UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

TÉCNICO
LISBOA

On the Service Placement in Community Network Micro-Clouds

Mennan Selimi

a dissertation
presented to the faculty
of Universitat Politècnica de Catalunya
in candidacy for the degree
of Doctor of Philosophy

recommended for acceptance
by the Department of
Computer Architecture

Advisors: Dr. Felix Freitag & Dr. Luís Veiga

January 2017

Abstract

Community networks (CNs) or "*Do-It-Yourself*" networks have gained momentum in the last few years in response to the growing demand for network connectivity in rural and urban communities. These networks, owned and managed by volunteers, offer various services to their members. Seamless computing and service sharing in CNs have gained momentum due to the emerging technology of community network *micro-clouds*. By putting services closer to users, community network *micro-clouds* pursue not only a better service performance, but also a low entry barrier for the deployment of mainstream Internet services within the CN. Unfortunately, the provisioning of the services is not so simple. Due to the large and irregular topology, high software and hardware diversity, asymmetric quality of wireless links in CNs, this requires a challenging effort to "carefully" assess and optimize the service and network performance.

In order to understand the *micro-cloud* based service performance, we perform deployment, feasibility analysis and in-depth performance assessment of *micro-cloud* services such as distributed storage, live video-streaming and discovery service. We characterize and define workload upper bounds for successful operations of such services in *micro-clouds* and perform cross-layer analysis and optimizations to improve the service performance. This deployment experience supports the feasibility of community *micro-clouds*, and our measurements contribute to understand the performance of services and applications in this challenging environment.

On the network level, in order to optimize the performance of the services by the network over which a service host provides a service to client nodes, it is necessary to continuously adapt the logical network topology to both

external (e.g., wireless connectivity, node availability) and internal (e.g., service copies, demand) factors. To achieve this, we propose to leverage state information about the network to inform service placement decisions, and to do so through an i) exploratory algorithm *PASP* that minimizes the service overlay diameter, while fulfilling service specific criteria and ii) through a fast and low-complexity service placement heuristic *BASP*, which maximizes bandwidth and improves user QoS.

Our results show that *PASP* and *BASP* consistently outperforms the existing in-place and naturally fast strategy in `Guifi.net`, with respect to end-to-end bandwidth and client response time when used with real *micro-cloud* services. Since this improvement translates in the QoE perceived by the user, our results are relevant for contributing to higher quality of experience, a crucial parameter for using services from volunteer-based systems.

Keywords

community networks; community micro-clouds; service placement;

List of Publications

The research results from this thesis have led to the following publications:

Journal Articles

[Sel+15a] Mennan Selimi, Felix Freitag, Llorenç Cerdà-Alabern, and Luís Veiga. "Performance evaluation of a distributed storage service in community network clouds". In: *Concurrency and Computation: Practice and Experience* 28.11 (2015). (JCR IF: 0.942, Q3), pp. 3131–3148.

[Sel+15b] Mennan Selimi, Amin M Khan, Emmanouil Dimogerontakis, Felix Freitag, and Roger Pueyo Centelles. "Cloud services in the Guifi.net community network". In: *Computer Networks* 93.P2 (Dec. 2015). (JCR IF: 1.446, Q2), pp. 373–388.

Conference Proceedings

[Sel+16a] Mennan Selimi, Davide Vega, Felix Freitag, and Luís Veiga. "Towards Network-Aware Service Placement in Community Network Micro-Clouds". In: *22nd International Conference on Parallel and Distributed Computing (Euro-Par 2016)*. (CORE Rank A). Grenoble, France, 2016, pp. 376–388.

[Sel+16b] Mennan Selimi, Llorenç Cerdà-Alabern, Liang Wang, Arjuna Sathiaseelan, Luís Veiga, and Felix Freitag. "Bandwidth-aware Service Placement in Community Network Micro-Clouds". In: *41st IEEE Conference on Local Computer Networks (LCN 2016)*. (CORE Rank A) (Short paper). Dubai, UAE, 2016.

[Sel+16c] M. Selimi, N. Apolónia, F. Olid, F. Freitag, L. Navarro, A. Moll, R. Pueyo, and L. Veiga. "Integration of an Assisted P2P Live Streaming Service in Community Network Clouds". In: *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom 2015)*. Nov. 2016, pp. 202–209.

[Sel+15] M. Selimi, F. Freitag, R. P. Centelles, A. Moll, and L. Veiga. "TROBADOR: Service Discovery for Distributed Community Network Micro-Clouds". In: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications (AINA 2015)*. (CORE Rank B). Mar. 2015, pp. 642–649.

[Sel+14] M. Selimi, F. Freitag, R. P. Centelles, and A. Moll. "Distributed Storage and Service Discovery for Heterogeneous Community Network Clouds". In: *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC 2014)*. Dec. 2014, pp. 204–212.

Pending Review

The following papers have been submitted for review.

[Sel+17] Mennan Selimi, Llorenç Cerdà-Alabern, Marc Sanchez-Artigas, Felix Freitag, and Luís Veiga. "Practical Service Placement Approach for Microservices Architecture". In: *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2017)*. Submitted for review (CORE Rank A). Madrid, Spain, June 2017.

Other Publications

The background research to this thesis has led to the following publications:

Conference Proceedings

[KSF14] Amin M Khan, Mennan Selimi, and Felix Freitag. "Towards Distributed Architecture for Collaborative Cloud Services in Community Networks". In: *6th International Conference on Intelligent Networking and Collaborative Systems (INCoS 2014)*. Salerno, Italy: IEEE, Sept. 2014.

[SF14a] Mennan Selimi and Felix Freitag. “Tahoe-LAFS Distributed Storage Service in Community Network Clouds”. In: *2014 IEEE Fourth International Conference on Big Data and Cloud Computing (BDCloud 2014)*. 2014, pp. 17–24.

[SF14b] Mennan Selimi and Felix Freitag. “Towards Application Deployment in Community Network Clouds”. In: *14th International Conference on Computational Science and Its Applications (ICCSA 2014), Guimarães, Portugal, June 30 - July 3, 2014*. 2014, pp. 614–627.

Abstracts, Demos & Posters

[Bai+15] Roger Baig, Rodrigo Carbajales, Pau Escrich Garcia, Jorge L. Florit, Felix Freitag, Agustí Moll, Leandro Navarro, Ermanno Pietrosemoli, Roger Pueyo Centelles, Mennan Selimi, Vladimir Vlassov, and Marco Zennaro. “The cloudy distribution in community network clouds in Guifi.net”. In: *IFIP/IEEE International Symposium on Integrated Network Management, (IM 2015), Ottawa, ON, Canada, 11-15 May, 2015*. 2015, pp. 1161–1162.

[Sel+14a] Mennan Selimi, Jorge L Florit, Davide Vega, Roc Meseguer, Ester Lopez, Amin M Khan, Axel Neumann, Felix Freitag, Leandro Navarro, Roger Baig, Pau Escrich, Agusti Moll, Roger Pueyo Centelles, Ivan Vilata, Marc Aymerich, and Santiago Lamora. “Cloud-Based Extension for Community-Lab”. In: *22nd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2014)*. (CORE Rank A). Paris, France: IEEE, Sept. 2014, pp. 502–505.

[Sel+14b] Mennan Selimi, Felix Freitag, Daniel Martí, Roger Pueyo Centelles, Pau Escrich Garcia, and Roger Baig. “Experiences with distributed heterogeneous clouds over community networks”. In: *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing (DCC 2014), Chicago, Illinois, USA, August 18, 2014*. (CORE Rank A). 2014, pp. 39–40.

Contents

List of Figures

List of Tables

1 Introduction

Since early 2000s, community networks (CNs) or "*Do-It-Yourself*'' networks have gained momentum in response to the growing demands for network connectivity in rural and urban communities. The main singularity of CNs is that they are built "bottom-up'', mixing wireless and wired links, with communities of citizens building, operating and managing the network. The result of this open, agglomerative process is a very heterogeneous network, with self-managing links and devices. For instance, devices are typically "low-tech", built entirely by off-the-shelf hardware and open source software, which communicate over wireless links. This poses several challenges, such as the lack of service guarantees, inefficient use of the available resources, and absence of security, to name a few.

These challenges have not precluded CNs from flourishing around. For instance, `Guifi.net`*, located in the Catalonia region of Spain, is a successful example of this paradigm. `Guifi.net` is defined as an open, free and neutral CN built by its members. That is, citizens and organizations pool their

*http://guifi.net/

resources and coordinate efforts to build and operate a local network infrastructure. `Guifi.net` was born in 2004, and until today, it has grown into a network of more than $32,000$ operational nodes. This makes it the largest CN worldwide [Bai+15].

Services running in community networks face specific challenges intrinsic to these infrastructures, such as the limited capacity of nodes and links, their dynamics and geographic distribution. Because `Guifi.net` nodes are geographically distributed, given a set of local services, we need to decide where these services should be placed in a network. Obviously, without taking into account the underlying network resources, a service may suffer from poor performance, e.g, by sending large amounts of data across slow wireless links while faster and more reliable links remain underutilized.

1.1 Motivation

`Guifi.net` is a "crowdsourced network" i.e., a network infrastructure built by citizens and organisations who pool their resources and coordinate their efforts to make these networks happen. In these networks the infrastructure is established by the participants and is managed as a common resource [Bai+16]. `Guifi.net` is the largest and fast growing community network worldwide. Some measurable indicators are the number of nodes ($> 32,000$), the geographic scope ($> 50,000km$ of links), Internet traffic etc. Regarding the Internet traffic, Figure 1.1 depicts the evolution of the total inbound (pink) and outbound (yellow) traffic to the Internet for the last two years. A mere inspection of this figure tells us that `Guifi.net` traffic has tripled($3Gbps$ peak). Traffic peaks correspond to the arrival of new users and deployment of bandwidth-hungry services in the network.

`Guifi.net` ultimate aim is to create a full digital ecosystem that covers a highly localized area. But this mission is not so simple, because a quick glance at the type of services that users demand reveals that the percentage of Internet services (proxies and tunnel-based) is higher than 50% [Sel+15c].

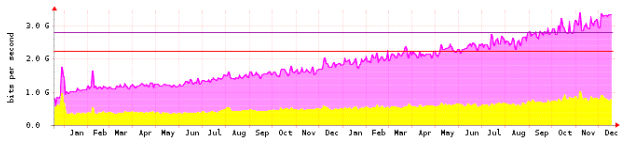


Figure 1.1: `Guifi.net` inbound and outbound traffic (Dec 2014 - Dec 2016).

This confirms that `Guifi.net` users are typically interested in mainstream Internet services, which imposes a heavy burden on the "thin'' backbone links, with users experiencing high service variability.

Among other issues, this question spurred the invention of "alternative'' service deployment models to cater for users in `Guifi.net`. One of these models was that based on *micro-clouds*[†]. A micro-cloud is nothing but a platform to deliver services to a local community of citizens within the vast CN. Services can be of any type, ranging from personal storage to video streaming and P2P-TV [Sel+15b]. Observe that this model is different from Fog computing, which extends cloud computing by introducing an intermediate layer between devices and datacenters. Micro-clouds take the opposite track, by putting services closer to consumers, so that no further or minimal action takes place in Internet. The idea is to tap into the shorter, faster connectivity between users to deliver a better service and alleviate overload in the backbone links.

Micro-clouds differ in which hardware resources they incorporate, ranging from resource constrained devices such as home gateways, routers, embedded devices etc., to desktop-style hardware used in office mesh networks. Given the characteristics of communication over a wireless channel, unreliable network and user devices at non-optimal locations, the physical topology of the mesh network where the *micro-clouds* are deployed is in a constant state of flux. In order to guarantee that the network is operational at all time, it is necessary to continuously adapt the logical configuration of the network, e.g., routing paths and neighborhood lists to the conditions in the physical world.

[†]http://cloudy.community/

This has been the research focus of the past decade i.e., optimizing the routing of packets between the nodes of the mesh network, thus resulting in a great variety of reactive, proactive and hybrid routing protocols [NLN15] [03].

Taking a more service-centric view on mesh networks, the distinction between clients and servers still exists as part of the logical network structure whenever certain nodes request services provided by other nodes. Therefore, the question arises whether the performance of the mesh network (i.e., service performance) such as Guifi.net can be optimized by carefully choosing exactly which of the nodes it is going to host the particular service. Key factors to take into account when answering this question are the connectivity between individual nodes, availability of the nodes, service demand and the suitability of services for being migrated between nodes.

The question of determining the location of deployment for *micro-clouds* is referred to as the *service placement problem*, whose goal is to establish an optimal or near-optimal *service configuration*, i.e., selection of nodes to host the instances of the service which is optimal in regard to some service-specific metric.

1.2 Problem Statement

The network topology in a wireless community network such as `Guifi.net` is organic and different with respect to conventional ISP networks. The current network deployment model in this mesh network is based on geographic singularities rather than QoS. The resources in the network are not uniformly distributed [CNE13a]. Furthermore, wireless links are with asymmetric quality for services and there is a highly skewed traffic pattern and highly skewed bandwidth distribution. Figure 1.2 depicts the bandwidth distribution in `Guifi.net` for the first five months of 2016. Figure reveals that 50% of links have a bandwidth smaller than 10 Mbps and the other 50% of the links in the network have a bandwidth between $10 - 100$ Mbps.

The highly skewed bandwidth distribution at `Guifi.net` is not the case in

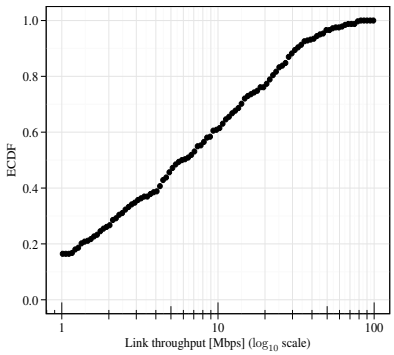


Figure 1.2: `Guifi.net` bandwidth distribution

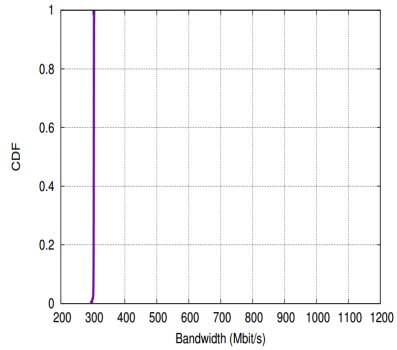


Figure 1.3: `Rackspace` bandwidth distribution

the data center network such as Rackspace or Amazon EC2 [LaC13]. Figure 1.3 reveals that in Rackspace data center there is very little spatial variation. In fact, every path has a throughput of almost exactly 300 Mbps. This implies that if a tenant were placing a single service on the Rackspace network, there would be virtually no variation for the service placement algorithms to exploit.

Furthermore, in contrast to the data center environment, the infrastructure in `Guifi.net` is highly unreliable and heterogeneous. Devices and the network are very heterogeneous comparing to data centers where they are homogeneous. In terms of demand distribution, in community networks the demand comes directly from the edge so there are no central load balancers as in the cluster environments.All these factors makes the problem of service placement even more challenging

The main challenge when deploying *micro-clouds* in the mesh network is that of the optimal placement of *micro-clouds* within the mesh network to overcome suboptimal performance i.e., the existing in-place strategy performance. Obviously, a placement algorithm that is agnostic to the state of the underlying network may lead to important inefficiencies. Although conceptually straightforward, it is challenging to calculate an optimal decision due to the dynamic nature of CNs and usage patterns.

The problem of service placement in *micro-clouds* deployed over the com-

munity mesh networks can be stated as follows: *"Given a service and network graph, how to place a service on a network as to maximize user QoS and QoE, while satisfying a required level of availability for each node (N) and considering a maximum of k service copies ?*

The cost function to maximize user QoS or QoE may include metrics such as network bandwidth, service overlay diameter or other service-dependent quality metrics. The choice of the cost function is mandated by the *service placement policy*. A placement policy states the goal that a service placement system to achieve. The type and the consequences of placement policy vary depending on the service type in the mesh network. A very fundamental placement policy and in the fact the one we will consider mostly in this work is the overall maximizing of the bandwidth required for the service provisioning. Other goals, may be pursued in more specialized service scenarios. For example, in urban wireless mesh network the placement policy may aim at a reduction of service access time (i.e., client response time). Alternatively, in mesh network with nodes in non-optimal locations, a regionally diverse service placement of service instances may be preferable.

Because of this, in order to adapt to zone-specific service demand, it is advantageous if the service can be provided by multiple *service instances* each of which is hosted on a different *micro-cloud* nodes. A service instance is an exact copy of the software component that provides the service, including the executable binary and the application-level data. Each of these service instances is capable of providing the complete service on its own. These service instances do not differ between them except for which node of the mesh network they are hosted on.

The concept of service instances gives rice to a distinction between *centralized* and *distributed* services. It is technically infeasible to create multiple instances for for the centralized services hence there is only a single service instance (e.g., identical with the *server* in traditional client/server architecture). On other side, for the distributed services is possible to create multiple

instances (i.e., storage nodes in the distributed storage service or source node in the video streaming service). However, they incur an additional overhead in the mesh network which is not present in the case of centralized services.

1.2.1 Assumptions

Deploying service placement algorithms in community network *micro-clouds* relies upon several assumptions about the context in which the wireless mesh networks are deployed and about the capabilities of the nodes. The assumptions are as follows:

- **Bounded heterogeneity of devices:** This work takes the network characteristics into account, while most of the service placement approaches generally consider only device characteristics (CPU, memory etc). However, the heterogeneity of the devices with regard to their capabilities needs to be bounded. We assume, most, if not all, nodes in the network possess sufficient resources (CPU and memory) to host a service instance.

- **Cooperation between nodes**: Since we are dealing with a contributory computing environment like wireless community networks, service placement relies upon the assumption that the nodes are willing to cooperate with each other in order to achieve a common goal. This assumption is used also in the core of routing protocols that are used and service placement applies to the area of service provisioning.

1.3 Research Questions

Measuring the performance of services in community network *micro-clouds* is very important in order to guarantee that the services will run successfully and not disrupt the proper function of the network. The approach for performance assessment of services in community network *micro-clouds* is to set

the experimental conditions as seen from the end user: experiment in production community networks, focus on metrics that are of interest for end users and deploy services on real nodes integrated in community networks. To this end, the first question that needs to be addressed is the following one:

Q1: `Is it feasible to run services in community network micro-clouds ?`

Not all services can be deployed in the community network *micro-clouds*. However, those services that can be deployed have different QoS requirements. For instance, bandwidth intensive services (e.g., distributed storage) and latency sensitive services (e.g., live-video streaming) can operate successfully upon different workloads. Hence, the following question emerges as a consequence:

Q1.1: `What are the service workload upper bounds for successful operation of bandwidth-intensive and latency-sensitive services ?`

Metrics that quantify the success of a different service type of operation are important to be included in the service placement algorithms. We address this issue by answering the following question:

Q1.2: `Which metrics should be applied for analysis ?`

The placement of the components in a distributed services depends largely on the interactions and the semantics between the service components. Based on that, services that require intensive inter-component communication (e.g streaming service), can perform better if the replicas (service components) are placed close to each other in high capacity links. On other side, bandwidth-intensive services (e.g., distributed storage, video on-demand) can perform much better if their replicas are as close as possible to their final users (e.g.

overall reduction of bandwidth for service provisioning). The service components of bandwidth-intensive and latency-sensitive services deployed create a service overlay graph. Therefore, we should tackle the following question:

Q2: `What is the impact of the service overlay diameter on the service performance ?`

As services become more network-intensive, they can become bottle-necked by the network, even in well-provisioned clouds. In the case of community network *micro-clouds*, network awareness is even more critical due to the limited capacity of nodes and links, and an unpredictable network performance. Without a network-aware system for placing services, locations with poor network paths may be chosen while locations with faster, more reliable paths remain unused, resulting ultimately in a poor user experience. To fulfil this need, we should tackle the following question:

Q3: `Given a community network micro-cloud infrastructure, what is an effective and low-complexity service placement solution that maximises end-to-end performance ?`

1.4 Contributions

In this section, we outline the major contributions of the thesis by mapping each contribution to the associated research question. The major contributions of the thesis are listed as follows:

`C1:` A performance evaluation of a distributed storage service, live-video streaming and discovery service in a community network *micro-cloud* platform. First, this contribution identifies the requirements of deploying these type of services in *micro-cloud* environment. This part specifically addresses the question **Q1**. Second, we characterize and define workload upper bounds for successful operation of such services and this addresses the question **Q1.1**.

Third, we conduct cross-layer analysis and optimizations on the service level to improve the service performance in a community network *micro-cloud* environment, which addresses the question **Q1.2**. We discuss this contribution in Chapter 3.

`C2:` A service placement algorithm *PASP* that explores different placements searching for the local minimal service overlay diameter, while at the same time fulfilling different service type quality parameters. The algorithms finds the minimum possible distance in terms of number of hops between two furthest selected resources (i.e. service components), without the need to verify the whole solution space. In addition to minimizing service overlay diameter, the *PASP* exploratory algorithm considers the latency and availability metric for the latency-ensitive services and closeness metric for bandwidth-intensive services. This contribution specifically addresses the question **Q2**, and we discuss it in Chapter 4.

`C3:` A placement heuristic called *BASP* (Bandwidth and Availability-aware Service Placement), which uses the state of the underlying community network to optimize service deployment. In particular, it considers two sources of information: i) network bandwidth and ii) device availability to make optimized decisions. Compared with brute-force search, which it takes of the order of hours to complete, *BASP* runs much faster; it just takes a few seconds, while achieving equally good results. This contribution addresses the question **Q3** and we discuss it in Chapter 5.

