computation problem (SMC) in the literature [9].

To address these issues, we provide a SMC protocol to ensure the security, i.e., confidentiality and integrity, of monitoring events against curious and malicious adversaries, and at the same time preserve data utility. Our contributions can be summarized as follows: 1) We define a security model of multi-party cloud system for complex monitoring event processing taking into account semi-honest third party and competitor attacks. 2) We provide a secure multi-party computation protocol to ensure the confidentiality of monitoring events against curious adversaries, and the integrity against malicious competitors. 3) We discuss the integration of the protocol in the architecture of EMaaS.

II. Formalization and Security Definition

An EMS can be defined as a monitoring events database coupled with a set of correlation rules. Formally,

Definition 1. (Monitoring Event) A monitoring event is a timed sequence of data values regarding a given IT component. A structured monitoring event is a tuple $e = \langle \bar{v}^{ci}, \bar{v}^{cm}, ts \rangle$ where: $\bar{v}^{ci} = (v_0^{ci}, ..., v_i^{ci})$ are component identification values with corresponding attributes $\bar{a}^{ci}$; $\bar{v}^{cm} = (v_{i+1}^{cm}, ..., v_n^{cm})$ are metric values with corresponding attributes $\bar{a}^{cm}$; and ts is a temporal value indicating the starting timestamp of the event.

Definition 2. (Monitoring Events Table) Given a monitoring events table T with a set of attributes $\{A_{ts}, A_1, \ldots, A_n\}$, $t\,[A_i]$ refers to the value of attribute A_i for the tuple t. Attributes of a table T are divided as follows:

- Timestamp A_{ts} indicating when the information is collected.
- Component-identifiers A^{ci} represent attributes that can be used (possibly with external information available to the adversary) to identify the infrastructure component associated with a tuple in a table. user id, IP address, and hostname are examples of component-identifiers.
- Component-metrics A^{cm} contain health information regardling an infrastructure component. cpu utilization rate is considered as health information.

Definition 3. (Correlation rules) A correlation rule can be defined as a clause with a set of predicate, P, being in CNF form with respect to a monitoring events table T, has the following form:

$$P = \bigwedge_{1 \le i \le n} \left(\bigvee_{1 \le j \le m_i} P_i^j \right)$$

where: P_i^j is a single-literal clause having a form $att\ op\ value$ and $op \in \{=, \le, <, \ge, >\}$. Each set of P_i's are defined further as: $P_{ci} = P_1 \wedge ... \wedge P_\alpha\ and\ P_{cm} = P_{\alpha+1} \wedge ... \wedge P_n$, where: P_{ci} are only applicable to component-identifiers attributes A^{ci} when $op \in \{=\}$, and P_{cm} are only applicable to component-metrics attributes A^{cm} when $op \in \{=, \le, <, \ge, >\}$.

Definition 4. (Adversary Model) We distinguish two adversary models. *Curious adversaries* can eavesdrop on various components of the system. These include: *(i)* the monitoring events table which contains all information that the EMS stores about the consumer, *(ii)* the communication channel which has all information sent between the consumer and curator, and *(iii)* the result of the processing. We consider adversaries that can eavesdrop on consumer components as outside of our attack model. However, *malicious adversaries* may not only be able to eavesdrop on the communication channel but could also insert fake monitoring events. We consider adversaries that can change the monitoring events table and correlation rules as outside of our attack model.

Definition 5. (Security Definition) To ensure *confidentiality* we will show that the protocol uses a cryptosystem without a key exchange, and the key-pair is stored inside the process consumer security perimeter. Moreover, complex monitoring event processing is done over encrypted component-identifiers and the adversary should recover the key-pair or cryptanalysis the system in order to infer them. To ensure *integrity* we will show that there is a check in place (either by the server or by the client) that an adversary with the specific resources cannot pass without having to invert a one-way function.

III. SMC Protocol for EMaaS

We outline a SMC protocol which consists of Five phases: *A)* system setup phase, *B)* monitoring events anonymization phase, *C)* correlation rules rewriting phase, *D)* results post-processing phase, and *E)* key change phase.

A. System Setup

To initialize the protocol, each process consumer generates a private key-pair and keyed hash function that are securely stored, i.e., no key exchange is required. Thus, a key generator $KeyGen_l(\lambda)$ outputs a key-pair $(k, \acute{k})$ of lenght l using a security parameter λ and a hash function $hash_y(.)$. According to the NIST, the longevity of such key-pair, in both symmetric and asymmetric cryptosystems, is less than 2 years [10]. Encryption key-pair $(k, \acute{k})$ is unique for each process consumer. Therefore, given two process consumers with corresponding key-pairs $(k_1, \acute{k}_1)$ and $(k_2, \acute{k}_2)$, ciphertexts $c, \acute{c}$, and plaintexts $m, \acute{m}$. We introduce collision propreties:

$$\forall m : Encrypt_{k_1}(m) \neq Encrypt_{k_2}(m)$$

$$\forall c : Decrypt_{\acute{k}_1}(c) \neq Decrypt_{\acute{k}_2}(c)$$

$$\exists \acute{m} \neq m : Encrypt_{k_1}(m) = Encrypt_{k_2}(\acute{m})$$

$$\exists \acute{c} \neq c : Decrypt_{\acute{k}_1}(c) = Decrypt_{\acute{k}_2}(\acute{c})$$

B. Monitoring Events Anonymization

We previously showed that a monitoring event contains component-identification values and metrics. Metrics should stay in clear without modification. This is due to the fact that mathematical operations are basically done on these values. However, component-identification values used to identify a monitoring event must be encrypted using a *deterministic* cryptosystem to ensure their anonymity and to guarantee data utility (i.e., two consecutive encryptions of the same plaintext output the same ciphertext). Based on the definition of a monitoring event, we introduce a monitoring event transformation function T to anonymize a monitoring event e by encrypting component identification values and adding a control value to authenticate events and avoid collision: $\mathrm{T}(e) = e^*$, where e^* represents the anonymized monitoring event. Formally,

Definition 6. (Monitoring Event Anonymization) Given a key-pair $(k, \acute{k})$ and a monitoring event $e = \langle \bar{v}^{ci}, \bar{v}^{cm}, ts \rangle$. An anonymized monitoring event $e^* = \mathrm{T}(e)$ is defined as:

$$\mathrm{T}_{(k,\acute{k})}(\langle \bar{v}^{ci}, \bar{v}^{cm}, ts \rangle) = \langle Encrypt_k(\bar{v}^{ci}), \bar{v}^{cm}, ts, ctl \rangle$$

where: $Encrypt_k(\bar{v}^{ci}) = (Encrypt_k(v_0^{ci}), ..., Encrypt_k(v_i^{ci}))$ are ciphertexts of component identification values encrypted using the key-pair $(k, \acute{k})$, $\bar{v}^{cm}$ are metric values, ts is the starting timestamp, and $ctl = hash_y(t_{s-1})$ is the keyed-hash value of previous event timestamp.

After anonymization process, monitoring events are send *on-the-fly* to process curator, through a VPN, to be stored in the monitoring events table. The process curator is only allowed to try to infer information about internal architecture of process consumers and it is assumed not to return incorrect or/and incomplete result, or alter the protocol in an attempt to gain information. Moreover, the curator does not modify the monitoring events table periodically updated by the process consumers.

C. Correlation Rules Rewriting

Given the fact that monitoring events are anonymously stored in the monitoring events table, correlation rules should be rewritten (at the process consumer side) to take into account the anonymized monitoring events. For that, single-literal clauses P_{ci} applicable to component-identifiers attributes A^{ci}, having a form $att = value$, are rewritten as follows: $att = Encrypt_k(value)$. This permits to identify monitoring events when only *equality* operation is necessary.