1.5 Scope

The goal of this research project is to assess the current service placement in effect in a representative community network, analyse its inefficiencies, and propose and evaluate a feasible and effective solution to improve it. The algorithms are designed to interact closely with the domains of routing and

service discovery. There are several other, closely related areas of research, which we will not consider in depth in this work:

- **Incentives for cooperation:** it is out of scope to establish under which motivation nodes of the *micro-clouds* should decide to host service or forward packets for other nodes. The assumption that nodes are willing to cooperate to achieve a common goal is widely used in research into wireless mesh network. However, we we can reference the work of Khan [KBF13] that is particular done for our scenario.
- **Security:** It is beyond the scope of our current work to make the system or algorithms robust against attacks from malicious nodes.

1.6 Outline of the Thesis

Figure 1.4 shows the chapters where we discuss the contributions. Furthermore, the figure shows also the publications accepted and how they match with the chapters. Based on that, this thesis work is structured as follows:

We begin with a review of the fundamentals of wireless community networks and service placement in Chapter 2. In this chapter, we give brief definitions for the terms that we have introduced informally in the whole thesis. We also present an in-depth review of the state of the art of service placement problem and classify current proposals by their architectural approach and applicability environment.

Chapter 3 presents the current state of service deployment in Guifi.net community network and the performance assessment of three type of popular services in these environments. This chapter presents the first part of our contribution, which is the feasibility and in-depth performance assessment of distributed storage, video streaming and discovery service.

In Chapter 4 we present our *PASP* algorithm, i.e., exploratory algorithm that finds all the optimal and sub-optimal service overlay placements. First we study the effectiveness of our approach in simulations using real-world

node and usage traces from Guifi.net nodes. Subsequently, we deploy our algorithm, driven by these findings, in a real production community network and quantify the performance and effects of our algorithm with a distributed storage service.

In Chapter 5 we present our low-complexity service placement heuristic called *BASP* that maximises the bandwidth allocation when deploying a community network *micro-clouds*. We present algorithmic details, analyse its complexity, and carefully evaluate its performance with realistic settings.

Chapter 6 concludes the thesis and indicates future directions.

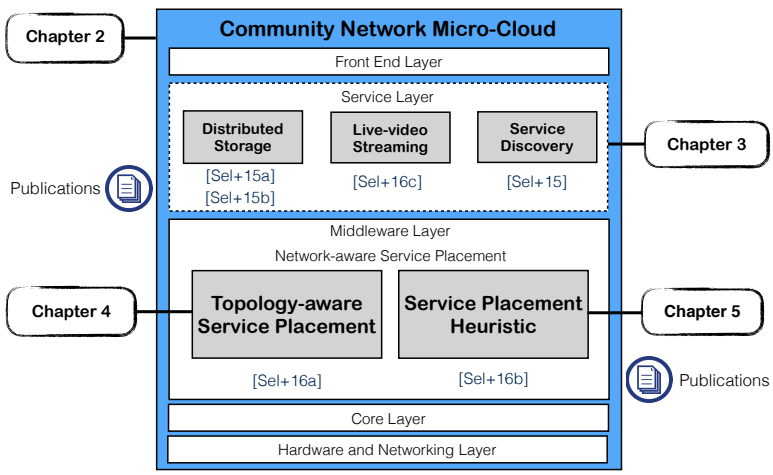


Figure 1.4: Outline of the thesis

2 Background

Community network *micro-clouds* are a social collective model, and need contribution from its participants for its sustainability and growth. This requires addressing the service deployment requirements and the challenging problem of effective service placement. In this chapter, first we give brief definitions for the terms introduced in the thesis. Then, we present an in-depth review of the state of the art of service placement problem and classify current proposals by their architectural approach and applicability environment.

2.1 Definitions

We start by introducing the terms used throughout this chapter and the rest of the thesis.

Community Network (CN) Decentralized and self-organized communication networks built and operated by citizens for citizens. In these networks the infrastructure is established by the participants and is managed as a common resource. The infrastructure consists of mesh routers, mesh clients

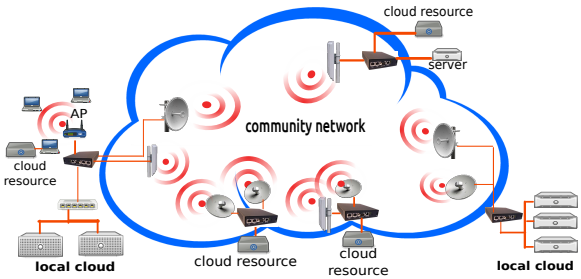


Figure 2.1: Community network resources

Figure 2.2: Micro-cloud resources

and optionally gateways as shown in Figure 2.1. The routers (i.e., outdoor routers) communicate with each other via radio transmissions and employ special-purpose routing protocols as BMX6 or OLSR [NLN15]. Mesh clients access the network via one of the routers, while gateways provide connectivity to the Internet. Client nodes consists of home gateways, laptops or desktop PCs. The main goal of community networks is to satisfy community's demand for Internet access and information and technology services. Most of them are based on Wi-FI technology.

Community Network Micro-Cloud (CNMC) Built on top of the community networks. In this model, a cloud is deployed closer to community network users and other existing infrastructure. The *micro-cloud* is deployed over a single or set of user nodes, and comparing to the public clouds it has a smaller scale, so one still gets high performance due to locality and control over application placement. The devices forming *micro-clouds* are co-located in either users homes (as home gateways, routers, laptops etc., as shown in Figure 2.2) or within other infrastructures distributed in the community networks. The concept of *micro-clouds* can also be introduced in order to split deployed community network nodes into different groups. For instance, a *micro-cloud* can refer to these nodes which are within the same service announcement and discovery domain. Different criteria can be applied to determine to which *micro-cloud* a node belongs to. Applying technical criteria (e.g. RTT, band-

width, number of hops, resource characteristics) for *micro-cloud* assignment is a possibility to optimize the performance of several applications. But also social criteria may be used, e.g. bringing in a *micro-cloud* cloud resources together from users which are socially close may improve acceptance, the willingness to share resources and to maintain the infrastructure.

Service A service is a software component executed on one or several nodes of the wireless community network. It consists of both service-specific logic and state. A service is accessed by local or remote clients by the means of issuing service request that, in case of remote clients are transmitted across the network. In our case the services are running in a Docker and LXC containers. Several attributes of a service are of special interest in the context of service placement:

- **Centralized vs. distributed services** The service placement is applicable to both centralized and distributed services. For centralized services i.e., running on one node, service placement it controls which node should host the service. For distributed services, the service placement algorithm it also manages the granularity with which the service is to be split up, i.e, the number of service components that are to be distributed in the network.

- **Monolithic vs. composite services** If a service can be decomposed into multiple independently deployable subservices, each of which contributes a different aspect of the overall service, then it is a *composite service*. In contrast, a *monolithic* service cannot be split into subservices either due to semantics or implementation concerns. The main focus of this work is on monolithic services.

Service overlay When service components are deployed on the *micro-clouds* they create a service overlay graph. Our service placement algorithm searches

in a large space of solutions, looking for those that minimize the overlay diameter (i.e., number of hops) between two selected resources.

Network Topology The physical network topology (i.e., *network graph*) refers to the connectivity between nodes that form the wireless community network. The network graph is subject to continues change due to node churn, mobility and changing properties of wireless links. The *network graph* of `Guifi.net` has a mesh-based topology. The logical network topology (i.e., *service graph*) consists of links created from communication of services.

Offline and Online Service Placement Throughout this thesis, we say that a service placement is offline when our goal is to place a single or a set of service graphs "in one shot". In contrast, an online service placement is the case where we have an incoming stream of service graphs, which have to be sequentially placed onto the network graph as each service graph arrives.

2.2 Community Clouds

Carving our path towards community clouds in community networks, we must consider the cloud essential characteristics, as described in [MG11]. *Broad network access* is already offered by the community networks and *resource pooling* should be an outcome of the resource sharing described above. *Measured services* are very important in order to guarantee that the services will not disrupt the proper function of the network. *On-demand self-service* is a higher-level concept concerning the responsiveness and the transparency of the system; thus, this is an important feature but of secondary priority. Similarly, *rapid elasticity* of the resources offered is a welcome property; yet, it should not be considered an essential one due to its complexity because of the highly distributed environment.

The idea of collaboratively built community clouds follows earlier distributed voluntary computing platforms, such as BOINC [And04], Fold-

ing@home [Beb+09], PlanetLab [Chu+03], and Seattle [Cap+09], which largely rely on altruistic contributions of resources from users, functioning as research platforms. There are only a few research proposals for community cloud computing [MB09], and most of them do not go beyond the architecture level, whereas very few present a practical implementation.

The Cloud@Home [DP12] project aims to harvest resources from the community for meeting the peaks in demand, working with public, private, and hybrid clouds to form cloud federations. The Clouds@Home [Yi+11] project focuses on providing guaranteed performance and ensuring quality of service (QoS), even when using volatile Internet volunteered resources. The P2PCS [BMT12] project has built a prototype implementation of a decentralised peer-to-peer cloud system. It uses Java JRMI technology and builds an IaaS system that provides very basic support for creating and managing virtual machines. These implementations, to our knowledge, are not actually deployed inside real community networks, considering the infrastructure diversity, and are not aiming to satisfy end-user needs.

Social cloud computing [Cha+12] is a relevant research field that takes advantage of the trust relationships between members of social networks to motivate contribution towards a cloud storage service. Users trade their excess capacity to earn virtual currency and credits that they can utilise later, and consumers submit feedback about the providers after each transaction, which is used to maintain the reputation of each user. Social clouds have been deployed in the CometCloud framework by federating resources from multiple cloud providers [Pun+13]. The social compute cloud [Cat+14], implemented as an extension of the Seattle platform [Cap+09], enables the sharing of infrastructure resources between friends connected through social networks and explores bidirectional preference-based resource allocation.

Among federated cloud infrastructures, Gall et al. [GSF13] have explored how an InterCloud architecture [BRC10] can be adapted to community clouds. Further, Esposito et al. [Esp+13] presented a flexible federated Cloud archi-

tecture based on a scalable 'publish and subscribe' middleware for dynamic and transparent interconnection between different providers. Moreover, Zhao et al. [ZLL14] explored efficient and fair resource sharing among the participants in community-based cloud systems. In addition, Jang et al. [Jan+14] implemented personal clouds that federate local, nearby, and remote cloud resources to enhance the services available on mobile devices.

2.3 Service Performance Evaluation

Development and deployment of services in *micro-clouds* it can be very challenging. The feasibility of running services in *micro-clouds* can be demonstrated by carefully performing measurements taking into account different data workloads on these cloud infrastructure. The services used in the thesis are selected according to their potential relevant for community network users. The most popular open source services we are taking into account are the distributed storage service, live-video streaming and discovery service [Sel+15c].

2.3.1 Distributed Storage Service

After basic connectivity, storage is the most general service being fundamental for cloud take-up in community network scenarios. In terms of providing cloud storage services in WAN settings, Chen's paper [Che+13] is the most relevant to our work. The authors deployed open source distributed storage services such as Tahoe-LAFS [WW08], QFS [16m], and Swift [16j] in a multi-site environment and measured the impact of WAN characteristics on these storage systems. The authors deployed their experiments on a multi-site data center with very different characteristics to our scenario. Our approach is to assess the distributed storage service (i.e., Tahoe-LAFS) performance in the real context of community networks, and we use heterogeneous and less powerful machines as storage nodes.

The authors in [Gra+13] present a measurement study of a few Personal

Cloud solutions such as DropBox, Box, and SugarSync. The authors examine central aspects of these Personal Cloud storage services to characterize their performance, with emphasis on the data transfers. They report that they found interesting insights such as the high variability in transfer performance depending on the geographic location; the type of traffic, namely inbound or outbound; the file size; and the hour of the day. Their findings regarding the impact of location on the performance is relevant for our work to better understand network dependence of distributed storage services.

Another work [Tse+12] implements a distributed file system for Apache Hadoop. The original Hadoop distributed file system is replaced with the Tahoe-LAFS cloud storage. The authors investigated the total transmission rate and download time with two different file sizes. Their experiment showed that the file system accomplishes a fault-tolerant cloud storage system even when parts of storage nodes had failed. However in the experiments only three storage nodes and one introducer node of Tahoe-LAFS were used, and their experiments were run in a local context, which is an unrealistic setting for our scenario. Another paper [SD12] evaluates XtreemFS [16s], Ceph [16b], GlusterFS [16d] and SheepDog [16p], using them as virtual disk image stores in a large-scale virtual machine hosting environment. The StarBED testbed with powerful machines is used for their experiments. Differently, we target a distributed and heterogeneous set of storage nodes.

The paper of Roman [10] evaluates the performance of XtreemFS under the IO load produced by enterprise applications. They suggest that XtreemFS has a good potential to support transactional IO load in distributed environments, demonstrating good performance of read operations and scalability in general. XtreemFS is an alternative candidate for implementing a storage service upon. Tahoe-LAFS, however, more strongly addresses fault-tolerance, privacy and security requirements.

From the review of the related work it can be seen that the experimental studies regarding the distributed storage, were not conducted in the context

of community networks. In our work, we emphasized the usage of distributed storage services, i.e., Tahoe-LAFS, XtreemFS etc, in a real deployment within community network clouds, to understand its performance and operational feasibility under real network conditions.

2.3.2 Live-video Streaming Service

The work of Baldesi et al. [BML14a; BML14b], evaluates PeerStreamer [16k], a P2P video streaming platform, on the Community-Lab, the wireless community network (WCN) testbed of the EU FIRE project CONFINE [14b]. Their experiments highlight the feasibility of P2P video streaming, but they also show that the streaming platform must be tailored ad-hoc for the WCN itself to be able to fully adapt and exploit its features and overcome its limitations. However they evaluated with a limited number of nodes (16 Guifi.net nodes), which were located in the city of Barcelona and they do not use live-video streaming. A recent PhD dissertation [Ala13] includes some discussion on P2P streaming on WCNs, but does not elaborate on live streaming, but consider streaming of Video on Demand (VoD) retrieval.

Another work [Tra+12] studies different strategies to choose neighbours in a P2P-TV system (PeerStreamer). The authors evaluate PeerStreamer on a cluster and on Planetlab. In wireless networks PULLCAST [RC13], is a cooperative protocol for multicast systems, where nodes receive video chunks via multicast from a streaming point, and cooperate at the application level, by building a local, lightweight, P2P overlay that supports unicast recovery of chunks not correctly received via multicast.

The impact of uncooperative peers on video discontinuity and latency during live video streaming using PlanetLab is studied in [Oli+13]. The paper in [Cou+11] investigates the impact of peer bandwidth heterogeneity on the performance of a mesh based P2P system for live streaming.

In our work we emphasis on studying the video streaming applications in a real deployment scenario within community networks, in order to understand

their performance and operation feasibility under real network conditions.

2.3.3 Service Discovery

In terms of examining the dependability aspects of decentralized service discovery concepts in unreliable networks, Dittrich's and Salfner's paper [DS10] is the most relevant to our work. The authors evaluate the responsiveness of domain name system (DNS) based service discovery under influence of packet loss and with up to 50 service instances. Their empirical results show that the responsiveness of the used service discovery mechanisms decreases dramatically with moderate packet loss of around 20 percent. However their experimental evaluation is based on simulations and has not been applied to wireless scenarios. The research described in [DMQ07] presents an alternative approach to extend the limit of Zeroconf beyond the local link. Here, the robustness of existing discovery mechanism is evaluated under increasing failure intensity. However, responsiveness is not covered in particular. The z2z toolkit [Lee+07] combines the Zeroconf with the scalability of DHT-based peer-to-peer networks allowing services to reach beyond local links. z2z connects multiple Zeroconf subnets using OpenDHT. Robustness of service discovery with respect to discovery delay times is addressed in [Oh+04]. The work of [CG06] describes and compares through simulation the performance of service discovery of two IETF proposals of distributed DNS: Multicast DNS and LLMNR (Link-Local Multicast Name Resolution). The authors propose simple improvements that reduce the traffic generated, and so the power consumption.

In wireless mesh network settings, the work of Wirtz [Wir+12] proposes DLSD (DHT-based Localized Service Discovery), a hierarchy of localized DHT address spaces that enable localized provision and discovery of services and data. Another work[Dit+14] proposes a stochastic model family to evaluate the user-perceived responsiveness of service discovery, the probability to find providers within a deadline, even in the presence of faults. From the review of

the related work it can be seen that the reviewed experimental studies related to service discovery were not conducted in the context of the community networks, which we address as scenario.

2.4 Service Placement

Service placement is a key function of cloud management systems. Typically, by monitoring all the physical and virtual resources on a system, service placement aims to balance load through the allocation, migration and replication of tasks. We look at the service placement in three different environments: data centre, distributed clouds and wireless networks.

2.4.1 Service Placement in Data Centre Environment

Choreo [LaC13] is a measurement-based method for placing applications in the cloud infrastructures to minimize an objective function such as application completion time. Choreo makes fast measurements of cloud networks using packet trains as well as other methods, profiles application network demands using a machine-learning algorithm, and places applications using a greedy heuristic, which in practice is much more efficient than finding an optimal solution. In [Her10] the authors proposed an optimal allocation solution for ambient intelligence environments using tasks replication to avoid network performance degradation. Volley [Aga+10] is a system that performs automatic data placement across geographically distributed datacenters of Microsoft. Volley analyzes the logs or requests using an iterative optimization algorithm based on data access patterns and client locations, and outputs migration recommendations back to the cloud service. A large body of work of service placement has been devoted to finding heuristic solutions [Gha+14].

Most of the work in the data center environment is not applicable to our case because we have a strong heterogeneity given by the limited capacity of nodes and links, as well as asymmetric quality of wireless links. The differ-

ence/asymmetry in the link capacities across the network makes the service placement a very different problem than in a mostly homogeneous cloud datacenter. Our measurement results demonstrate that 25% of the links have a symmetry deviation higher than 40% [**SelimiArxiv2099**].

2.4.2 Service Placement in Decentralized Clouds

When the service placement algorithms decide how the communication between computation entities is routed in the substrate network, then we speak of network-aware service placement, i.e., closely tied to Virtual Network Embedding (VNE).

There are few works that provides service placement in distributed clouds with network-aware capabilities. The work in [SG12] proposes efficient algorithms for the placement of services in distributed cloud environment. The algorithms need input on the status of the network, computational resources and data resources which are matched to application requirements. In [KIH12] authors propose a selection algorithm to allocate resources for service-oriented applications and the work in [AL12] focuses on resource allocation in distributed small datacenters. Another example of a network-aware approach is the work from Moens in [Moe+14] which employs an Service Oriented Architecture (SOA), where applications are constructed as a collection of services. Their approach performs node and link mapping simultaneously. The work in [SBL15] extends the work of Moens et al. in wireless settings taking into account IoT. Another work is Mycocloud [Dub+15], which provides elasticity through self-organized service placement in decentralized clouds.

The recent work in [Tär+16] simultaneously and holistically take into account rapid user mobility and vast resource cost- and capacity-heterogeneous infrastructures when placing applications in Mobile Cloud Networks (MCN). Based on their proposed system model, a globally optimal placement of static and mobile applications is designed. Their optimal solution achieves a 25% reduction in cost compared to the naive methods.

Most of the work in the distributed clouds consider micro-datacenter, where in our case the CN micro-clouds consists of constraint/low-power devices such us home gateways. Furthermore, in our case we have a partial information regarding the computational devices, so their approaches are not applicable to our environment.

2.4.3 Service Placement in Wireless Networks

To the best of our knowledge, not many works exist regarding the service placement in wireless environment. Some of the works that we find similarities with our work in this thesis are the following below.

The authors in [Wit10] propose a service placement framework as a novel approach to service placement in wireless ad hoc network. Their SP_i framework takes advantage of the interdependenices between service placement, service discovery and the routing of service requests to minimize signaling overhead. They propose two service placement algorithms: one for centralized services with a single instance, and one algorithm for distributed services with variable number of instances. The SP_i framework employs these algorithms to optimize the number and location of service instances based on usage statistics and a partial network topology derived from routing information. The examine the performance of their algorithms in simulations, emulations and real-world experiments on a IEEE 802.11 wireless testbed.

Another work in [Cab14] tries to optimize the resource selection for service allocation in the contributory computing model. The claim that by using historical availability behavior to select nodes in the system, can lead to higher service availability levels. Thus, user impression is improved and more users and services could be attracted to contributory environments. Furthermore, they design a network-aware methodology to allocate service replicas in a network graph aiming to minimize the distances from each client to its closest service replica. They model the problem as a Facility Location Problem, on which service replicas are facilities (with a set up cost) and the clients in the

network are customers. Then, the goal is to minimize the sum of distances and set up costs of the selected replicas.

The work in [Nov+15] analyzes network topology and service dependencies, and combined with set of system constraints determines the placement of services within the wireless network. The authors use a multi-layer model to represent a service-based system embedded in a network topology and then apply an optimization algorithm to this model to find where best to place or reposition the services as the network topology and workload on the services changes. This technique improves the reachability of services when compared to uninformed placements.

The work of Davide et.al [Veg+14] introduces a service allocation algorithm, that from being optimal in computation time, provides near-optimal overlay allocations without the need to verify the whole solution space. This is work is related to ours since it is done in Guifi.net community network. Their algorithm uses the static data from the Guifi.net community network to identify node traits and propose several algorithms that minimize the coordination and overlay cost along a network.

The focus of our work in this thesis however is to design a low-complexity service placement heuristic for community network *micro-clouds* to maximise bandwidth and improve user QoS and QoE.

2.4.4 Service Placement through Migration

Regarding the service migration in distributed edge clouds, few works came out recently. The authors in [Wan+15b] and [Wan+15a] study the dynamic service migration problem in mobile edge-clouds that host cloud-based services at the network edge. They formulate a sequential decision making problem for service migration using the framework of Markov Decision Process (MDP) and illustrate the effectiveness of their approach by simulation using real-world mobility traces of taxis in San Francisco. The work in [Urg+15] studies when services should be migrated in response to user mobility and

demand variation. Another work [Mac+16] proposes a three-layer framework for migrating running applications that are encapsulated either in virtual machines (VMs) or containers. They evaluate the migration performance of various real applications under the proposed framework.

3

Micro-Cloud Service Performance

Internet and communication technologies have lowered the costs to collaborate for communities, leading to new services like user-generated content and social computing and, through collaboration, collectively built infrastructures, such as community networks. Internet access is often considered the main service of community networks, but the provision of services of local interest within the network is a unique opportunity for community networks, which is currently predominantly unexplored. The consolidation of today's cloud technologies offers community networks the possibility to collectively build community *micro-clouds*, building upon user-provided networks, and extending towards an ecosystem of cloud services. In this chapter first we present the current state of service deployment in community networks through a study of `Guifi.net`, and identify the popular services within this network. Then, we conduct real deployments of *micro-clouds* in the `Guifi.net` community network and evaluate cloud-based applications such as distributed storage, live-video streaming and service discovery. This deployment experience supports the feasibility of community *micro-clouds* (i.e., research question **Q1**)

and our measurements demonstrate the performance of services and applications running in these community *micro-clouds* (i.e., research questions **Q1.1** and **Q1.2**). Our results encourage the development and operation of collaborative cloud-based services using the resources of a community network. We anticipate that such services can effectively complement commercial offers and have the potential to boost innovation in application areas in which end-user involvement is required.

3.1 Background

The predominant trend in many community networks is to use the available resources as a means to access external services provided elsewhere on the Internet. This might be seen as a contradiction to their spirit since traffic within the community network is freely available, while Internet access is charged or restricted to some extent (bandwidth limitations apply or packets exit through proxies). Ubiquitous cloud services, such as private data storage and backup, instant messaging, media sharing, social networking, etc., are generally operated by well-known Internet service vendors. Community network participants are thus increasingly affected by the problems and disadvantages of this model (privacy, security, property, legislation, dependency, etc.).

In some cases, Internet cloud services have equivalent alternatives that are owned and operated at the community level; in other cases, however, there are no locally driven alternatives, yet. Possible reasons for the absence of these community-owned services can be found in the difficulty to deploy such services and the shortage or lack of individuals, organisations, or companies interested in the commercial operation of these services.

As we describe next, since community networks have become popular, there have been efforts to develop and promote different services and applications from within community networks but without significant adoption. One of the reasons identified is the technological barrier. Before providing content,

users willing to share information with the community must first take care of the technical aspects, such as the deployment of a server with a set of services. For example, the key characteristic of the Guinux [16h] distribution, explained below, was a set of scripts that automatised the configuration process. End users were only asked for a few parameters, such as their e-mail address and the node identifier. Shortly after the distribution was made available, the number of end users sharing resources proliferated. Thus, it became clear that lowering (or removing) the technological entry barrier encouraged users to provide more services and share their resources with the community. Nevertheless, community networks are still typically used to access external Internet services, as presented later. We believe that this is a result of the lack in number, diversity, and user-friendliness of services as well as performance disadvantages of non-commercial distributed applications compared to Internet cloud services.

3.1.1 Current State of Service Deployment in Guifi.net

To obtain the dimension of the current situation, we analyse the list of services published (i.e., publicly announced) by the Guifi.net community network. We do so by means of the list of services available on the Guifi.net web page for the Catalonia region of Spain, the origin and most dense location of Guifi.net [16g].

Tables 3.1 and 3.2 [16g] indicate the network-focused and user-focused services, respectively, of Guifi.net and the proportion of each service in the services offered. We consider that the number of instances of a service implies the demand of the service inside the network. Comparing the tables, we notice that the services related to the network operation itself slightly outnumber the services intended for end-users. Considering that network management is of interest only to a fragment of the network members compared to user-focused services, which could be of interest to all users, we would expect user-focused services to be more developed. Moreover, the most frequent of all

Services	Catalonia	
Network graph server	219	39.24%
DNS server	198	35.48%
NTP server	96	17.20%
Bandwidth measurement	36	6.45%
Logs server	4	0.71%
LDAP server	3	0.53%
Wake on LAN	2	0.35%
Total	**558**	

Table 3.1: List of network-focused Guifi.net services in Catalonia area

Services	Catalonia	
Proxy server (Internet access)	275	53.50%
Web pages	57	11.08%
VoIP / audio / video / chat / IM	48	9.33%
Data storage server	41	7.97%
Radio / TV stations	18	3.50%
P2P server	17	3.50%
Linux mirrors	15	2.91%
Webcam	12	2.33%
Tunnel-based Internet access	10	1.94%
Mail server	6	1.16%
Weather station	6	1.16%
Games server	5	0.97%
CVS repository	2	0.38%
Server virtualisation (VPS)	2	0.38%
Total	**514**	

Table 3.2: List of user-focused Guifi.net services in Catalonia area

the services, whether user-focused or network-focused, are the proxy services. Specifically for the user-focused services, the percentage of Internet access services (proxies and tunnel-based) is higher than 55%, confirming that the users of Guifi.net are typically interested in accessing the Internet. We can also claim that there is a diverse set of services inside Guifi.net, even though their adoption is overshadowed by Internet access.

It is important to point out that this situation is not unique to Guifi.net. Other community networks exhibit similar situations, where the network is typically used to access the Internet, and the few services available within the community network are similar to those available in Guifi.net. For instance, Elianos et al. [Eli+09] presented similar information regarding AWMN [16a]. The authors mainly focused on user-oriented services, which are quite similar to the Guifi.net services. The most popular services are web hosting, data storage, VoIP, and video streaming.

The services provided by Guifi.net can be categorised under cloud computing service models (though not following the traditional cloud elastic on-demand service approach): *IaaS*, *PaaS* and *SaaS*. Concerning *IaaS*, following the global trend, the popularity of virtualisation technologies is rising in Guifi.net. Currently, almost all critical services are run on virtualised environments, frequently using Proxmox [16l]. Guifi.net also provides specific hardware infrastructure and software, supporting virtual networks and tunnelling. Additionally, some efforts have been made in the past to provide the end users with tools to help them with the deployment and expansion of the community network from the software and services perspective. This was the case of Guinux, a GNU/Linux distribution for end users allowing them to deploy servers with services useful for community networking, namely Proxy, DNS, and SNMP (Simple Network Management Protocol) graphs servers. Similarly to *PaaS*, a diverse set of services has been deployed, such as automated node configuration, user authentication, service monitoring (servers and network), and an on-line service directory as well as network information and

administration databases. Finally, taking into account the *SaaS* model in the context of community networks, data storage services have been sporadically deployed by enthusiastic users who wanted to share some of their content (pictures, documents, etc.) with the rest of the community [Sel+14]. In some cases, users have also enabled uploading to folders, allowing other users to upload their files for sharing with the community. Despite this, it should not be considered a data storage service for end users. Moreover, Guifi.net users have developed GuifiTV [16e], a project initially conceived to harmonise the captured video formats and the content from seminars and workshops, which later included video streaming services.

The services described above are representative examples of those usually deployed in community networks. Nevertheless, both network-oriented and user-oriented services are centralised and offered by individuals. Distribution and decentralisation are concepts that are closely related to the community network philosophy; nonetheless, since centralised solutions are generally much easier to develop and deploy, in most of the cases they end up being implemented according to the classical client-server approach. As a result, basic cloud service requirements, as described in [MG11], are not fulfilled. Most importantly, there is no common pool of resources but instead a set of separate resources, since the same services are deployed independently and not coordinated in a common way. As a result, service coordination and resource sharing mechanisms are the first milestones towards creating cloud services for community networks, which are provided by the cloud framework we propose.

Our effort targets fostering the deployment of *micro-clouds* and cloud-based services on top of community networks. Based on the experiences of current community networks, providing end users with the appropriate applications has proven to be an effective way of encouraging them to use, provide, and promote services.

3.1.2 Cloudy: Community Cloud-in-a-Box

As we discussed earlier in Section 3.1, we believe that the failure of services gaining traction in community networks was largely due to the difficulty of implementing the services and for the end-users to consume these services. To overcome these issues, we provide a community cloud distribution, codenamed Cloudy [Clo16]. *Cloudy* is a tool that fosters the adoption and uptake of *micro-cloud* services among the users.

The current prototype of Cloudy implements the modules/layers shown in Figure 5.11. This distribution contains the platform and application services of the community cloud system. Cloudy is the core software of our micro-clouds, because it unifies the different tools and services of the cloud system in a Debian-based Linux distribution. Cloudy is open-source and can be downloaded from public repositories*. A Cloudy instance can be run directly on a bare metal machine or on a virtual machine. Independent of the hardware that Cloudy runs on, connectivity to other Cloudy instances is needed in order to fully exploit the potential of Cloudy.

3.1.3 Cloudy Services

Cloudy comprises a number of services, designed to help build cloud-based services in community networks. Cloudy's main components can be considered a layered stack with services residing both inside the kernel and higher up at the user-level. All of the software included in the Cloudy platform is open-source. All service accesses are assisted and managed through the main panel of the Cloudy GUI. The following three groups classify Cloudy services.

Infrastructure Services

Virtualisation is the main enabling technology for cloud computing. As such, providing community network users the resources to deploy virtual machines

*http://repo.clommunity-project.eu/images/

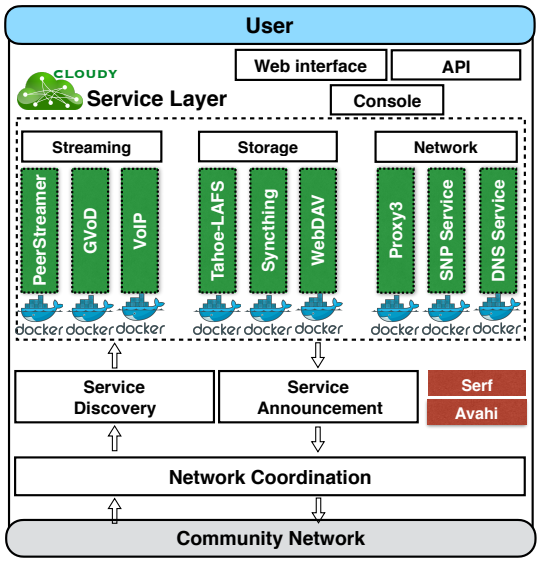


Figure 3.1: Cloudy architecture

with a few clicks is a very convenient way to bring the cloud closer to their premises. This allows the non-experienced user to focus on the services and applications themselves rather than on learning how to cope with the underlying infrastructure.

OpenVZ [15f] is an operating system-level virtualisation technology for Linux based on containers. OpenVZ allows creating multiple secure, isolated operating system instances called containers (commonly known as VPSs) on a single physical machine enabling better server utilisation and ensuring that applications do not conflict with each other. Each container performs and executes exactly like a stand-alone server (which can have root access, users, IP addressing, memory, files, etc.) and can be started and stopped independently from the others and from the host machine. OpenVZ is the preferred solution for providing virtual machines in Cloudy with low to mid-end hardware as only a negligible portion (1-2%) of the CPU resources is spent on virtualisation. The Cloudy distribution includes a script that downloads and

installs all the required OpenVZ packages in one click and Cloudy instances can be run on the virtual machines created using the OpenVZ Web Panel.

Other virtualisation methods used in Cloudy are LXC and Docker. This approach adds special support for IaaS, as the cloud nodes are able to create multiple virtual machine instances for other purposes in addition to the ones dedicated to Cloudy. The infrastructure services of Cloudy enable resource sharing inside the community network.

Service Discovery and Network Coordination Services

Cloudy provides custom decentralised services for network coordination and service discovery. Network coordination ensures visibility between the nodes that participate in the cloud. Service discovery is a crucial building block in Cloudy for enabling distributed services to be orchestrated to provide platform and application services. Service discovery is based on the network coordination component.

For service discovery, Cloudy includes a customised version of Avahi [15a] to provide decentralised service discovery at Layer 2, which is needed to discover other services that will be used to provide higher-level services. The multicast-based design does not allow the Avahi service to reach beyond the local link, which is the case in community networks, where services are spread over different nodes that belong to different broadcast domains. While in this environment, it would not be possible for Avahi packets to be directly exchanged from one node to another; this problem is solved by the network coordination component.

For the network coordination component, we adopt TincVPN [15k], a virtual private network (VPN) daemon that uses tunnelling and encryption to create a secure private Layer 2 network between hosts of different domains. This Layer 2 connectivity is needed between nodes, since they may reside on different administrative domains and even be located behind firewalls. The TincVPN is automatically installed and configured on every Cloudy node,

ready to be activated. After its activation, a VPN daemon is started in order to reach other Cloudy instances via the established Layer 2 network; thus, Avahi can communicate transparently with other nodes. To easily install and configure a system with TincVPN, a tool called Getinconf [16c] has been developed, which is integrated into Cloudy.

Cloudy also includes Serf [Ser16], a lightweight tool to announce and discover services in community networks. Serf is a decentralised solution for cluster membership, failure detection, and orchestration. It relies on an efficient and lightweight gossip protocol to communicate with other nodes that periodically exchange messages between each other. This protocol is, in practice, a fast and efficient method to share small pieces of information. An additional by-product of having this service distributed all over the community cloud is that it allows the evaluation of the quality of the point-to-point connections between different Cloudy instances. This way, Cloudy users can decide which service provider to choose based on network metrics, such as round trip time (RTT), number of hops, or packet loss. The combination of Avahi, TincVPN, Getinconf, and Serf in Cloudy facilitates the coordination of the resources and the services in the community cloud.

User Services

Platform as a Service (PaaS). Providing attractive platform services to community members, such as a distributed file system, highly available key-value store, file synchronisation, video streaming, video-on-demand, VoIP, network address translation (NAT) traversal support, and many more, is of high importance.

One of the promising services for storage is Tahoe-LAFS [WW08]. Tahoe-LAFS is a free, open, and secure cloud storage system. Tahoe-LAFS allows community users to share their storage with other members. A Tahoe-LAFS cluster consists of a set of storage nodes, client nodes, and a single coordinator node called the introducer. The storage nodes connect to the introducer and

announce their presence, and the client nodes connect to the introducer to obtain the list of all connected storage nodes [SF14]. The configuration of Tahoe-LAFS and the process of deploying a whole storage grid are assisted by the Avahi and Serf service discovery tools using the web interface of Cloudy, where the user only needs to introduce some basic information. The Tahoe-LAFS service can also be used to provide higher-level file sharing applications.

Etcd [15c], a highly available key value store for shared configuration and service discovery, and Syncthing [15j], an open-source file synchronisation client/server application, are already included in the Cloudy distribution.

Software as a Service (SaaS). Cloudy allows the user services to be present inside the community network and to be easily deployed and managed via the Cloudy interface. Users can deploy their preferred services and share them with others. One of these multimedia services included in Cloudy is PeerStreamer [15h], an open source live streaming platform. PeerStreamer includes a streaming engine for the efficient distribution of media streams, a source application for the creation of channels, and player applications to visualise the streams. Streaming is assisted by Cloudy by supporting the user in publishing a video stream or connecting to a peer (assisted by Serf or Avahi). Services that enable users to find and watch video content on-demand at any time, such as Gvod [BD10], a decentralised search service, such as Sweep [15i], and a distributed key-value store, such as CaracalDB [15b] are additional services that are part of the Cloudy.

3.2 Experiments

In this section, we explain our work on deploying and evaluating the community network *micro-clouds*. In order to have a realistic community network setting, which includes geographically distributed nodes, we have used the Community-Lab [14a] testbed nodes for setting up our community cloud infrastructure. Community-Lab is a distributed infrastructure developed by the Community Networks Testbed for the Future Internet (CONFINE) pro-

ject [Bra+13], where researchers can deploy experimental services on several nodes deployed within merged community networks. The Community-Lab testbed is currently deployed on the nodes from Guifi.net and the AWMN community networks. This allows us to run our experiments on nodes from both the community networks, which has the added advantage that we can test how Cloudy performs in a combined community network environment, as well as how Cloudy services perform over large geographical distances.

In the Community-Lab, Guifi.net and the AWMN community networks are connected on the IP layer though the Federated E-infrastructure Dedicated to European Researchers (FEDERICA) [15d], enabling the federation of both networks. Most Community-Lab nodes are built with a Jetway device that is equipped with an Intel Atom N2600 CPU, 4GB of RAM and 120GB SSD. Nodes in the Community-Lab testbed run a custom firmware (based on OpenWRT [15g]) provided by CONFINE, which allows running several container instances on one node simultaneously implemented as LXC. We deploy the Cloudy distribution in these containers on the nodes in Community-Lab.

We exhibit results from our experiments related to distributed storage, live-video streaming and discovery service.

We choose to experiment with distributed storage service, based on Tahoe-LAFS. Tahoe-LAFS has features that are very important for the community network environment, such as like data encryption at the client side, coded transmission and data dispersion among a set of storage nodes. This approach of Tahoe-LAFS results in high availability (e.g., even if some of the storage nodes are down or taken over by an attacker, the entire file system continues to function correctly, while preserving privacy and security).

Regarding the live-video streaming service, we are experimenting with PeerStreamer. PeerStreamer is an open source live P2P video streaming service, and mainly used in our Cloudy distribution as the live streaming service example. This service is built on a chunk-based stream diffusion, where peers offer a selection of the chunks that they own to some peers in their neighbour-

hood.

We choose to experiment with service discovery, based on Avahi-TincVPN and Serf, since it is a crucial building block of Cloudy that enables distributed services to be orchestrated in order to provide platform and application services. All the services inside Cloudy use these two service discovery protocols to publish and discover services. Our goal is not to compare Avahi-TincVPN and Serf, as both of them are available in Cloudy and can be used by users for different scenarios. One of them is lightweight and fast (Serf), the other is not scalable and is suitable and preferable for environments with a smaller number of nodes (Avahi-TincVPN).

3.3 Distributed Storage

After basic connectivity, storage is the most general service being fundamental for cloud take-up in community network scenarios. Allowing users in a community network to share and use the storage of other users in a reliable, secure, and privacy-preserving way, is of a great importance. For this reason, we use Tahoe-LAFS as a main storage service in Cloudy. Understanding the performance of Tahoe-LAFS from an experimental scenario that represents real use case situations is highly important because it informs the end users regarding the application performance they will receive. Such performance results are needed to pave the way for bringing applications such as Tahoe-LAFS as well as other applications into community networks.

3.3.1 Tahoe-LAFS

Tahoe-LAFS [WW08] is a decentralised storage system with provider-independent security. This feature means that the user is the only one who can view or modify disclosed data. The data and metadata in the cluster is distributed among servers using erasure coding and cryptography. The erasure coding parameters determine how many servers are used to store each

file, which is denoted as N, and how many of them are necessary for the files to be available, denoted as K. The default parameters used in Tahoe-LAFS are $K = 3$ and $N = 10$ (3-of-10). The Tahoe-LAFS cluster consists of a set of *storage nodes*, *client nodes*, and a single coordinator node called the *introducer*. The storage nodes connect to the introducer and announce their presence, and the client nodes connect to the introducer to obtain the list of all connected storage nodes. The introducer does not transfer data between clients and storage nodes, but the transfer is done directly between them. The introducer is a first-point-of-contact for new clients or new storage peers, because they need it for joining the storage grid. When the client uploads a file to the storage cluster, a unique public/private key pair is generated for that file, and the file is encrypted (on the client side prior to upload), erasure coded, and distributed across storage nodes (with enough storage space) [WW08]. The location of erasure coded shares is decided by a server selection algorithm that hashes the private key of the file to generate a distinct server permutation. Then, servers without enough storage space are removed from the permutation, and the rest of the servers are contacted in sequence and asked to hold one share of the file. To download a file, the client asks all known storage nodes to list the number of shares of that file they hold, and in the subsequent rounds (second round-trip), the client chooses which share to request based on various heuristics such as latency, node load, etc.

Bringing Tahoe-LAFS into Community-Lab: For our experiments we are using some nodes from the Community-Lab testbed. The nodes run a custom OS based on OpenWrt provided by the CONFINE project which allows running on one node several slivers simultaneously implemented as Linux containers (LXC) [15e]. A sliver is defined as the partition of the resources of a node assigned to a specific slice (group of slivers). We can think of slivers as containers inside a node. To deploy Tahoe-LAFS in the Community-Lab nodes, we use the Cloudy distribution, which contains Tahoe-LAFS, and place the introducer, storage, and client nodes of Tahoe-LAFS inside the slivers (i.e.,

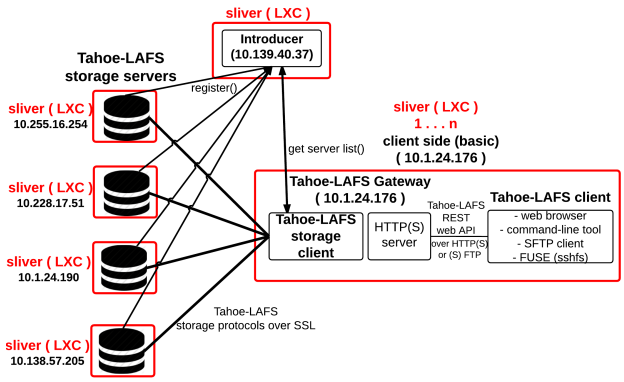


Figure 3.2: Tahoe-LAFS deployed in the Community-Lab testbed

containers) of the testbed. Figure 3.2 shows the resulting Tahoe-LAFS architecture used by our experiments in the Community-Lab testbed.