However, regarding clauses P_{cm} applicable to component-metrics attributes A^{cm}, having a form $att\ op\ value$ where: $op \in \{=, \leq, <, \geq, >\}$, are not modified to permit mathematical operations. Moreover, transfering and processing these information in clear will not permit to the adversary to infer information about the internal architecture of the IT infrastructure. Formally,

Definition 7. (Correlation Rule Rewriting) Given a key-pair $(k, \acute{k})$ and a correlation rule P with respect to a monitoring events table T where:

$$P = \bigwedge_{1 \leq i \leq n} \left(\bigvee_{1 \leq j \leq m_i} P_i^j \right)$$

A correlation rule P^* with respect to an anonymized monitoring events table T^* is defined as :

$$P^* = R_{(k,\acute{k})}(P)$$

$$P^* = \bigwedge_{1 \leq i \leq n} \left(\bigvee_{1 \leq j \leq m_i} Encrypt_k(P_i^j) \right)$$

If P_i^j is applicable to a component-identifier A^{ci} then $Encrypt_k(P_i^j) \equiv (att = Encrypt_k(value))$ else $Encrypt_k(P_i^j) \equiv P_i^j$.

After rewriting process, correlation rules are stored in the process curator. The process curator nor provider do not modify the correlation rules set periodically updated by the process consumers.

D. Results Postprocessing

The outputs of complex monitoring event processing will be in the form of sets of anonymous event e^*. Anonymous monitoring events sets should be recovered by the process consumer, and then decrypted in order to generate alerts. For this purpose, the key-pair $(k, \acute{k})$, generated at setup phase, is used to de-anonymize monitoring events. Formally,

Definition 8. (Results Postprocessing) Given a key-pair $(k, \acute{k})$ and an anonymous monitoring event $e^* = \langle Encrypt_k(\bar{v}^{ci}), \bar{v}^{cm}, ts, ctl \rangle$. A result monitoring event $e^\star = \mathrm{T}^{-1}(e^*)$ is defined as:

$$\mathrm{T}^{-1}_{(k,\acute{k})}(\langle Encrypt_k(\bar{v}^{ci}), \bar{v}^{cm}, ts, ctl \rangle) =$$

$$\langle Decrypt_{\acute{k}}(Encrypt_k(\bar{v}^{ci})), \bar{v}^{cm}, ts, Verif(ctl) \rangle =$$

$$\langle \bar{v}^{ci}, \bar{v}^{cm}, ts, Verif(ctl) \rangle$$

Where the function $Verif(ctl)$ permits to authenticate monitoring events.

E. Key change

The process consumer must change the key-pair $(k, \acute{k})$ periodically. To do so a new key-pair $(k^\circ, \acute{k}^\circ)$ is generated, and during a period of time λ, monitoring events will be duplicated and anonymized using both the old key-pair $(k, \acute{k})$ and new key-pair $(k^\circ, \acute{k}^\circ)$. Also for the correlation rules. At the end of this period of time λ, only the new key-pair $(k^\circ, \acute{k}^\circ)$ is used to anonymize monitoring events and correlation rules.

IV. Security Analysis of SMC Protocol for CMEP

We briefly analyse the security of the proposed protocol. More details will be given in an extended version of the paper.

Theorem 1. (Confidentiality) The protocol is *as secure as* the symmetric cryptosystem used to cipher the compnent-identification values.

PROOF SKETCH. Until the process consumer arrives to guarantee the confidentiality of the encryption/decryption key-pair $(k, \acute{k})$, an adversary can not decipher component-identifiers values for monitoring events and then infer sensitive information. In addition, monitoring events are processed in the server side as they are provided by the client and are never decipher outside its security perimeter. However, the protocol is not proven secure against brute force attacks, and according to NIST recommendation the key-pair should be modified each 2 years.

Theorem 2. (Integrity) The protocol is *as secure as* the keyed hash function used to calculate the control value in anonymized monitoring events.

PROOF SKETCH. Until the process consumer arrives to guarantee the confidentiality of the keyed hash function $hash_y(.)$, an adversary can not insert a new monitoring event instead of a process consumer.