3.3.2 Experiment Setup

All tests were conducted using the IOzone [16i] cloud storage benchmark. IOzone is a filesystem benchmark tool, which generates the cloud storage workload and measures various file operations. The benchmark tests file input/output (I/O) performance of many important storage-benchmarking operations, such as read, write, re-read, re-write, random read/write, etc. We run all 13 IOzone tests and vary the file size from 64KB to 128MB and record length of 128 KB. An *-a* flag allows us to run all 13 tests. We add the *-b* flag to write the test output in binary format to a spreadsheet. We use a FUSE (Filesystem in Userspace) kernel module in combination with SSHFS (SSH Filesystem), an SFTP client that allows filesystem access via FUSE, to mount a Tahoe-LAFS directory to the local disk of the client. Tahoe's SFTP frontend includes several workarounds and extensions to make it function correctly with SSHFS. When mounting with SSHFS, we disable the cache and use direct I/O and synchronous writes and reads, using the parameters *-o cache=no, big_writes, direct_io, and sshfs_sync*. We observe that the *-o*

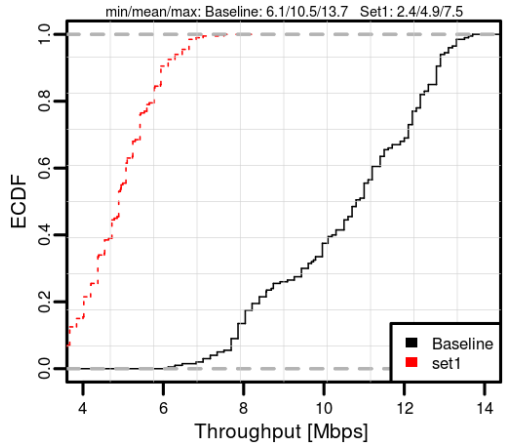


Figure 3.3: ECDF of the average throughput for two clients

big_writes option to SSHFS improves write performance without affecting the read operations [RG10]. Results presented in this work with regard to performance are measured in MB/s and are referred to as operation speed. Tests with concurrent reading and writing were not conducted.

To better understand the impact that the network imposes on a community network environment, we established a Tahoe-LAFS cluster of 30 nodes geographically distributed in the `Guifi.net` community network and connected to the outdoor routers [CNE13b]. Connections to the nodes from the clients have been done hourly (ten samples obtained per day), during the whole month of February 2015, where 300 samples are obtained for each client. Figure 3.3 shows the empirical cumulative distribution function (ECDF) of the average throughput for the two clients. On the top of the figure, the minimum/mean/maximum throughput values are shown.

Two sets of tests were conducted. One is when the writes/reads are initiated from a client that has the best connectivity in the network, such as best RTT to other nodes and the best throughput, and this is our *baseline* case; the other is when they are initiated from a client node, which is the farthest node in the network (in terms of number of hops, RTT, and throughput to other

nodes), and this is referred as a *set1* case in the graphs.

3.3.3 Experimental Results

Figures 3.4 and 3.5 show the best and worst client write/read performance. Figure 3.6 depicts the summary of all tests performed with the IOzone benchmark. Median, first and third quartile values of read and write operations are plotted in Figure 3.6. A few observations are noted below.

- In terms of network connectivity, both clients in the community network perform differently. This is related to the fact that the two clients are not connected in the same way to other nodes. The client in the baseline is better connected and is much closer in terms of RTT to the other nodes than the client in set 1.

- In terms of write performance, the baseline client performs better. Write performance for the baseline is higher and more stable than the read performance. As the file size increases, the write performance of the baseline client slightly decreases (minimum throughput achieved is 1.15 MB/s when writing a 4 MB file). The higher throughput is achieved (1.28 MB/s) when writing a small file (128 KB file), as shown in Figure 3.4.

  It is interesting to note that when writing smaller files, Tahoe-LAFS performs better, and this can be attributed to the fact that the default stripe size of Tahoe-LAFS is well optimised for writing small objects (the stripe size determines the granularity at which data is being encrypted and erasure coded). The same thing happens with the set 1 client, where the maximum write throughput achieved is 0.86 MB/s when writing a 128 KB file, and the minimum write throughput is 0.72 MB/s when writing a 2 MB file. Furthermore, write performance is affected by another factor; when writing new objects, Tahoe-LAFS generates a new public/private key, which is a computationally expensive operation.

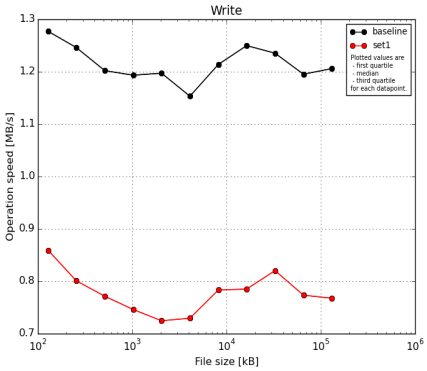


Figure 3.4: Performance of write operation

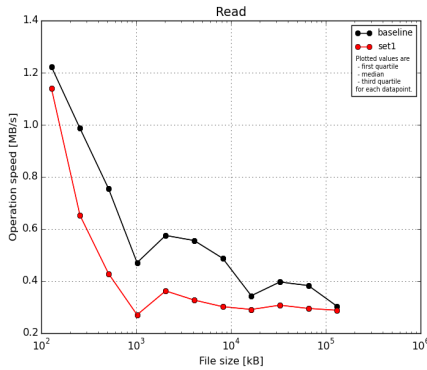


Figure 3.5: Performance of read operation

- Read operations are accomplished by submitting the request to all storage nodes simultaneously; hence, the relevant peers are found with one round-trip to every node. The second round-trip occurs after choosing the peers from which to read the file. The intention of the second round-trip is to select which peers to read from, after the initial negotiation phase, based on certain heuristics. When reading from the storage nodes, the performance of both clients drops significantly as the file size increases as shown in Figure 3.5.

  This is because when reading a file of 128 MB, a client must contact more Tahoe-LAFS storage peers in order to complete the shares of the file. In addition, reading the file system meta-object (i.e., the mutable directory objects) every time an object is accessed results in overhead, thus influencing the results.

- Figure 3.6 shows the summary of all tests performed with the IOzone benchmark. The benchmark tested file I/O performance for the 13 operations as shown in Figure 3.6. As shown, the baseline client performs better than the set 1 client, reaching an average operation speed of 0.74 MB/s for all 13 tests performed.

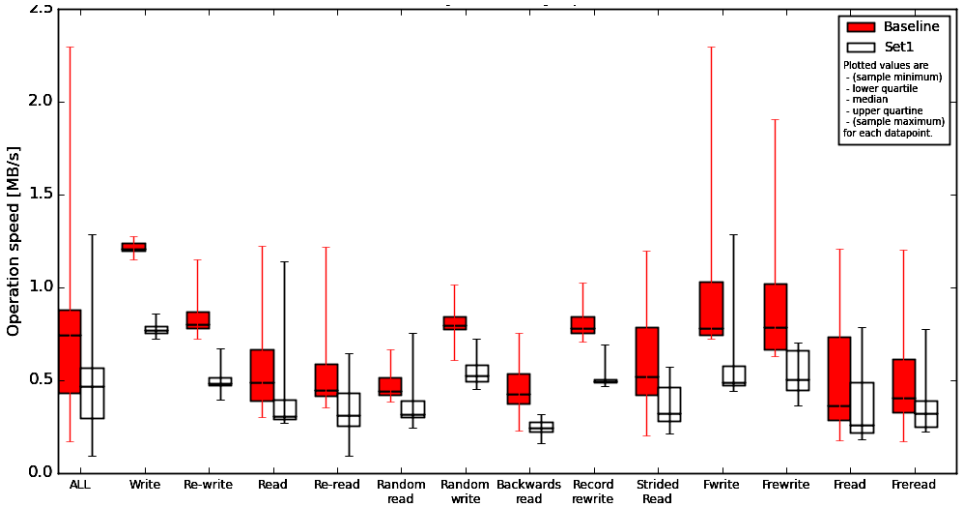


Figure 3.6: Summary of all storage benchmark operations for different tests in the `Guifi.net`

To summarise, Tahoe-LAFS is a relevant application for community networks, since it offers privacy and security, as it encrypts data already on the client side, and it offers fault-tolerance regarding storage node failures due to erasure coding (replication factors). A general important result from the experiments is that Tahoe-LAFS performed correctly in uploading and retrieving all the different file sizes under the challenging conditions of the community network, which make Tahoe-LAFS a promising application to consider for preserving privacy, and secure and fault-tolerant storage in the dynamic environment of community networks Furthermore, the process of deploying a whole storage grid is assisted by Cloudy, which allows a user easily to join or offer a storage grid.

3.4 Live-video Streaming

Among the services that are very appealing in community networks, P2P live streaming is an important candidate, as can be seen by the growing success and usage of commercial systems such as PPLive, SopCast [16q], etc. P2P

live streaming systems allow to watch live streams such as events or television channels over a network, granting anyone to become a content provider. To enable these types of services within CN nodes is very challenging, since community networks are diverse and dynamic networks with limited capacity of wireless links and often low-resource and cheap devices. Streaming applications, however, have high demands of bandwidth, they require low and stable latency and only withstand low packet loss.

We evaluate the performance of PeerStreamer as a P2P live streaming service deployed over geographically distributed real community network nodes. We then study the effects of different parameters of PeerStreamer on its performance in the community network environment.

3.4.1 PeerStreamer

PeerStreamer [16k] is an open source live P2P video streaming service, and mainly used in our Cloudy distribution as the live streaming service example. This service is built on a chunk-based stream diffusion, where peers offer a selection of the chunks that they own to some peers in their neighbourhood. The receiving peer acknowledges the chunks it is interested in, thus minimizing multiple transmissions of the same chunk to the same peer. Chunks consists of parts of the video to be streamed (by default, this is one frame of the video). At the beginning of the streaming process, these chunks are all from the same peer (since only one peer is the source), then the source sends m copies of the chunks to random peers ($m = 3$ by default), creating an overlay topology with all peers in order to exchange chunks between them. The whole architecture and vision of PeerStreamer is described in detail in [B+11].

3.4.2 PeerStreamer Assumptions and Notation

We call the community network the *underlay* to distinguish it from the *overlay* network which is built by PeerStreamer. The underlay network is supposed

to be connected and we assume each node knows whether other nodes can be reached (next hop is known). We can model the underlay graph as:

$$G^{ug} = (S, L^{ug}) \tag{3.1}$$

where S is the set of super nodes present in community network and L^{ug} is the set of wireless links that connect them. This is the global level.

In the micro-cloud level we have a set of outdoor routers (OR) that are connected to each other in the same micro-cloud as shown in Figure **??**,

$$G^{um} = (OR, L^{um}) \tag{3.2}$$

where OR is the set of outdoor routers present in the micro-clouds of the CNs and L^{um} is the set of wireless links that connects them.

The nodes of the underlay (connected to super nodes through outdoor routers) run an instance of the PeerStreamer and are called *peers*. Each peer P_i at time t chooses a subset of the other peers as a set of neighbours that are called $N_i(t)$. The peer P_i exchange video frames (chunks) only with peers in $N_i(t)$, and the union of all the $N_i(t)$ and the related links defines the network topology of the application, also represented as graph and called *overlay*. The overlay built by PeerStreamer is a directed graph:

$$G^{og}(t) = (P_{set}, L^{og}(t)) \tag{3.3}$$

where P_{set} is the set of peers and

$$L^{og}(t) = (P_i, P_j) : P_j \in N_i(t) \tag{3.4}$$

is the set of edges that connect a peer to its neighbours. The main difference between the overlay and the underlay is that the underlay is determined by the network topology, on which PeerStreamer does not have control, while the overlay is generated by PeerStreamer.

3.4.3 Experiment Setup

For the experimental research, our main configuration includes geographically distributed CN nodes from Guifi.net in Spain, AWMN in Greece and Ninux in Italy. These nodes are co-located in either users homes (as home gateways, set-top-boxes etc) or within other infrastructures around each city. Two CNs (Guifi.net and AWMN) are connected on the IP layer via the FEDERICA infrastructure, enabling network federation. The nodes of Ninux CN in Italy are not connected to FEDERICA, therefore we experiment with them separately (without including other CN nodes). In our experiments the nodes from UPC (Technical University of Catalonia) are a subset of Guifi.net CN nodes which are distributed in our UPC campus in Barcelona. We use these nodes as a baseline in order to be able to better understand the effects of the network given by the statistical data gathered from the community networks.

Our experimental evaluation is comprised of 55 physical nodes distributed across Europe, among the working nodes available from the three CNs. Table 3.3 shows the number of nodes used in three community networks, their location and type of devices deployed.

In our experiments we connect a live streaming camera (maximum 512 kbps bitrate, 30 fps) to a local PeerStreamer instance which acts as the *source* for the P2P streaming. We choose as a source a stable node with good connectivity and bandwidth to the camera in order to minimize the video frame loss from the networked camera. The source is responsible for converting the video stream into chunk data that is sent to the peers. In the default configurations of PeerStreamer a single chunk is comprised of one frame of the streaming video. Also, the source PeerStreamer node sends three copies ($m = 3$) of the same chunk to the peers, meaning that only three peers receive the chunks directly from the source at a given time. Thus, each peer that receives the chunks exchange with other peers in order to form the P2P exchange network.

The evaluation metrics presented were chosen in order to understand the network behavior, quality of service and quality of experience. On the qual-

Nr. of nodes	Cat.	Location	Type
23	UPC	Barcelona, Spain	Physical nodes and VMs
8	Guifi.net	Catalonia, Spain	Physical nodes
12	AWMN	Athens, Greece	Physical nodes
12	Ninux	Rome, Italy	Physical nodes

Table 3.3: Nodes in the cluster and their location

ity of service side, we measure the number of chunks that are received by peers and the chunk loss percentage in order to understand the impact of the network on the reliable operation of this type of service. On the quality of experience, we gather statistical data from the chunks that are played out locally by each of the peers to understand the quality of the images that the edges show to the users. These metrics, show the impact of such networks when using streaming services while also guaranteeing the image quality that each node can display on average. Regarding the network interference issues of other users' concurrent activity which can impact the results of the experiments, we reference to [CNE13a] and is out of the scope of this work.

3.4.4 Scenarios

To assess the applicability of PeerStreamer in CNs, the following describes a chosen scenario that reflects a use case of live video streaming in CNs. Also, we augment our findings with a scenario reflecting different parameters of PeerStreamer usage, in order to understand possible improvements of the overlay network created by the PeerStreamer instances. The parameters used in the scenarios are summarized in Table 3.4.

For the first scenario we choose the default parameters of PeerStreamer and run in the challenging environment of CNs. One of the nodes, which has the best connectivity to the camera stream is chosen to be the source peer, while the rest of the available nodes will initially contact the source in order to enter the P2P network for chunk exchange. Since the Ninux group of nodes do not

Scenario	1 and 2
Total number of nodes	55
Groups of nodes	UPC, Guifi.net, AWMN, Ninux
Tests time-frame	T1 = 30m | T2 = 1h | T3 = 2h
Source 1 Send Rate (chps)	T1 = 31 | T2 = 32 | T3 = 31
Source 2 Send Rate (chps)	T1 = 55 | T2 = 55 | T3 = 49
Metrics	Peer Receive Ratio, Chunk Loss Chunk Playout, Neighborhood Size

Table 3.4: Summary of our Scenario Parameters

have connectivity in IPv4 to other CNs (they are not part of FEDERICA), we deliberately executed the experiment apart from the other CNs, in order to understand different CNs network behaviors. The experiment ran on this group was different because of the non-connectivity to the camera stream, therefore another solution was devised. We introduced a live TV streaming channel as the streaming source, transcoded to 512 kbps bitrate, 30 fps on average similar to the camera stream. However, this stream also included audio, which made the exchange of data between peers higher than the peers of other CNs. Each experiment is composed of 20 runs, where each run has 10 repetitions, and averaged over all the successful runs (90% of the runs were successful). In the 10% of the runs the source was not able to get the stream from the camera, so peers did not receive the data. The measurements we present consists of 3 weeks of experiments, with roughly 300 hours of actual live video distribution and several MBytes of logged data.

We then establish three experiments shown for 30 minutes, 1 hour and 2 hours of continuous live streaming from the PeerStreamer source. This was done in order to gather statistical information within different time-frames and to use as initial step towards live events coverage on CNs. Other nodes were started at the same moment in time, 10 seconds after the source started, in order for the source to gather enough data to be able to exchange with the

peers. This also allows the randomization of the nodes that the source PeerStreamer will first push the chunks to, and thus on all experiments the peers that begin receiving chunks from the source will be different (PeerStreamer overlay topology changes in every run of experiments). In all experiments we try to guarantee the number of nodes to remain constant. However, since we are dealing with a very dynamic and challenging environment, there is an issue of churn rate of nodes. This happens in the CNs because most of the nodes are connected wirelessly and their connectivity depends on many factors (such as weather, electric failures, router connectivity, among others). PeerStreamer for its own overlay performs operations to manage the peer churn rate by constantly updating each peer neighborhood, an important feature for the potentially unstable and dynamic nodes that we find in community networks.

For our second scenario, the evaluation performed includes the findings of different configuration parameters of PeerStreamer, which results in better quality streaming. This was done in order to understand the different behaviors of the PeerStreamer algorithms such that the overlay network that it constructs can be optimized. The different parameters chosen include sending different amount of copies of the chunks from the PeerStreamer source ($m = 5$, $m = 1$); keeping the best peers in the neighborhood in between topology updates of the overlay that PeerStreamer creates ($TopoKeepBest$); and the addition of the peers that can be selected to the neighborhood by extending the default RTT (10 ms) of the peer selection metric [B+11] to 20 ms.

3.4.5 Experimental Results

Figure 3.7 depicts the amount of chunks on average the peers receive. Knowing that Source 1, sends out to the peers around 31 chunks per second (chps), we notice that the distant groups (Guifi.net and AWMN) in relation to the source, receive less chunks than the closer group (UPC), in relation to the

source. This is because of the network impact on the delivery time of the chunks. Thus, more chunks arrive out of the time allotted, the farther the chunks have to travel. We also notice that the number of chunks received on average increases with longer time-frames, this occurs because the peers can gather more statistical information about each other and therefore update their neighboring peers accordingly, while securing a subset of peers in which they can rely on to receive the chunks in the time allotted to be displayed.

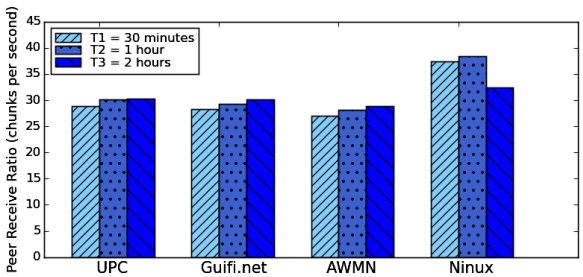


Figure 3.7: Average Peer Receive Ratio

We also show that on Ninux side the amount of chunks received tends to be higher that of the other CNs. This is due to the fact that we use a different stream (Live TV channel stream), in which Source 2 sends around 55 chps instead (accounting with the added audio part of the stream). We also notice a drop of receiving chunks for longer times, because of the inherited instability of this group of nodes, where the loss of data is more constant/visible when dealing with longer times.

Figure 3.8 shows the average chunk loss for each group of peers. We can see that the loss is greater for shorter time-frames (loss in UPC 7%, Guifi.net 9% and AWMN 13%) and are amortized for longer time-frames (loss in UPC 2%, Guifi.net 3% and AWMN 7%). We also notice that distant groups (distant from the source stream) are more affected by the diminished rate of chunks received, which demonstrates the influence the network has to the amount of data that is lost (either by losses on the network or by not arriving on time to

be displayed). As for the Ninux group, as previously mentioned, the network behavior is more volatile since there is a higher packet loss. Therefore, we notice that since Source 2 sends more chunks per second (around 55) than Source 1, the loss of chunks in the peers is greater than in other groups and in longer time-frames the network instability has a higher impact on the data exchanged (34% loss).

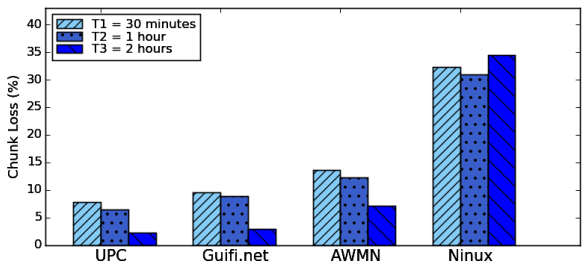


Figure 3.8: Average Chunk Loss

Figure 3.9 illustrates the quality (chunks played) of video offered on the peers side. The closer groups display more chunks, because the loss between farther nodes is greater than closer nodes and since the network plays a big role on the delivery of chunks. We also notice that the longer time-frames have on average a better chunk playout because more chunks arrive on time to be displayed (UPC 98%, Guifi.net 98%, AWMN 92%). For the Ninux group we see a more stable chunk playout for each of the time-frames, which means that since the network instability occurs during the whole evaluation the same amount of chunks (on average 71%) arrive to be displayed, also meaning that the network bandwidth/throughput between nodes (on average) is lower than on other CNs and remains constant over time.

Figure 3.10 demonstrates the chunk loss gathered during 30 minutes experiment, with different parameters given to the PeerStreamer. The parameters shown ($TopoKeepBest$, $RTT = 20ms$ and $m = 5$) have been selected in order to predict the behavior and improvements that PeerStreamer can have when executed in CNs. We notice that increasing the RTT for the overlay topo-

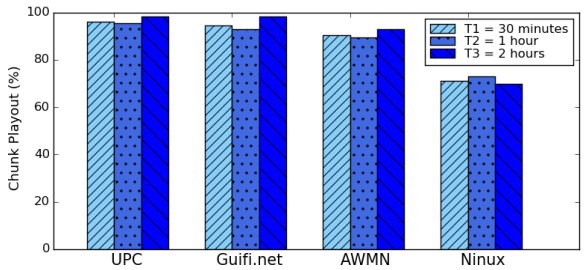


Figure 3.9: Average Chunk Playout

logy gives the peers higher probability to receive chunks in time and therefore decreasing the chunk loss in each of the groups. The other parameters have a higher impact on losing chunks, especially when the source only sends one copy ($m = 1$) of the chunks to peers (not shown in the figure). We also notice that keeping the best neighbors on topology overlay updates, lowers groups loss chunks (as in UPC case) that have nodes closer to each other, in which the selection of peers for exchanging chunks will have higher probability to choose the best nodes from previous topology updates. For the Ninux group we notice that when keeping the best nodes on topology updates there is a greater improvement (23% in loss, comparing with default parameters where we got 32% loss), because the probability of choosing the best nodes will be higher, since the nodes on this CN have worst connectivity. Also for Ninux, giving a RTT of 20 ms has mostly the same average as the previous experiments (with default parameters) since the nodes are farther apart (in RTT terms), meaning that there will be no significant changes in the neighborhood created for these peers. We also show that there is improvement when changing the number of chunk copies Source 2 sends to peers ($m = 5$). This is because of the resources that Source 2 has at its disposal, which makes it able to send more copies without losing bandwidth and computation time (against Source 1 as a low-power device); and also, since the network has more packet loss than in other CNs, flooding the network with more copies makes a higher probability for peers to be able to receive more chunks on time.

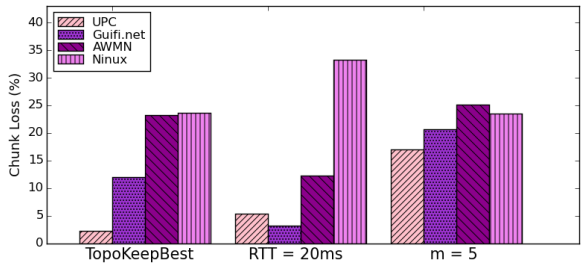


Figure 3.10: Average Chunk Loss with different parameters

3.5 Service Discovery

Cloud service discovery is essential for allowing cloud usage and user participation. Service discovery involves service providers publishing services and clients being able to search and locate service instances. Since community cloud nodes are distributed all over the network and administrated by their owners, a mechanism is needed that allows the cloud users to discover services offered by other community cloud nodes and announce their own services. We experiment with the Avahi-TincVPN and Serf search service of Cloudy that offers service announcement and discovery to community users.

3.5.1 Experiment setup

For our service discovery experiment we use 25 nodes spread between two community networks (Table 3.5). We use 20 nodes from the Guifi.net community network, where 13 of the nodes are located in the city of Barcelona (UPC) and seven of them are located in the Catalonia region of Spain. From AWMN we use five nodes, which are located in Athens, Greece.

Figure 3.11 shows the throughput of three categories of nodes in our cluster. Network measurements have been obtained connecting by SSH to each node and measuring the average aggregated throughput from the client nodes. For some scenarios we use more than one client node. The clients are located in the Guifi.net community network. Connections to the nodes have been

Table 3.5: Nodes, their location and RTT from the client node

Number of nodes	Community Network	Location	RTT
13	Guifi.net (UPC)	Barcelona	1–7 ms
7	Guifi.net	Catalonia	10–20 ms
5	AWMN	Athens	90–100 ms

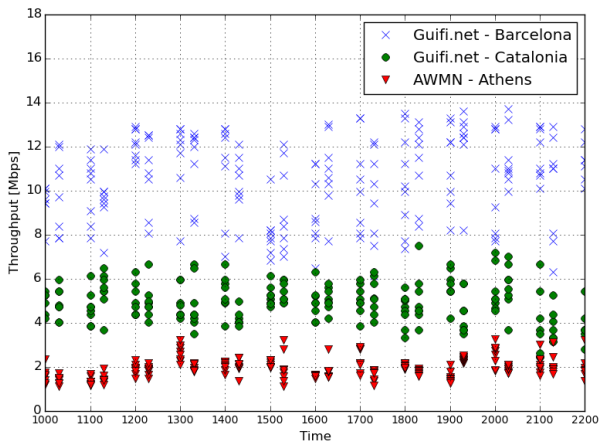


Figure 3.11: Throughput of the nodes

done every half hour (12 hours per day), during the entire month of January 2015 where 720 samples are obtained for each category of nodes. All obtained samples are plotted in the graph. The average throughput obtained for the Guifi.net nodes in UPC is 10.5 Mbps, Guifi.net nodes is 4.8 Mbps, and AWMN nodes is 1.9 Mbps.

The objective of the experiments is to understand the responsiveness of the discovery mechanism. We consider responsiveness to be the probability of successful operation within deadlines, which, when applied to our case, refers to successful service discovery within the given time limits. Furthermore, we attempt to understand how the clients perceived responsiveness changes when they are located in different parts (zones) of the Guifi.net community network.

We run the discovery requests from three client nodes that are searching and locating service instances. All other nodes acted as service providers responding to discovery requests. All service providers are spread between two community networks. Discovery times are measured on the clients directly before the request was sent and directly after responses were received to measure user-perceived responsiveness. No nodes joined or left the network; therefore, no configuration on the network layers occurred during measurements which would interfere with the discovery operation. We consider the discovery successful when all instances have been discovered. Discoveries were aborted and considered failed if no responses arrived until an experiment ran with a deadline of 25 seconds in the Community-Lab testbed. This value was chosen because in Zeroconf [15l], the time between retries doubles after each retry to reduce the network load. Therefore, for example, after 20 seconds, we have reached five discovery requests, and the next one would be sent after 41 seconds. Depending on the scenarios that we consider, each service discovery experiment is comprised of several runs (normally between 15 to 20) and is averaged over all the successful runs. Each run consists of 20 repetitions.

In Avahi, when publishing and discovering, no entries are cached per interface; thus, no caching is used. After service discovery, a client should have enough information to contact a service instance. Hence, discovery in our case means resolving the IP address and port for every service instance. During the experiments we use different Cloudy services to publish and discover as summarised in Table 3.6.

3.5.2 Our Scenarios

In order to judge the applicability of decentralised discovery mechanisms in community networks, three scenarios are chosen that reflect common use cases of service discovery.

Scenario 1: Single service discovery. Our first goal is to measure the responsiveness of single service discovery. In this scenario, the service network

Table 3.6: Services used for the experiments

Service	Description
PeerStreamer	Live-video streaming service
Tahoe-LAFS	Decentralised cloud storage service
Syncthing	File synchronisation service
Serf	Cluster membership and discovery service
OWP	Container-based virtualisation service
Proxy3	Guifi.net proxy service
SNP Service	Guifi.net network graph service
DNS Service	Guifi.net DNS service

consists of one client and one provider. The client is allowed to wait up to ten seconds for a positive response. This is a common scenario for service discovery and can be considered the baseline. Only one answer needs to be received and there is enough time to wait for it. In this case, the client discovers a Tahoe-LAFS distributed storage service and contacts the service. For this scenario, a service provider from the Guifi.net community network is considered. Both Avahi-TincVPN and Serf are used for this scenario.

Scenario 2: Timely service discovery of the same service type. Service networks are populated with multiple instances of the same service type. The clients need to discover as many instances as possible and will then choose one that optimally fits their requirements. The faster discovery is better. In this scenario, we have one service client and 25 service providers (from Guifi.net and the AWMN community network). The discovery is successful if all provided service instances of the same type have been discovered. We measure how responsiveness increases with time. The faster we reach a high value, the better. In this scenario, the providers publish one or more PeerStreamer live-video streaming services. The client waits 15 seconds to receive responses. In this scenario only the service discovery based on Serf is used. The total number of services published by providers is 40.

Scenario 3: Client perceived responsiveness. In this scenario, we have three service clients and 20 service providers that publish five popular services of Cloudy such as PeerStreamer, Tahoe-LAFS, Syncthing, DNSService, and OpenVZ. The total number of the five Cloudy services published is 23 (seven PeerStreamer services, three Tahoe-LAFS services, six Syncthing services, three DNSServices and four OpenVZ services). The clients are located in different parts of Guifi.net, and they need to discover 23 instances of different service types. Considering the dynamic environment of the Guifi.net community network, the discovery is successful if all clients discover the same number of services (23) published by service providers. For this scenario, the service discovery based on Serf is used. The clients are allowed to wait 20 seconds.

3.5.3 Experimental Results

In this section, the results for the three scenarios described above are presented.

Scenario 1: Single service discovery. The discovery of a single service instance within ten seconds proved to be reasonably responsive. This experiment is comprised of 15 runs, where each run has 20 repetitions. In Figure 3.12, the standard deviation error bars per round are plotted on the mean values obtained. The values are obtained using Avahi-TincVPN and Serf. Due to the efficient and lightweight gossip protocol that Serf uses, it decreases the discovery time for 3x, reaching an average of two seconds for a single service discovery compared to the Avahi-TincVPN combination that reaches six seconds.

Scenario 2: Timely service discovery of the same service type. Figure 3.13 illustrates that the discovery of services increases rapidly with time. The standard deviation error bars are plotted on the mean values. In the first six seconds the client discovers 75% of the published services, which is equal to 30 PeerStreamer video-streaming services. The last 25% of the services are

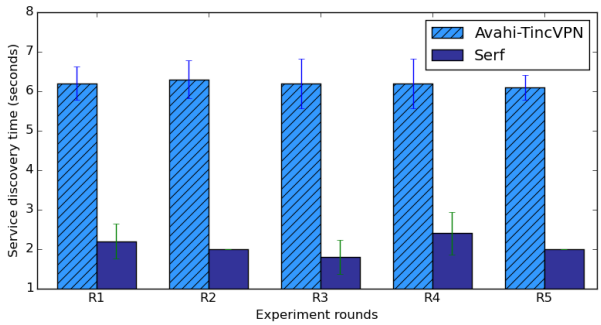


Figure 3.12: Single service discovery time (Scenario 1)

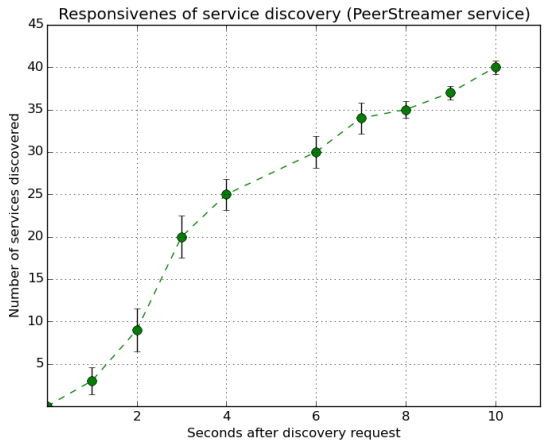


Figure 3.13: Responsiveness of service discovery (Scenario 2)

discovered from seconds six to ten. These ten services are from the AWMN community network. The eventually consistent gossip model of Serf with no centralised servers allows the client to discover in a very fast and extremely efficient way all the nodes for a PeerStreamer service based on the tags their agent is running. However, the structure and diameter of the community network graph (topology), fluctuations in the network due to load, and faults can increase the discovery time [Veg+12].

Figure 3.14 shows a partial screenshot of the Cloudy web interface, de-

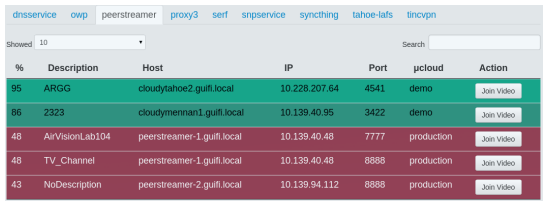


Figure 3.14: Partial screenshot of Cloudy's Service discovery (Scenario 2)

picting the service discovery section. The five PeerStreamer video-streaming services discovered are shown. The user will choose one that optimally fits the user's requirements. The services are ranked according to Cloudy's QoS-aware service selection algorithm, where colour represents service quality: darker colour indicates poorer service quality.

Scenario 3: Client perceived responsiveness. Figure 3.15 demonstrates the number of services discovered by three clients. The standard deviation error bars are plotted on the mean values. As shown, the three clients perceive a different number of services. Only Client 2 discovers all services (23). Client 1 is missing one PeerStreamer service and one DNSService. Client 3 is missing just one PeerStreamer service. Missing services can be subject to the high diversity of the quality of wireless links, the availability of nodes, and the location of client nodes. Heterogeneous low-resource hardware, slow wireless links, and packet loss between nodes also can impact the service performance [CNE13b].

3.6 Discussion

Distributed Storage: We evaluated how the Tahoe-LAFS storage system performs when it is deployed over distributed community *micro-cloud* nodes in a real community network. We observed that the write operation of Tahoe-LAFS resulted in better performance than the read operation. Read operation is a more expensive operation which is done in two round-trips. In the first round-trip the relevant peers holding the shares are found, and second

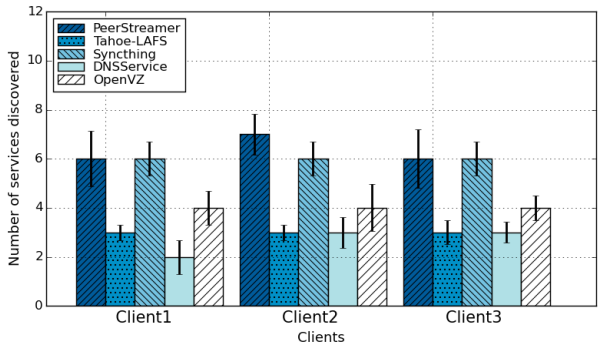


Figure 3.15: Number of Cloudy services discovered by clients (Scenario 3)

round trip occurs after choosing the peers from which to read. Specifically for the community network environment such as *Guifi.net*, successfully running the distributed storage as Tahoe-LAFS is a trade-off between the replication settings and read performance. Keeping the default Tahoe-LAFS replication setting in 3-of-10 (N=10, K=3) and workload limit to 128 MB works satisfactorily when having up to 30 nodes in the network. In order to have a better fault-tolerance and lower response times while using bigger workloads, taking network characteristics into account when deciding placement of storage nodes is a must. Chapter 4 and Chapter 5 addresses this issue.

Live-video Streaming: We started our evaluation by demonstrating the performance PeerStreamer has on community networks, with the default parameters, in order to understand what improvements can be achieved. We found that PeerStreamer neighborhood selection lacks accountability for network instability and therefore PeerStreamer can perform poorly in community networks. The metric for randomly selecting a subset of peers for the neighborhood reduces the probability to receive chunks in time, since peers can select the worst neighbors. We also found that while modifying the number of chunk copies that the source sends, can have beneficial results, guaranteeing that the chunks will travel to more nodes and be available to be traded in the P2P network over more peers. However, since the wireless links in com-

munity network are with high diversity in bandwidth, this issue can arise and should be studied more thoroughly. Regarding the amount of data exchanged between peers, we consider that in current wireless community networks using the high quality video streams (i.e., 1080p) affects the service performance since more data or sizable data needs to pass through the network to the peers, and may even congest it. While using standard quality video streams, as shown in our evaluation the amount of loss is lower and more efficiently exchanged between peers in the network.

Service Discovery: Service discovery operates in parallel at both the global community network cloud level and at the *micro-cloud* level. At the *micro-cloud* level, a number of Cloudy instances are federated and share a common, private Layer 2 over Layer 3 network. At that level, Avahi was used for announcement and discovery. Originally this solution was to be applied to the whole community network but as more Cloudy instances started to appear it became clear that the solution would not scale further than the tens of nodes as we explain in [Sel+15a]. However, in the context of an orchestrated micro-cloud, it can be used not only for publishing cloud services but also other resources like network folder shares, etc. Because of the scaling issues Serf, a decentralized solution for cluster membership, failure detection, and orchestration was used, This protocol is, in practice, a very fast and extremely efficient way to share small pieces of information. An additional byproduct is the possibility of evaluating the quality of the point-to-point connection between different Cloudy instances.

3.7 Summary

Community networks would greatly benefit from the additional value of applications and services deployed inside the network through community *micro-clouds*. However, such clouds in community networks have not yet been demonstrated in related studies as operational systems, missing proof of feasibility, which would enable exploring further innovations.

In this chapter, a deployed community *micro-cloud* operating in the `Guifi.net` community network was demonstrated, supporting the feasibility of such a system. In addition, performance measurements were conducted, which demonstrate the usability of cloud-based services for end users. The *micro-cloud* environment was materialised by the implementation of the Cloudy distribution. Using Cloudy and the hardware infrastructure available in the community network, the community *micro-cloud* was deployed in Guifi.net, demonstrating the community cloud feasibility.

We identified three services to focus on, considering their importance for the system operation and the end users. On the user-level we evaluated distributed storage service Tahoe-LAFS and live-video streaming service PeerStreamer. Tahoe-LAFS performed correctly in uploading and retrieving all the files under the challenging conditions of the community network and appeared to be a promising application to consider for preserving privacy, and for secure and fault-tolerant storage in community clouds. PeerStreamer, a P2P live streaming service was deployed over geographically distributed real community network nodes. We then studied the effects of different parameters of PeerStreamer on its performance in the community network environment. We experimented with service discovery based on Avahi and Serf, and the results exhibited their proper functioning. These system-level services allowed users to discover services offered by other community cloud nodes as well as announce their own services.

Performance measurement of services and applications provided by the *micro-clouds* were conducted in order to assess their usability by end users. Our results demonstrated the operation of the community network *micro-cloud* in the community network and the usability of services.

Notes

The research discussed in this Chapter 3 was included in the following publications:

[Sel+16] M. Selimi, N. Apolónia, F. Olid, F. Freitag, L. Navarro, A. Moll, R. Pueyo, and L. Veiga. "Integration of an Assisted P2P Live Streaming Service in Community Network Clouds". In: *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom 2015)*. Nov. 2016, pp. 202–209.

[Sel+15a] Mennan Selimi, Felix Freitag, Llorenç Cerdà-Alabern, and Luís Veiga. "Performance evaluation of a distributed storage service in community network clouds". In: *Concurrency and Computation: Practice and Experience* 28.11 (2015). (JCR IF: 0.942, Q3), pp. 3131–3148.

[Sel+15b] Mennan Selimi, Amin M Khan, Emmanouil Dimogerontakis, Felix Freitag, and Roger Pueyo Centelles. "Cloud services in the Guifi.net community network". In: *Computer Networks* 93.P2 (Dec. 2015). (JCR IF: 1.446, Q2), pp. 373–388.

4

Topology-aware Service Placement

Cloud services in community networks have been enabled by micro-cloud providers. They form community network micro-clouds (CNMCs), which grow organically, i.e. without being planned and optimized beforehand. Services running in community networks face specific challenges intrinsic to these infrastructures, such as the limited capacity of nodes and links, their dynamics and geographic distribution. CNMCs are used to deploy distributed applications, such as streaming and storage services, which transfer significant amounts of data between the nodes on which they run. Currently there is no support given to users for enabling them to chose better or the best option for specific service deployments. This chapter looks at the next step in community network cloud service deployments, by taking network characteristics into account when deciding placement of service instances. The main contributions of this chapter can be summarized as follows:

- We introduce a service placement algorithm that provides optimal service overlay allocations without the need to verify the whole solution space. The algorithm *PASP* finds the minimum possible distance in

terms of the number of hops between two furthest selected resources, and at the same time fulfil different service type quality criteria (i.e., research question **Q2**).

- We extensively study the effectiveness of our approach in simulations using real-world node and usage traces from 30,000 Guifi.net nodes. From the results obtained in the simulation study, we are able to determine the key features of the network and node selection for different service types.

- Subsequently, we deploy our algorithm, driven by these findings, in a real production community network and quantify the performance and effects of our algorithm with a distributed storage service.

4.1 System Model

4.1.1 Network structure

The Guifi.net community network consists of a set of *nodes* interconnected through mostly wireless equipment that users, companies, administrations must install and maintain in addition to its links, typically on building rooftops. The set of nodes and links are organized under a set of mutually exclusive and abstract structures called administrative *zones*, which represent the geographic areas where nodes are deployed. Figure 4.1 shows as example the nodes and links of Guifi.net in the city of Barcelona. Figure 4.2 shows the topology structure followed in Guifi.net. Client nodes are connected to the super-nodes. These super-nodes interconnect through wireless links different administrative zones.

We have collected network description data through CNML files [16f] (obtained January 2016). CNML (Community Networks Markup Language) is an XML-based language used to describe community networks. Guifi.net publishes a snapshot of its network structure every 30 min with a description

Figure 4.1: `Guifi.net` nodes and links in Barcelona

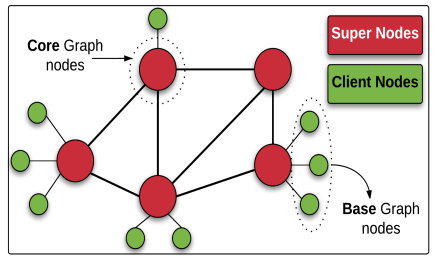


Figure 4.2: `Guifi.net` topology

of registered nodes, links and their configurations. In the CNML description, the information is arranged according to different geographical zones in which the network is organised. Furthermore, we used a *Node database*: a dump of the community network database that, in addition to the data described in CNML, includes other details about dates and people involved in the creation and update of the configuration of nodes and links.