V. Implementation

To integrate the protocol to EMaaS, we used a transformation framework consisting in Three modules:

A. Correlation Rule Rewriting Module

Our approach to rewrite rules is based on a lightweight agile parsing techniques supported by the TXL source transformation system. TXL [11] is a special-purpose programming language designed to provide rule-based source transformation using functional specification and interpretation. TXL programs have two main parts : a context-free grammar that describes the syntactic structure of inputs to be transformed, and a set of context sensitive, example-like transformation rules organized in functional programming style. It consists in Three phases:

1) A parsing phase to create an internal representation of the correlation rule as a parse tree under control of a context-free grammar. We use the SQL grammar to construct parse trees.
2) A transformation phase to transform the parse trees created by the parser under control of a set of example-like transformation rules. At this stage, we identify attributes that should be encrypted and we add corresponding transformation rules using the key-pair $(k, \acute{k})$.
3) An unparsing phase to unparse the transformed parse tree to text output with standard spacing and pretty-printing under control of the grammar. We use the SQL grammar to generate the new correlation rules.

B. Monitoring Events Anonymization Module

Plug-ins are implemented on existing monitoring tools in order to anonymize events and transfer them to the probe which syncs with the remote server. Using the key-pair $(k, \acute{k})$ and a set of component identification attributes $\bar{a}^{ci}$, the plug-ins intercept monitoring events then encrypt corresponding values. The keyed hash function $hash_y(.)$ permits to add a control value to check and authenticate monitoring events.

C. Results Post-processing and Display Module

We show complex monitoring event processing results using a web interface or a mobile application. In the two cases, the web server and the web service are in the server side, i.e., component identification values can not be decrypted. Therefore, a plugin is necessary in the web browser (in the client side) to permit to decipher identification values and display alert. Based on the same principle, the mobile application integrates a mechanism to store the key-pair $(k, \acute{k})$ in order to decipher monitoring events and authenticate them using $hash_y(.)$.

VI. Conclusion and future work

This paper has introduced a SMC protocol for complex monitoring event processing. Unlike conventional security architectures, the protocol is secure in that component-identification values are protected, it is "hard" for a curious to infer sensitive information about process consumers IT infrastructure. Moreover, malicious can not perturb health information of process consumers. As future work, we will evaluate communication-computation-storage tradeoff between several cryptosystems and different keylengths.

References

[1] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M. Shan, "Business process intelligence," *Computers in Industry*, vol. 53, no. 3, pp. 321–343, 2004.

[2] C. Gaspar, "Deploying nagios in a large enterprise environment," in *LISA 2007, Dallas, Texas, USA, November 11-16, 2007*, 2007.

[3] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White, "Cayuga: A general purpose event monitoring system," in *CIDR 2007, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, 2007, pp. 412–422.

[4] S. P. Miri, P. Garg, B. Schultz, S. K. Singhal, and M. Sivakumar, "Cross-machine event log correlation," 2013, patent WO/2013/039815.

[5] M. Bentounsi, "Business process as a service - bpaas: Securing data and services," Ph.D. dissertation, Université Paris Descartes, 2015.

[6] M. Bentounsi and C. S. Deme, "Procédé sécurisé d'analyse externe de données d'exploitation d'une infrastructure de traitement de données," 2015, french patent App 15.61009.

[7] M. Bentounsi, S. Benbernou, and M. J. Atallah, "Security-aware business process as a service by hiding provenance," *Computer Standards & Interfaces*, vol. 44, pp. 220–233, 2016.

[8] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: a review and open problems," in *Proceedings of the New Security Paradigms Workshop 2001, Cloudcroft, New Mexico, USA, September 10-13, 2001*, 2001, pp. 13–22.

[9] A. C. Yao, "Protocols for secure computations (extended abstract)," in *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, 1982, pp. 160–164.

[10] "Recommendation for key management," National Institute of Standards and Technology, Tech. Rep., 2012.

[11] J. R. Cordy, "The TXL source transformation language," *Sci. Comput. Program.*, vol. 61, no. 3, pp. 190–210, 2006.

List of Authors