The CNML information obtained has been used to build two topology graphs: *base-graph* and *core-graph*. The base-graph of Guifi.net is constructed by considering only operational nodes, marked in *Working* status in the CNML file, and having one or more links pointing to another node in the zone. Additionally, we have discarded some disconnected clusters. All links are bidirectional, thus, we use an undirected graph. We have formed what we call the core-graph by removing the terminal nodes of the base graph (i.e., client nodes). Table 4.1 summarizes the main properties of base and core graphs that we use in our study e.g., number of nodes, node degree, diameter (number of max hops in the sub-graph) and number of zones traversed in core and base-graph.

4.1.2 Allocation model and architecture

In order to generalize the placement model for community services, we made the following assumptions that give to our model the flexibility to adapt to

	nodes/edges	node degree max/mean/min	diameter	zones
BaseGraph	**13636/13940**	537/2.04/1	35	129
CoreGraph	**687/991**	20/2.88/1	32	85

Table 4.1: Summary of the used network graphs

many different types of real services. In our case, a *service* is a set of S generic processes or replicas (with different roles or not) that interact or exchange information through the community network. The service can also be a composite service (e.g., three-tier service) built from simpler parts. These parts or components of a service would create an overlay and interact with each other to offer more complex services. Each of the service replicas or components will be deployed over a node in the network, where each node will host only one process no matter which service it belongs to.

It is important to remark that the services aimed in this work should be at infrastructure level (IaaS), as cloud services in current dedicated datacenters. Therefore the services are deployed directly over the core resources of the network (nodes in the core-graph) and accessed by base-graph clients. Services can be deployed by Guifi.net users or administrators. The architecture that we consider is based on a hybrid peer-to-peer model with three hierarchical levels of responsibility. On each level, members are able to share information among themselves.

The coordination is managed by some peer (i.e., as a super-peer) designated from the immediate upper layer. Three types of peers can be identified:

1. **Community Nodes**: are the computing equipment placed along the wireless community network by users. Besides contributing to the network quality and stability, they share all or part of their physical re-

sources with other community members in an infrastructure as a service (IaaS) fashion. In terms of type and amount of resources, our model assumes the nodes are different. This means that from service point of view there is allocation preference.

2. **Zone Managers**: are single nodes - only one within each zone, selected among all the Community nodes with the extra responsibility to manage local zone services and coordinate inter zones aggregated information. In our model we do not explicitly identify these managers and we assume the existence of at least one of them in each area.

3. **The Controller**: is a unique centralized entity in our system. The role of the controller is to manage all the service allocation requests from the users and update service structures by pulling the configuration information for the zone managers. The allocation algorithms are implemented in the controller.

4.1.3 Service quality parameters

Resource dispersion in a community network scenario can be a drawback or an advantage, as the Nebula [Ryd+14] authors claim in their research. The overlay created by composite services abstracts from actual underlying network connections. Based on that, services that require intensive inter-component communication (e.g streaming service), can perform better if the replicas (service components) are placed close to each other in high capacity links [Sel+15b]. On other side, bandwidth-intensive services (e.g., distributed storage, video on-demand) can perform much better if their replicas are as close as possible to their final users (e.g. overall reduction of bandwidth for service provisioning).

If we have some information about the application SLAs in community networks and node behaviour from the underlying network, decisions can be

made accordingly, in order to promote that certain types of applications are executed in certain type of nodes with better QoS.

Our algorithm considers the following network and graph metrics as shown in Table 4.2, when allocating different type of services.

- **Availability:** The availability of a node is defined as the percentage of ping requests that the node replies when requested by the graph-server system. Graph-servers are distributed in Guifi.net and are responsible for performing network measurements between nodes. This is an important metric for service life-cycle and is considered for two service types. It is measured in percentage (%).

- **Latency:** The latency of a node is defined as the time it takes for a small IP packet to travel from the Guifi.net graph-servers through the network to the nodes and back. It is an important metric for latency-sensitive service in CNMCs. It is measured in milliseconds (ms).

- **Closeness:** The closeness is defined as the average distance (number of hops) from the solution obtained from the algorithm to the clients. It is an important metric for bandwidth-sensitive services. It is measured in number of hops.

In terms of graph centrality metrics, we consider closeness and betweenness centrality. Closeness centrality is a good measure of how efficient a particular node is in propagating information through the network. Betweenness centrality quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.

4.2 Service Placement Algorithm

We designed an algorithm that explores different placements searching for the local minimal service overlay diameter while, at the same time, fulfilling different service type quality parameters. Algorithm 4.1 relies on the method

Type of service	*Examples of services*	*Network metrics*	*Graph metrics*
Bandwidth-intensive	distributed storage, video-on-demand	availability	**closeness**
	network graph server, mail server	**closeness**	**centrality**
Latency-sensitive	VoIP, video-streaming	availability	**betweenness**
	game server, radio station server	**latency**	**centrality**

Table 4.2: Service-specific quality parameters

$PASP()$ to evaluate the different service placements in different zones and generate the solutions. The algorithm tries to find a solution in each zone by applying Breadth-First Search (BFS) and utilizing the $IsBetter$ method to choose the best solutions by applying service policies shown at Table 4.2. In the case of equal diameter allocations, the mean out-degree (the mean boundary of the nodes in the service overlay with the nodes outside of it) is taken. The service allocation with smallest diameter and largest mean out-degree fulfilling different service quality parameters is kept as the optimal.

The algorithm iterates using Breadth-First-Search algorithm (BFS) in the network graph, taking as root the given node and selecting the first $S-1$ closest resources to it. The node with high degree centrality is initially chosen as root. Degree centrality is the fraction of nodes that a particular node is connected to. In the case of several nodes at the same distance, nodes are selected randomly, distributing thus uniformly. Thanks to this feature, our algorithm performs faster than a pure exhaustive search procedure, since size equivalent placements are not evaluated. It is worth noting that the same set of nodes might be obtained from different root nodes, since placements in nearby network areas would involve the exact same nodes. We avoid re-evaluating these placements with a cache mechanism, that improves algorithm efficiency. After the placement solutions for different number of services are returned from BFS, the solutions are compared regarding the service quality parameters.

For each solution set obtained, we check our defined service-specific policies

Algorithm 4.1 Policy-Aware Service Placement (PASP)

Input: $N(V_n, E_n)$ ▷ Network graph
Input: $Z(V_z, E_z)$ ▷ Zones graph
Input: $Zone$ ▷ Search solution zone
Input: S ▷ Number of nodes in the service
Input: $ServicePolicy$ ▷ Service specific policies

```
1: procedure PASP(N, Z, Zone, S, ServicePolicy)
2:     Community ← V_n ∈ V_{z_i}
3:     BestSolution ← null
4:     for all node ∈ Community do
5:         solution ←BreadthFirstSearch(N, Community, node, S)
6:         if isBetter(solution, BestSolution, ServicePolicy) then
7:             BestSolution ← solution
8:         end if
9:     end for
10:    return BestSolution
11: end procedure
12: procedure isBetter(currentSolution, bestSolution, ServicePolicy)
13:    for all p ∈ ServicePolicy do
14:        result ←CheckPolicy(currentSolution, bestSolution, p)
15:    end for
16:    return result
17: end procedure
```

and then accordingly we calculate different scores (e.g., latency or availability score). Once we have the these scores for each solution set, we utilize the $IsBetter$ method to compare the solutions and to choose the new best placement solution according to different service types. Currently, the algorithm has not been optimized regarding the computation time, but it provides near-optimal overlay allocations, as our results show, without need of verifying the whole solution space.

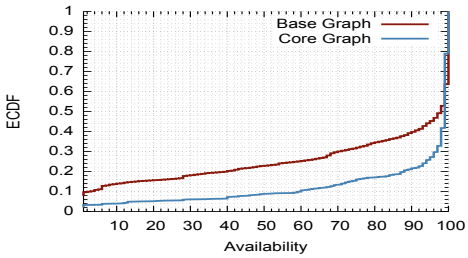


Figure 4.3: ECDF of node availability

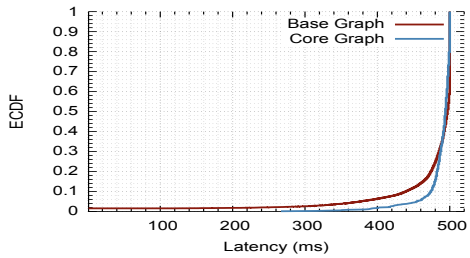


Figure 4.4: ECDF of node latency

4.3 Experimental Results

4.3.1 Network behaviour and algorithmic performance

Our service placement algorithm proposed in Section 5.2.4 is used to simulate the placement of different services in Guifi.net. Our goal is to determine the key features of the network and its nodes, in particular to understand the network metrics that could help us to design new heuristic frameworks for smart service placement in CNMCs.

From the data obtained, our first interest is to analyse the availability and latency of Guifi.net nodes. This can be used as an indirect metric of quality of connectivity that new members may expect from the network.

Figure 4.3 shows that 40% of the base-graph nodes are reachable from the network 90% or less of the time. The situation seems to be even worse with the core-graph nodes, which are supposed to be the most stable part of the network (20% of the core-graphs have availability of 90% or less). Base-graph nodes have higher availability because they are closer to users, and is of high interest to users to take care of them. It is interesting to note that 20% of the core-graph nodes have availability between 98-100%, and those are most probably the nodes that comprise the backbone of the network and connect different administrative zones. Since the service placement is done on the core-graph nodes, selecting the nodes with higher availability (e.g., 90-100%) is of high importance.

Figure 4.4 depicts the Empirical Cumulative Distribution Function (ECDF) plot of the node latency. Similar to the availability case, the latency of base-graph nodes is slightly better. For both cases, 30% of the nodes have latency of 480 ms or less, which makes the other 70% of the nodes to have higher latency. The availability and latency graph demonstrate the importance of, and indeed the need for, a more effective, network-aware placement in CNMCs. By not taking the performance of the underlying network into account, applications can end up sending large amounts of data across slow wireless links while faster and more reliable links and nodes remain under-utilized.

In order to see the effects of the network-aware placement in the solutions obtained, we compare two versions of our algorithm. The first version i.e., *Baseline*, allocates services just with the goal of minimizing the service overlay diameter without considering node properties such as availability, latency or closeness. The second version of the algorithm called $PASP$, tries to minimize the service overlay diameter, *while* taking into account these node properties.

The availability and latency of the *Baseline* solutions are calculated by taking the average of nodes in the optimal solutions (after the optimal solution is computed), where the optimal solution is the best solution that minimizes the service overlay diameter, that can only be calculated exhaustively offline.

We allocate services of size 3, 5, 7, 9, 11 and 15. Figure 4.5 and Figure 4.6 reveal that nodes obtained in the solutions with $PASP$ have higher average availability and lower latency than with *Baseline*, with minimum service overlay diameter. On average, the gain of $PASP$ over *Baseline* is 8% for the availability, and 45 ms for the latency (5-20% reduction).

We find that our $PASP$ algorithm is good in finding placement solutions with higher availability and lower latency, however the service solutions obtained might or might not be very close (in terms of number of hops) to base-graph clients. Because of this we also developed another flavour of PASP algorithm called $PASP - closeness$. Figure 4.7 shows the number

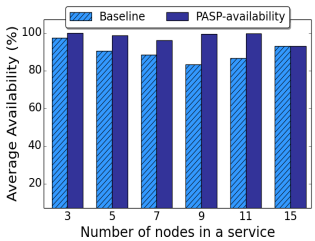


Figure 4.5: PASP-Availab.

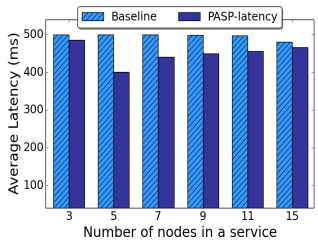


Figure 4.6: PASP-Latency

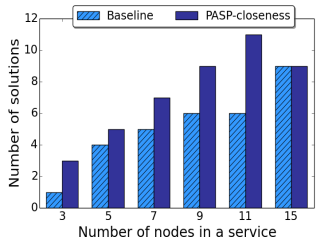


Figure 4.7: PASP-Closeness

of solutions obtained that are 1-hop close to the base-graph clients. When $PASP - closeness$ algorithm allocates three services, on average there are three solutions whose internal nodes (e.g, any of the nodes) are at 1-hop distance to any of the base-graph client nodes, contrary to the *Baseline* where on average there is one solution whose nodes are at 1-hop distance to base-graph clients.

Overall, in the two algorithms, there is a trade-off between latency and closeness. For bandwidth-intensive applications closeness seems to be more important when allocating services (e.g., $PASP - closeness$ can be used), while for latency-sensitive applications it is the latency the one that naturally seems to be more important (e.g., $PASP - latency$ can be used).

4.3.2 Deployment in a real production Community Network

In order to understand the gains of our network-aware service placement algorithm in a real production community network, we deploy our algorithm in real hardware connected to the nodes of the QMP [16o] network, which is a subset of Guifi.net located in the city of Barcelona. Figure 4.8 depicts the topoloqy of the QMP network. Furthermore, a live QMP monitoring page [16n] updated hourly is available in the Internet.

We use 16 servers connected to the wireless nodes of QMP. The nodes and the attached servers are geographically distributed in the city of Barcelona. The hardware of the servers consists of Jetway devices, which are equipped with an Intel Atom N2600 CPU, 4 GB of RAM and 120 GB SSD. They run

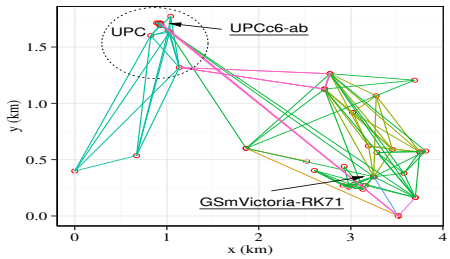


Figure 4.8: QMP topology (2015)

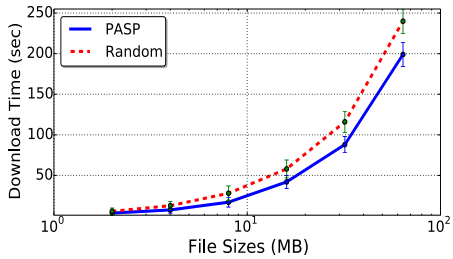


Figure 4.9: Average client reading times

an operating system based on OpenWRT, which allows running several slivers (VMs) on one node simultaneously implemented as Linux containers (LXC).

The slivers host the Cloudy operating system. Cloudy contains some pre-integrated distributed applications, which the community network user can activate to enable services inside the network. Services include a streaming service, a storage service and a folder synchronizing service, among others. For our experiments, we use the storage service, which is based on Tahoe-LAFS. Tahoe-LAFS is an open-source distributed storage system with enforced security and fault-tolerance features, such as data encryption at the client side, coded transmission and data dispersion among a set of storage nodes.

As the controller node we leverage the experimental infrastructure of Community-Lab. Community-Lab provides a central coordination entity that has knowledge about the network topology in real time. Out of the 16 devices used, three of them are storage nodes and 13 of them are clients (chosen randomly) that read files. The clients are located in different geographic locations of the network. The controller is the one that allocates the distributed storage service in these three nodes and clients access this service. On the client side we measure the file reading times. We monitored the network for the entire month of January 2016. The average throughput distribution of all the links for one month period was 9.4 Mbps.

Figure 4.9 shows the average download time for various file sizes (2-64 MB) perceived at the 13 clients, after allocating services using *Random* algorithm

(i.e., currently used at Guifi.net) and using our $PASP$ algorithm. The experiment is composed of 20 runs, where each run has 10 repetitions, and averaged over all the successful runs. Standard deviation error bars are also shown.

Regarding the network interferences that may be caused by other users concurrent activities which can impact the results of our experiments, we reference to our earlier work [CNE13a] which investigated these issues.

Allocation of services using *Random* algorithm by Controller is done without taking into account the performance of the underlying network. It can be seen for instance that when using our $PASP$ algorithm for allocation, it takes around 17 seconds for the clients on average to read a 8 MB file. In the random case, the time is almost doubled, reaching 28 seconds for reading a file from the clients side. We observed therefore that when allocating services, taking into account the closeness and availability parameters in the allocation decision, on average (for all clients) our algorithm reduces the client reading times for 16%. Maximum improvement (around 31%) has been achieved when reading larger files (64 MB). When reading larger files client needs to contact many nodes in order to complete the reading of the file.

4.4 Discussion

From working with the `Guifi.net` data, we found that augmenting the service overlay diameter solutions with service specific metrics has a direct impact on the service performance. We found that the optimal flooding radius in `Guifi.net` is small, i.e. 2 hops. This means that it is always possible to find an optimal service placement with an overlay diameter of 2 hops within Guifi.net. Additionally, the average solutions diameter increases as the number of nodes that composes the services does.

Furthermore, we observed some patterns in the node features that conforms optimal placements. We saw that the solution overlay diameter depends on the nodes degree centrality. Minimum degree centrality can be used to select the first node that composes the service (the solution). We saw that

most of the solutions obtained are concentrated on a small set of of average centrality values. Selecting the next nodes in a particular range of closeness centrality (for bandwidth-intensive services) and betweenness centrality (for latency-sensitive services) is specially useful to obtain more optimal overlays.

4.5 Summary

We addressed the need for network-aware service placement in community network micro-cloud infrastructures. We looked at a specific case of improving the deployment of service instance on micro-servers for enabling an improved distributed storage service in a community network.

As services become more network intensive, the bandwidth, latency etc., between the used nodes becomes the bottleneck for improving performance. In community networks, the limited capacity of nodes and links and an unpredictable network performance becomes a problem for service performance. Network awareness in placing services allows to chose more reliable and faster paths over poorer ones.

In this chapter we introduced a service placement algorithm that provides improved overlay service selection for distributed services considering service quality parameters, without the need for exploring the whole solution space. For our simulations we employed a topological snapshot from Guifi.net to identify node traits in the optimal service placements. We deployed our service placement algorithm in a real network segment of Guifi.net, a production community network, and quantified the performance and effects of our algorithm. We conducted our study on the case of a distributed storage service. We found that our *PASP* algorithm reduces the client reading times by an average of 16% (with a max. improvement of 31%) compared to the currently used organic placement scheme. Our results show how the choice of an appropriate set of nodes, taken from a larger resource pool, can influence service performance significantly.

In next steps we plan to develop and implement a decentralized version of

our investigated service placement algorithm. Service migration should also be addressed to support performance objectives in the case of user mobility and within dynamic changes in the network.

Notes

The research discussed in this Chapter 4 was included in the following publications:

[Sel+16] Mennan Selimi, Davide Vega, Felix Freitag, and Luís Veiga. "Towards Network-Aware Service Placement in Community Network Micro-Clouds". In: *22nd International Conference on Parallel and Distributed Computing (Euro-Par 2016)*. (CORE Rank A). Grenoble, France, 2016, pp. 376–388.

5

Service Placement Heuristic

Community networks (CNs) have gained momentum in the last few years with the increasing number of spontaneously deployed WiFi hotspots and home networks. These networks, owned and managed by volunteers, offer various services to their members and to the public. To reduce the complexity of service deployment, community *micro-clouds* have recently emerged as a promising enabler for the delivery of cloud services to community users. By putting services closer to consumers, micro-clouds pursue not only a better service performance, but also a low entry barrier for the deployment of mainstream Internet services within the CN. Unfortunately, the provisioning of the services is not so simple. Due to the large and irregular topology, high software and hardware diversity of CNs, it requires of a "careful" placement of micro-clouds and services over the network. To achieve this, this chapter proposes to leverage state information about the network to inform service placement decisions (i.e., research question **Q3**), and to do so through a fast heuristic algorithm, which is vital to quickly react to changing conditions. We contribute in this chapter a new placement heuristic called BASP (Band-

width and Availability-aware Service Placement), which uses the state of the underlying community network to optimize service deployment. In particular, it considers two sources of information: i) network bandwidth and ii) device availability to make optimized decisions. Compared with brute-force search, which it takes of the order of hours to complete, BASP runs much faster; it just takes a few seconds, while achieving equally good results.

5.1 Network Characterization

Our service placement strategy considers two aspects: device availability and network bandwidth. As the first step it is vital to understand the behaviour of these two dimensions in a real CN. We achieve this by characterizing a production wireless CN such as a QMP network over a five month period. Our goal is to determine the key features of the network (e.g. bandwidth distribution) and its nodes (e.g. availability patterns) that could help us to design new heuristics for intelligent service placement in CNs.

5.1.1 QMP Network: A Brief Background

QMP network, began deployment in 2009 in a quarter of the city of Barcelona, Spain, called Sants, as part of the *Quick Mesh Project (QMP)* [16o]. QMP is an urban mesh network and it is a subset of the `Guifi.net` CN sometimes called GuifiSants. At the time of writing, QMP has around 71 nodes. There are two gateways (proxies) distributed in the network that connect QMP to the rest of Guifi.net and Internet (see Figure 5.2). A detailed description of QMP can be found in [CNE13a].

Typically, QMP users have an outdoor router (OR) with a Wi-fi interface on the roof, connected through Ethernet to an indoor AP (access point) as a premises network. The most common OR in QMP is the NanoStation M5 as shown in Figure 5.1, which is used to build links on the network and integrates a sectorial antenna with a router furnished with a wireless 802.11an

Figure 5.1: QMP devices

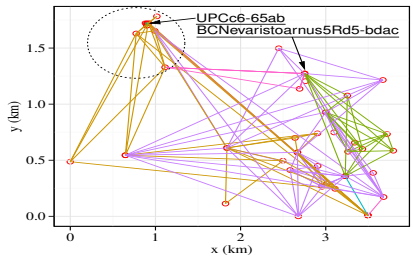


Figure 5.2: QMP network topology (2016)

interface. Some strategic locations have several NanoStations, that provide larger coverage. In addition, some links of several kilometers are set up with parabolic antennas (NanoBridges). ORs in QMP are flashed with the Linux distribution which was developed inside the QMP project which is a branch of OpenWRT and uses BMX6 as the mesh routing protocol [NLN12].

The user devices connected to the ORs consists of Minix Neo Z64 and Jetway mini PCs, which are equipped with an Intel Atom CPU. They run the Cloudy operating system, which allows running services in Docker containers.

Methodology and data collection: Measurements have been obtained by connecting via SSH to each QMP OR and running basic system commands available in the QMP distribution. This method has the advantage that no changes or additional software need to be installed in the nodes. Live measurements have been taken hourly over a 5 months period, starting from July 2016 to November 2016, and our live monitoring page and data is publicly available in the Internet*. We use this data to analyse main aspects of QMP network.

5.1.2 Device Availability

The quality and state of the heterogeneous hardware used in QMP, influences the stability of the links and network performance. Availability of the QMP

*http://dsg.ac.upc.edu/qmpsu/index.php

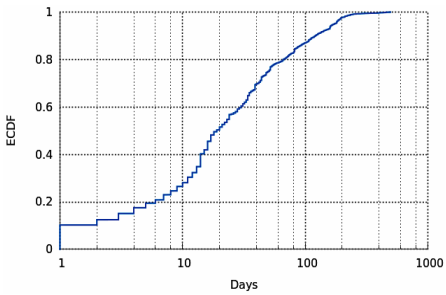


Figure 5.3: Node sysUptime

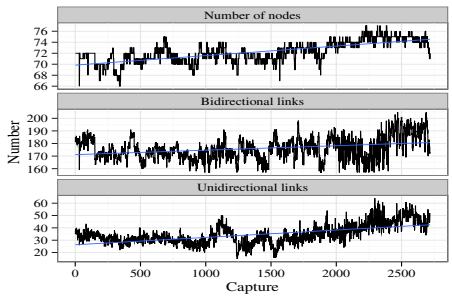


Figure 5.4: Node and link presence

nodes is used as an indirect metric for the quality of connectivity that new members expect from the network.

Figure 5.3 shows the Empirical Cumulative Distribution Function (ECDF) plot of the sysUpTime collected from the SNMP service snapshots from the QMP network, on a random day in 2016. The Linux kernel counter of sysUpTime resets to zero every time a node is rebooted or shut down, independently of the reason that causes this. In a CN such as QMP, users do not tend to deliberately reboot the device unless they have to perform an upgrade, which is not very common. Hence, the number of days reported by the sysUpTime is a relatively good measure of the device availability due to random failures.

We also took into account the last time a node had been rebooted, voluntarily or not, which gave us a direct measure of its availability. Figure 5.3 depicts that, there is a high number of nodes (about 10%) that have been restarted during the last day, while there are also some nodes that had not been reset for almost one year. When we compare the system uptime reported in a similar study and environment on PlanetLab [VP11], a QMP node has a higher probability of being disconnected or not reachable from the network. The fact that PlanetLab showed a higher average sysUpTime on its nodes may be because it is an experimental testbed running on much more stable computers and environment. Furthermore, the QMP members are not only responsible for the maintenance of their nodes, but also for ensuring a

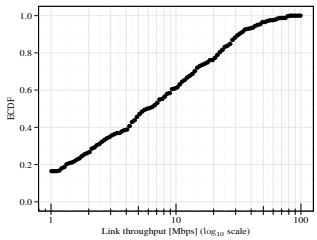


Figure 5.5: Bandwidth distribution

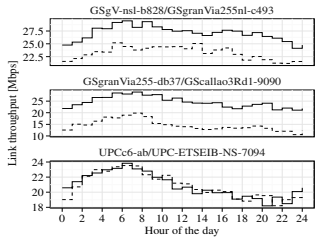


Figure 5.6: Bandwidth in the three busiest links

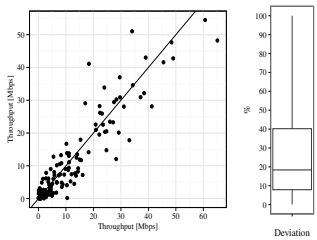


Figure 5.7: Bandwidth asymmetry

minimum standard of connectivity with other parts of the network.

Figure 5.4 depicts the node and link presence during captures. We define the presence as the percentage a given node or link is observed (i.e., is reachable) over the captures. A capture is an hourly network snapshot that we take from the QMP network for a five months period (2718 captures in total). Figure shows that QMP is growing. Overall, 77 different nodes were detected. From those, 71 were alive during the entire measurement period. Around 6 nodes were missed in the majority of the captures. These are temporarily working nodes from other mesh networks and laboratory devices used for various experiments. Figure 5.4 also reveals that on average 175 of links used between nodes are bidirectional and 34 are unidirectional. For bidirectional links, we count both links in opposite direction as a single link.

In summary, device availability is important to identify those nodes that will minimize service interruptions over time. Based on the measurements, we assign availability scores to each of the devices. The high available devices are the possible candidates for deploying on them the micro-clouds.

5.1.3 Bandwidth characterization

A significant amount of applications that run on QMP and Guifi.net network are network-intensive (bandwidth and delay sensitive), transferring large amounts of data between the network nodes [Sel+15c]. The performance of such kind of applications depends not just on computational and disk re-

sources but also on the network bandwidth between the nodes on which they are deployed. Therefore, the placement of such services in the network is of high importance.

We characterize the wireless links of the QMP network by studying their bandwidth. Figure 5.5 shows the average bandwidth distribution of all the links. The figure shows that the link throughput can be fitted with a mean of 21.8 Mbps. At the same time Figure 5.5 reveals that the 60% of the nodes have 10 Mbps or less throughput. The average bandwidth of 21.8 Mbps obtained in the network can allow many popular bandwidth-hungry service to be run without big interruptions. This high performance can be attributed to the 802.11an devices used in the network.

In order to see the variability of the bandwidth, Figure 5.6 shows the bandwidth averages in both directions (upload vs. download) of the three busiest links. The nodes of three busiest links are highlighted on the top of the figure. We noted that that the asymmetry of the bandwidths measured in both directions it not always due to the asymmetry of the user traffic (not shown in the graphs). For instance, node GSgranVia255, around 6am, when the user traffic is the lowest and equal in both directions, the asymmetry of the links bandwidth observed in Figure 5.6 remains the same. We thus conclude that even though bandwidth time to time is slightly affected by the traffic, the asymmetry of the links that we see might be due to the link characteristics, as level of interferences present at each end, or different transmission powers.

In order to measure the links asymmetry, Figure 5.7 depicts the bandwidth measured in each direction. A boxplot of the absolute value of the deviation over the mean is also depicted on the right. The figure shows that around 25% of the links have a deviation higher than 40%. At the same time, the other 25% of the links have a deviation less than 10%. After performing some measurements regarding the signaling power of the devices, we discovered that some of the community members have re-tuned the radios of their devices, trying to achieve better performance (transmission power, channel and other

parameters), thus, changing the characteristics of the links. Thus, we can conclude that the symmetry of the links, an assumption often used in the literature of in wireless mesh networks, is not very realistic for our case and service placement algorithms definitely need to take into account.

5.1.4 Observations

Here are some observations (features) that we have derived from the measurements in QMP network:

Dynamic Topology: QMP network is highly dynamic and diverse due to many reasons, e.g., its community nature in an urban area; its decentralised organic growth with extensive diversity in the technological choices for hardware, wireless media, link protocols, channels, routing protocols etc.; its mesh nature in the network etc. The current network deployment model is based on geographic singularities rather than QoS. The network is not scale-free. The topology is organic and different w.r.t. conventional ISP network.

Non-uniformly distributed resources: The resources are not uniformly distributed in the network. Wireless links are with asymmetric quality for services (25% of the links have a deviation higher than 40%). We observed a highly skewed traffic pattern and highly skewed bandwidth distribution (Figure 5.5).

Currently used organic (random) placement scheme in QMP and Guifi.net in general, is not sufficient to capture the dynamics of the network and therefore it fails to deliver the satisfying QoS. The strong assumption under random service placement, i.e., uniform distribution of resources, does not hold in such environments.

Furthermore, the services deployed have different QoS requirements. Services that require intensive inter-component communication (e.g streaming service), can perform better if the replicas (service components) are placed close to each other in high capacity links [Sel+15b]. On other side, bandwidth-intensive services (e.g., distributed storage, video-on-demand) can perform

much better if their replicas are as close as possible to their final users (e.g., overall reduction of bandwidth for service provisioning) [Sel+16].

Our goal is to build on this insight and design a network-aware service placement algorithm that will improve the service quality and network performance by optimizing the usage of scarce resources in CNs such as bandwidth.

5.2 Context and Problem

First we describe our model for network and service graph. Subsequently we build on this to describe the service placement problem. The symbols used are listed in Table 5.1.

5.2.1 Network Graph

The deployment and sharing of services in CNs is made available through *community network micro-clouds* (CNMCs). The idea of CNMC is to place the cloud at the edge closer to community end-users, so users can have fast and reliable access to the service. To reach its full potential, a CNMC needs to be carefully deployed in order to utilize the available bandwidth resources.

In a CNMC, a server or low-power device (i.e, home gateway) is directly connected to the wireless base-station (ORs) providing cloud services to users that are either within a reasonable distance or directly connected to base-station.

We call the CN the *underlay* to distinguish it from the *overlay* network which is built by the services. The underlay network is supposed to be connected and we assume each node knows whether other nodes can be reached (i.e., next hop is known). We can model the underlay graph as: $G \leftarrow (N, E)$ where N is the set of nodes connected to the outdoor routers (ORs) present in the CNs and E is the set of wireless links that connects them. Physical links between nodes are characterized by a given bandwidth (B_i). Furthermore, each link has a bandwidth capacity (B_e). Each node in the network has an

Symbol	Description
N	set of physical nodes in the network
E	set of edges (physical links) in the network
S	set of services
D	set of service copies
k	max number of service copies
B_e	bandwidth capacity of link e
$\beta_{s1,s2}$	bandwidth requirement between services s1 and s2
R_n	Availability of node n
λ	Availability threshold
$X_{s1,s2}$	use of physical link e by at least one service for the placement of virtual link between s1 and s2, 1 iff placed

Table 5.1: Input and decision variables

availability score (R_n) derived from the real measurements at QMP.

5.2.2 Service Graph

The services aimed in this work are at infrastructure level (IaaS), as cloud services in current dedicated datacenters. Therefore, the services are deployed directly over the core resources of the network and accessed by clients. Services can be deployed by QMP users or administrators.

The services we consider in this work are distributed services (i.e, independently deployable services as in Microservices Architecture). The distributed services can be composite services (non-monolithic) built from simpler parts, e.g., video streaming (built from the source and peers component), web service (built from database, memcached and client component) etc. In the real deployment, one service component corresponds to one Docker container. These parts or components of service would create an overlay and interact with each other to offer more complex services. Bandwidth requirement between two services s_1 and s_2 is given by $\beta_{s1,s2}$. At most k copies can be placed for each

service s.

A service may or may not be tied to a specific node of the network. Each node can host one or more type of services. In this work we assume an offline service placement approach where a single or a set of applications are placed "in one shot" onto the underlying physical network. We might rearrange (migrate) the placement of the same service over the time because of the service performance fluctuation (e.g. weather conditions, node availability, changes in use pattern, and etc.). We do not consider real-time service migration.

5.2.3 Service Placement Problem

The concept of service and network graph allows us to formulate the problem statement more precisely as: *"Given a service and network graph, how to place a service on a network as to maximize user QoS and QoE, while satisfying a required level of availability for each node (N) and considering a maximum of k service copies ?*

Let B_{ij} be the bandwidth of the path to go from node i to node j. We want a partition of k clusters (i.e., services) : $C \leftarrow C_1, C_2, C_3, ..., C_k$ of the set of nodes in the mesh network. The cluster head i of cluster C_i is the location of the node where the service will be deployed. The partition maximizing the bandwidth from the cluster head to the other nodes in the cluster is given by the objective function:

$$\arg\max_C \sum_{i=1}^{k} \sum_{j \in Ci} B_{ij} \tag{5.1}$$

with respect to the following constraints:

1. The total bandwidth used per link cannot exceed the total link capacity:

$$\forall e \in \mathbb{E} : \sum_{s1,s_2 \in \mathbb{S}} X_{s1,s2}(e) \times \beta_{s1,s2} \leq B_e \tag{5.2}$$

2. Availability-awareness: the node availability should be higher than the predefined threshold λ:

$$\forall n \in \mathbb{N} : \sum_{n \in \mathbb{N}} R_n \geq \lambda \tag{5.3}$$

3. Admission control: At most, k copies can be placed for each service:

$$|D| = k \tag{5.4}$$

5.2.4 Proposed Algorithm: BASP

Solving the problem stated in Equation 5.1 in brute force for any number of N and k is NP-hard and very costly. The naive brute force method can be estimated by calculating the *Stirling number of the second kind* [16r] which counts the number of ways to partition a set of n elements into k nonempty subsets, i.e., $\frac{1}{k!}\sum_{j=0}^{k}(-1)^{j-k}\binom{n}{k}j^n \Rightarrow \mathcal{O}(n^k k^n)$. Thus, due to the obvious combinatorial explosion, we propose a low-cost and fast heuristic called BASP. The BASP (Bandwidth and Availability-aware Service Placement) allocates services taking into account the bandwidth of the network and the device availability.

Our BASP algorithm (see Algorithm 5.1) runs in three phases:

1. **Phase 1: K-Means:** Initially, we use the naive K-Means partitioning algorithm in order to group nodes based on their geo-location. The idea is to get back clusters of nodes that are close to each other. The K-Means algorithm forms clusters of nodes based on the Euclidean distances between them, where the distance metrics in our case are the geographical coordinates of the nodes. In traditional K-Means algorithm, first, k out of n nodes are randomly selected as the cluster heads (centroids). Each of the remaining nodes decides its cluster head nearest to it according to the Euclidean distance. After each of the

nodes in the network is assigned to one of k clusters, the centroid of each cluster is re-calculated. Grouping nodes based on geo-location is in line with how the QMP is organized. The nodes in QMP are organized into a tree hierarchy of *zones* [al15]. A zone can represent nodes from a neighborhood or a city. Each zone can be further divided in child zones that cover smaller geographical areas where nodes are close to each other. From the service perspective we consider placements inside a particular zone. We use K-Means with geo-coordinates as an initial heuristic for our algorithm. As an alternative, clustering based on network locality can be used. Several graph community detection techniques are available for our environment. [LF09].

2. **Phase 2: Aggregate Bandwidth Maximization:** The second phase of the algorithm is based on the concept of finding the cluster heads maximizing the bandwidth between them and their member nodes in the clusters C_k formed in the first phase. The bandwidth between two nodes is estimated as the bandwidth of the link having the minimum bandwidth in the shortest path. The cluster heads computed are the candidate nodes for the service placement. This is plotted as Naive K-Means in the Figure 5.8.

3. **Phase 3: Cluster Re-Computation:** The third and last phase of the algorithm includes reassigning the nodes to the selected cluster heads having the maximum bandwidth, since the geo-location of nodes in the clusters formed during phase one is not always correlated with their bandwidth. This way the clusters are formed based on nodes bandwidth. This is plotted as BASP in the Figure 5.8.

Complexity: The complexity of the BASP is as follows: for BASP, finding the optimal solution to the k-means (i.e., phase one) clustering problem if k and d (the dimension) are fixed (e.g., in our case $n = 71$, and $d = 2$), the problem can be exactly solved in time $\mathcal{O}(n^{dk+1} \log n)$, where n is the number

of entities to be clustered. The complexity for computing the cluster heads in phase two is $\mathcal{O}(n^2)$, and $\mathcal{O}(n)$ for the reassigning the clusters in phase three. Therefore, the overall complexity of BASP is polylogarithmic $\mathcal{O}(n^{2k+1} \log n)$, which is significantly smaller than the brute force method and thus practical for commodity processors.

5.3 Evaluation

5.3.1 Setup

We take a network snapshot (capture) from 71 physical nodes of the QMP network regarding the bandwidth of the links[†] and node availability. The node and bandwidth data obtained has been used to build the topology graph of the QMP. The QMP topology graph is constructed by considering only operational nodes, marked in Working status, and having one or more links pointing to another node. Additionally, we have discarded some disconnected clusters. The links are bidirectional and unidirectional, thus we we use a directed graph. The nodes of QMP consists of Intel Atom N2600 CPU, 4GB of RAM and 120 GB of disk space.

Our experiment is comprised of 5 runs and the presented results are averaged over all the runs. Each run consists of 15 repetitions.

5.3.2 Comparison

To emphasise the importance of the different phases of Algorithm 5.1, we compare in this section the two phases of our heuristic with *Random Placement*, i.e., the default placement at QMP.

Random Placement: Currently, the service deployment (much as network deployment) at QMP is not centrally planned but initiated individually by the CN members. Public, user and community-oriented services are placed

[†]http://tomir.ac.upc.edu/qmpsu/index.php?cap=56d07684

randomly on supernodes and users' premises, respectively. The only parameter taken into account when placing services is that the devices must be in "production" state. The network is not taken into consideration at all. All nodes in the production state appear equally to the users.

NaiveKMeans Placement: This corresponds to the second phase of Algorithm 5.1. The service is installed on the node having the maximum bandwidth on the initial clusters formed by K-Means. We limit the choice of the cluster heads to be inside the sets of clusters obtained using K-Means

BASP Placement: It includes the three phases of Algorithm 5.1. The service is placed on the node having the maximum bandwidth after the clusters are re-computed.

5.3.3 Results

Figure 5.8 depicts the average bandwidth to the cluster heads obtained with the *Random*, *Naive K-Means* and the *BASP* algorithm. This figure reveals that for any number of services k, *BASP* outperforms both *Naive K-Means* and *Random* placement.

For $k = 2$, the average bandwidth to the cluster heads is increased from 18.3 Mbps (*Naive K-Means*) to 27.7 Mbps (*BASP*), which represents a 50% improvement. The biggest increase of 67% is achieved when $k = 7$. On average, when having up to 7 services in the network, the gain of *BASP* over *Naive K-Means* is of 45%. Based on the observations from Figure 5.8, the gap between the two algorithms grows as k increases. We observe that k will increase as the network grows. And hence, BASP will presumably render better results for larger networks than the rest of strategies.

Regarding the comparison between *BASP* and *Random* placement, we find that *Random* placement leads to an inefficient use of network's resources, and consequently to suboptimal performance. The gain of *BASP* over naive *Random* placement is of 211%.

Comparison to the optimal solution. Note that our heuristic enables us

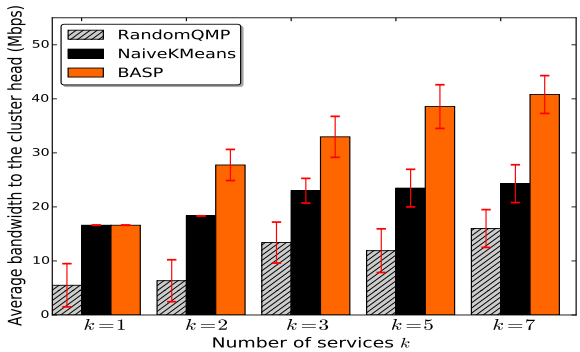


Figure 5.8: Average bandwidth to the cluster heads

to select cluster heads that provide much higher bandwidth than any other random or naive approach. But, if we were about to look for the optimum bandwidth within the clusters (i.e., optimum average bandwidth for the cluster), then this problem would be NP-hard. The reason is that finding the optimal solution entails running our algorithm for all the combinations of size k from a set of size n. This is a combinatorial problem that becomes intractable even for small sizes of k or n (e.g., $k = 5$, $n = 71$). For instance, if we wanted to find the optimum bandwidth for a cluster of size $k = 3$, then the algorithm would need to run for every possible (non-repeating) combination of size 3 from a set of 71 elements, i.e., $choose(71, 3) = 57K$ combinations. We managed to do so and found that the optimum average was 62.7 Mbps. For $k = 2$, the optimum was of 49.1 Mbps. For $k = 1$, it was of 16.9 Mbps.

The downside was that, the computation of the optimal solution took very long time in a commodity machine. Concretely, it took 5 hours for $k = 3$ and 30 minutes for $k = 2$. Instead, *BASP* spent only 23 seconds for $k = 3$ and 15 seconds for $k = 2$. Table 5.9 shows the improvement of *BASP* over *Random* and *Naive K-Means*. To summarize, BASP is able to achieve good bandwidth performance with very low computation complexity.

Correlation with centrality metrics. Figure 5.9 shows some centrality measures and some graph properties obtained for each cluster head. Also,

	$k=1$	$k=2$		$k=3$			$k=5$				
Clusters [node id]	C1 [27]	C1 [20]	C2 [39]	C1 [20]	C2 [39]	C3 [49]	C1 [20]	C2 [4]	C3 [49]	C4 [51]	C5 [39]
Head degree	20	6	6	6	6	10	6	10	10	12	6
Neighborhood Connectivity	7.7	9.6	9.6	**9.6**	**9.6**	**10.8**	9.6	8.7	10.8	8.1	9.6
Diameter	6	5	3	4	3	5	4	2	3	1	3
RandomQMP - Bandwidth [Mbps]	5.3	6.34		13.4			11.9				
Naive K-Means Bandwidth [Mbps]	16.6	18.3		23			23.4				
BASP - Bandwidth [Mbps]	16.9	27.7		32.9			38.5				
BASP - Running Time	7 sec	15 sec		23 sec			30 sec				

Figure 5.9: Centrality measures for cluster heads

Figure 5.10 shows the neighborhood connectivity graph of the QMP network. The neighborhood connectivity of a node v is defined as the average connectivity of all neighbors of v. In the figure, nodes with low neighborhood connectivity values are depicted with bright colors and high values with dark colors. It is interesting to note that some the nodes with the highest neighborhood connectivity are those chosen by BASP as cluster heads. The cluster heads (for $k = 2$ and $k = 3$) are illustrated with a rectangle in the graph. A deeper investigation into the relationship between service placement and network topological properties is out of the scope of this work and will be reserved as our future work.

5.4 Experimental Evaluation

5.4.1 Cloudy: A Service Hub

In order to foster the adoption and transition of the community micro-cloud environment, we provide a community cloud distribution, codenamed Cloudy. This distribution contains the platform and application services of the community cloud system. Cloudy is the core software of our micro-clouds, because it unifies the different tools and services of the cloud system in a Debian-based Linux distribution. Cloudy is open-source and can be downloaded from public repositories[‡].

Cloudy's main components can be considered a layered stack, with services residing both inside the kernel and at the user level. Figure 5.11 reports some

[‡]http://repo.clommunity-project.eu/images/

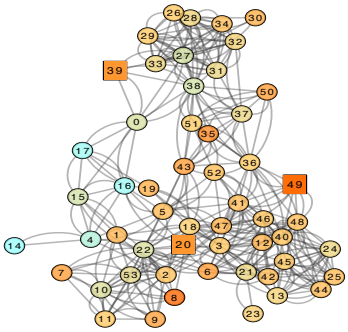


Figure 5.10: Neighborhood connectivity in QMP network

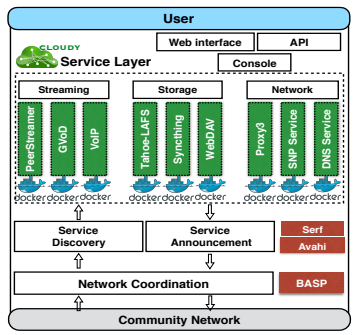


Figure 5.11: Cloudy architecture

of the available services running on Docker containers. Cloudy includes a tool for users to announce and discover services in the micro-clouds based on Serf, which is a decentralized solution for cluster membership and orchestration. On the network coordination layer, having sufficient knowledge about the underlying network topology, *BASP* decides about the placement of the service which then is announced via Serf (see Figure 5.11). Thus, the service can be discovered by the other users.

5.4.2 Evaluation in Real Production Community Network

In order to understand the gains of our network-aware service placement algorithm in a real production CN, we deploy our algorithm in real hardware connected to the nodes of the QMP network, located in the city of Barcelona. We concentrate on benchmarking two of the most popular network-intensive applications: *Video streaming service*, and *Web 2.0 Service* performed by the most popular websites.

Live-video streaming service

PeerStreamer, an open source live P2P video streaming service, has been paradigmatically established as the live streaming service in Cloudy. This

service is based on *chunk diffusion*, where peers offer a selection of the chunks they own to some peers in their neighborhood. A chunk consists of a part of the video to be streamed (by default, this is one frame of the video). PeerStreamer differentiates between a source node and peer node. A source node is responsible for sending the video chunks to the peers in the network. In our case, both the source nodes and peers run in Docker containers atop the QMP nodes.

Setup: We use 20 real nodes connected to the wireless nodes of QMP. These nodes are co-located in either users homes (as home gateways, set-top-boxes, etc.) or within other infrastructures distributed around the city of Barcelona. They run the Cloudy operating system. As the controller node, we leverage the experimental infrastructure of Community-Lab. Community-Lab provides a central coordination entity that has knowledge about the network topology in real time and allows researchers to deploy experimental services and perform experiments in a production CN. The nodes of QMP that are running the live video streaming service are part of Community-Lab.

In our experiments, we connect a live streaming camera (maximum bitrate of 512 kbps, 30 frame-per-second) to a local PeerStreamer instance that acts as a source node. The source is running in a Docker container. The source is also responsible for converting the video stream into chunks that are sent to the peers. In the default configuration of PeerStreamer, each chunk contains a single video frame.

The location of the source in such a dynamic network is therefore crucial. Placing the source in a QMP node with weak connectivity will negatively impact the QoS and QoE of viewers. In order to determine the accuracy of *BASP* upon choosing the appropriate QMP node where to host the source, we measure the average chunk loss percentage at the peer side, which is defined as the percentage of chunks that were lost and not arrived in time. This simple metric will help us understand the role of the network on the reliable operation of live-video streaming over a CN.

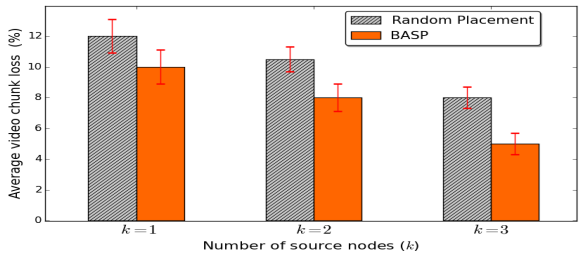


Figure 5.12: Average video chunk loss in QMP

Our experiment is composed of 20 runs, where each run has 10 repetitions. Results are averaged over all the successful runs. 90% of them were successful. In the 10% of failed runs, the source was unable to stream the captured images from the camera, so peers did not receive the data. This experiment was run for 2 weeks, with roughly 100 hours of live video data and several MBytes of logged content. The presented results are from one hour of continuous live streaming from the PeerStreamer source.

Results: Figure 5.12 shows the average chunk loss for an increasing number of sources k. The data reveals that for any number of source nodes k, BASP outperforms the currently adopted random placement in QMP network. For $k = 1$, BASP decreases the average chunk loss from 12% to 10%. This case corresponds to the scenario where there is one single source node streaming to the 20 peers in the QMP network. Based on the observations from Figure 5.12, the gap between the two algorithms is growing as k increases. For instance, when $k = 3$, we get a 3% points of improvement w.r.t. chunk loss, and a significant 37% reduction in the loss packet rate.

Web 2.0 Service

The second type of service that we experiment is the Web 2.0 Service. The workloads of Web2.0 websites differ from the workloads of older generation websites. Older generation websites typically served static content, while Web2.0 websites serve dynamic content. The content is dynamically gener-

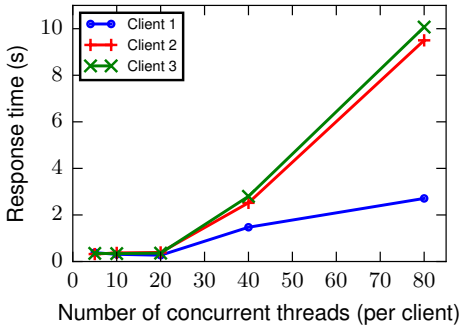


Figure 5.13: UpdateActivity when web server placed Randomly

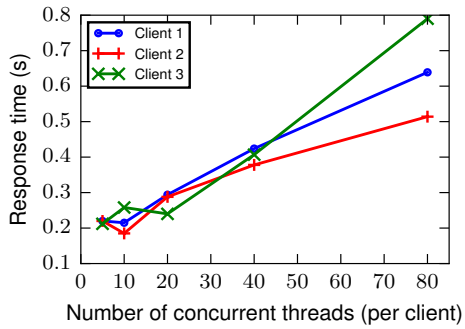


Figure 5.14: UpdateActivity when web server placed with BASP

ated from the actions of other users and from external sources, such as news feeds from other websites. Because of this, writes to the backend database (a NoSQL data store) are frequent, and the data written are consumed by other users. We are experimenting with a Web 2.0 service, which is an example of a Microservices architecture, since it is formed by a group of service components (i.e., web server, database server, memcached server and clients). In this type of service, independent client requests are accepted by a stateless web server which serves files and content through the middleware in backend databases such as cloud NoSQL data stores or traditional relational SQL servers. The placement of the web server (together with the database server) is of high importance.

Setup: For the evaluation, we use the dockerized version of the CloudSuite Web Serving benchmark [PSF16]. Cloudsuite benchmark has four tiers: the web server, the database server, the memcached server, and the clients. Each tier has its own Docker image. The web server runs Elgg and it connects to the memcached server and the database server. The Elgg social networking engine is a Web2.0 application developed in PHP, similar in functionality to Facebook. The clients (implemented using the Faban workload generator) send requests to login to the social network and perform different operations. We use 10 QMP nodes in total, where 3 of them act as clients. The other 7

nodes are candidates for deploying the web server. The web server, database server and memcached server are always collocated in the same host. On the client side, we measure the response time when performing some operations such as login, updating the life feed, sending messages, etc. In Cloudsuite, to each operation is assigned an individual QoS latency limit. If less than 95% of the operations meet the QoS latency limit, the benchmark is considered to be failed (marked as F). The location of the web server, database server and memcache server has a direct impact on the client response time.

Results: Figure 5.13 and Figure 5.14 depict the response time observed by the 3 clients for the update live feed operation for *Random* and *BASP*, respectively.

Figure 5.13 reveals that, as far as we increase the number of threads (i.e., concurrent operations) per client, the response time observed in the 3 clients increases drastically. For up to 120 operations per client (i.e., 20 threads), the 3 clients perceive a similar response times (300-350 ms). Response time increases more than one order of magnitude in Client 2 and Client 3 when performing 160 operations (i.e. 80 threads), and one order of magnitude in Client 1, respectively.

Operations	Update live feed				Do login			
Threads	10	20	40	80	10	20	40	80
QMP-Random	T	F	F	F	T	T	F	F
QMP-BASP	T	T	T	F	T	T	T	F
Stdev	0.02s	0.03s	0.01	0.01	0.02	0.02	0.01s	0.03s
Improvement	0.1s	0.2s	1.8s	6.7s	0.1s	0.1s	1.2s	4.2s

Table 5.2: Cloudsuite benchmark results

Figure 5.14 depicts that, the client response times for higher workloads, decreases an order of magnitude when using our *BASP* heuristic compared with the *Random* approach, as shown in Figure 5.13. For up to 120 operations per client, the response times that the 3 clients observed is slightly better (200-280 ms) than the response time when the web server is deployed with

the *Random* approach. Furthermore, Table 5.2 demonstrates the successful and failed tests for the update and login operations in the Cloudsuite benchmark. Table reveals, that using the *BASP* heuristic the number of successful tests i.e., those that met the QoS latency limit, is higher than than the number of successful tests with *Random* approach. Furthermore, it also shows the standard deviation values and average client response time improvements when using *BASP* heuristic over *Random* approach. We can notice that the gain brought by the *BASP* heuristic is higher for more intensive workloads.

5.5 Discussion

We addressed the problem of workload placements in Community Network Micro-Clouds (i.e. IaaS-like infrastructures where resources are provided by citizens/ organizations and interconnected through a community network). The major issue of such a platforms is the uncertainties regarding the resources as we observed in the network measurements (distributed, heterogeneous and constantly changing environment) that make the efficient allocation of workloads challenging. Comparing to other works done in this field, this work takes into account the network network characteristics i.e., bandwidth, while most of the service placement approaches generally consider only CPU and memory requirements. Second, it is applied to wireless networks, with all their specific challenges, including range and stability, whereas network-aware service placement are often applied to more traditional wired networks.

Bandwidth is a scarce resource in community networks. Augmenting the bandwidth metric with the link quality prediction metric i.e., technique that surpasses link quality tracking by foreseeing which links are more likely to change its quality, can help more accurately to places services in the network.

5.6 Summary

In this chapter, we motivated the need for bandwidth and availability-aware service placement on CN micro-cloud infrastructures. CNs provide a perfect scenario to deploy and use community services in contributory manner. Previous work done in CNs has focused on better ways to design the network to avoid hot spots and bottlenecks, but did not relate to schemes for network-aware placement of service instances.

However, as services become more network-intensive, they can become bottle-necked by the network, even in well-provisioned clouds. In the case of CN micro-clouds, network awareness is even more critical due to the limited capacity of nodes and links, and an unpredictable network performance. Without a network-aware system for placing services, locations with poor network paths may be chosen while locations with faster, more reliable paths remain unused, resulting ultimately in a poor user experience.

We proposed a low-complexity service placement heuristic called BASP to maximise the bandwidth allocation when deploying a CN micro-clouds. We presented algorithmic details, analysed its complexity, and carefully evaluated its performance with realistic settings. Our experimental results show that BASP consistently outperforms the currently adopted random placement in Guifi.net by 211%. Moreover, as the number of services increases, the gain tends to increase accordingly. Furthermore, we deployed our service placement algorithm in a real network segment of QMP network, a production CN, and quantified the performance and effects of our algorithm. We conducted our study on the case of a live video streaming service and Web 2.0 Service integrated through Cloudy distribution. Our real experimental results show that when using BASP algorithm, the video chunk loss in the peer side is decreased up to 3% points, i.e., worth a 37% reduction in the loss packet rate. When using the BASP with the Web 2.0 service, the client response times decreased up to an order of magnitude, which is a significant improvement.

As a future work, we plan to look into service migration, i.e, the controller

needs to decide which micro-cloud should perform the computation for a particular user, with the presence of user mobility and other dynamic changes in the network. In this problem, the user may switch between micro-clouds thus another question is whether we should migrate the service from one micro-cloud to another when the user location or network condition changes.

Notes

The research discussed in this Chapter 5 was included in the following publications:

[Sel+16] Mennan Selimi, Llorenç Cerdà-Alabern, Liang Wang, Arjuna Sathiaseelan, Luís Veiga, and Felix Freitag. "Bandwidth-aware Service Placement in Community Network Micro-Clouds". In: *41st IEEE Conference on Local Computer Networks (LCN 2016)*. (CORE Rank A) (Short paper). Dubai, UAE, 2016.

Algorithm 5.1 B A S P

Input: $G(N, E)$ ▷ Network graph
$C \leftarrow C_1, C_2, C_3, ..., C_k$ ▷ k partition of clusters
B_i ▷ bandwidth of node i
R_n, λ ▷ availability of node n, λ availability threshold

1: **procedure** PERFORMKMEANS(G, k)
2: **if** $R_n \geq \lambda$ **then**
3: **return** C
4: **end if**
5: **end procedure**
6: **procedure** FINDCLUSTERHEADS(C)
7: $clusterHeads \leftarrow list()$
8: **for all** $k \in C$ **do**
9: **for all** $i \in C_k$ **do**
10: $B_i \leftarrow 0$
11: **for all** $j \in setdiff(C, i)$ **do**
12: $B_i \leftarrow B_i + estimate.route.bandw(G, i, j)$
13: **end for**
14: $clusterHeads \leftarrow \max B_i$
15: **end for**
16: **end for**
17: **return** $clusterHeads$
18: **end procedure**
19: **procedure** RECOMPUTECLUSTERS($clusterHeads, G$)
20: $C' \leftarrow list()$
21: **for all** $i \in clusterHeads$ **do**
22: $cluster_i \leftarrow list()$
23: **for all** $j \in setdiff(G, i)$ **do**
24: $B_j \leftarrow estimate.route.bandw(G, j, i)$
25: **if** B_j is best from other nodes i **then**
26: $cluster_i \leftarrow j$
27: **end if**
28: $C' \leftarrow cluster_i$
29: **end for**
30: **end for**
31: **return** C'
32: **end procedure**

6

Conclusion

The work on this thesis has focused on the service placement problem in community network *micro-cloud* environment, containing low-resource devices. The hierarchical structure of community network *micro-clouds* (CNMCs) allows us to exploit and develop new algorithms for service placement problem which are more efficient than existing in-place approaches. Meanwhile, CNMCs bring in much more infrastructure and user dynamics compared to a traditional cloud environment and therefore we have proposed low-complexity heuristics in order service placement algorithms to cope with these dynamics.

As our contribution in this field, first we have presented the current state of service deployment in `Guifi.net` community network and in-depth performance assessment of three type of services in these environments such as distributed storage, live video-streaming and service discovery. Based on the performance and feasibility studies that we have performed on these services, we proposed service placement algorithms for the community network environment.

We conducted extensive evaluations, employing simulations and real-world

experiments and were able to demonstrate that our placement algorithms, in particular *PASP* and *BASP* outperform other approaches in a variety of scenarios. When compared to the performance of a service implemented in the form of traditional client/service architecture, i.e., random placement, without service placement approach, our approach significantly improves the quality with which the service is provided to clients (i.e., response time) while at the same time reducing the bandwidth.

6.1 Future Work

The service placement problem presented in this work addresses the major concerns of placing a service in the wireless community networks. Given that our algorithm has proven itself in a variety of experiments, it can be expected to serve as a strong basis to investigate the topics listed in this section.

6.1.1 Composite Service Placement

The main focus of this work was the service placement of the monolithic services. Our current work does not cover the placement of composite services. The placement of composite services is more complex and depends largely on the interactions and the semantics between the subservices. This requires more sophisticated placement algorithms which include the flow of information between instances of subservices into their model of the network. In order to achieve this, we think we need to learn how the subservices interact with each other through observations at run time.

6.1.2 Distributed Decision Making

In our algorithms presented in the thesis, decisions on service placement are made by a centralized controller (i.e., client node or Community-Lab controller). This is very practical because we can always see the controller as a service running at one or multiple *micro-clouds*. However, it is desirable

to have a distributed control mechanism for the sake of robustness. Issues regarding distributed service placement can be studied in the future, where randomization techniques such as in [Nee16] can be used.

6.1.3 Service Migration

As the network topology, network performance or volume of service requests from different clients change over time, the current service configuration needs to be adapted to the new situation. This means migrating the services from one node to another one. However, migrating services or creating new service instances incurs additional network traffic. For this reason, the exact circumstances under which an adaptation actually makes sense are not trivial if the goal is the overall reduction of network traffic. Issues regarding the service migration in edge-clouds can be studied as in [Urg+15].

6.1.4 Security

Distributed service provisioning is obviously problematic from a security perspective. Instead of only having to trust a single central instance, each client essentially has to extend its trust to all nodes of the community network *micro-cloud* since each node is a potential service hots. We have not addressed this significant aspect of the service placement problem in our thesis, and this can be studied in the future.

Bibliography

[16a] *AWMN - Athens Wireless Metropolitan Network*. 2016. URL: http://www.awmn.net/ (cit. on p. 31).

[Bai+16] Roger Baig, Lluís Dalmau, Ramon Roca, Leandro Navarro, Felix Freitag, and Arjuna Sathiaseelan. "Making Community Networks Economically Sustainable, the Guifi.Net Experience". In: *Proceedings of the 2016 Workshop on Global Access to the Internet for All*. GAIA '16. Florianopolis, Brazil: ACM, 2016, pp. 31–36 (cit. on p. 2).

[16b] *Ceph - distributed object store*. 2016. URL: http://ceph.com/ (cit. on p. 19).

[Clo16] *Cloudy GNU/Linux Distribution*. 2016. URL: http://cloudy.community (cit. on p. 33).

[16c] *Getinconf tool*. 2016. URL: https://github.com/Clommunity/getinconf (cit. on p. 36).

[16d] *Gluster - scalable network file system*. 2016. URL: https://www.gluster.org/ (cit. on p. 19).

[16e] *Guifi TV*. 2016. URL: http://project.Guifi.net/projects/Guifitv (cit. on p. 32).

[16f] *Guifi.net CNML file*. 2016. URL: https://guifi.net/en/guifi/cnml/2413 (cit. on p. 68).

[16g] *Guifi.net: Services of Catalunya (by zone)*. 2016. URL: https://guifi.net/en/node/2413/view/services (cit. on p. 29).

[16h] *Guinux*. 2016. URL: https://guifi.net/en/node/29320 (cit. on p. 29).

[16i] *IOzone: a filesystem benchmark tool*. 2016. URL: http://www.iozone.org/ (cit. on p. 41).

[Mac+16] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bong Jun Ko, and Theodoros Salonidis. "Migrating Running Applications Across Mobile Edge Clouds: Poster". In: *Proceedings of the 22Nd Annual International Conference on Mobile Computing and Networking*. MobiCom '16. New York City, New York: ACM, 2016, pp. 435–436 (cit. on p. 26).

[Nee16] M. J. Neely. "Distributed Stochastic Optimization via Correlated Scheduling". In: *IEEE/ACM Transactions on Networking* 24.2 (Apr. 2016), pp. 759–772 (cit. on p. 111).

[16j] *OpenStack Swift*. 2016. URL: `https://www.swiftstack.com/product/openstack-swift` (cit. on p. 18).

[PSF16] Tapti Palit, Yongming Shen, and Michael Ferdman. "Demystifying Cloud Benchmarking". In: *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Apr. 2016, pp. 122–132 (cit. on p. 102).

[16k] *PeerStreamer: fast and efficient P2P streaming*. 2016. URL: `http://peerstreamer.org/` (cit. on pp. 20, 46).

[16l] *Proxmox: Server-Virtualization with KVM and Containers*. 2016. URL: `https://www.proxmox.com/` (cit. on p. 31).

[16m] *QFS-Quantcast File System*. 2016. URL: `https://quantcast.github.io/qfs/` (cit. on p. 18).

[16n] *QMP Monitoring page*. 2016. URL: `http://dsg.ac.upc.edu/qmpsu/index.php` (cit. on p. 77).

[16o] *QMP Network*. 2016. URL: `http://qmp.cat/` (cit. on pp. 77, 84).

[Sel+16] Mennan Selimi, Davide Vega, Felix Freitag, and Luís Veiga. "Towards Network-Aware Service Placement in Community Network Micro-Clouds". In: *22nd International Conference on Parallel and Distributed Computing (Euro-Par 2016)*. Springer International Publishing, 2016, pp. 376–388 (cit. on p. 90).

[Ser16] *Serf*. 2016. URL: `https://www.serfdom.io/` (cit. on p. 36).

[16p] *Sheepdog project*. 2016. URL: `https://sheepdog.github.io/sheepdog/` (cit. on p. 19).

[16q] *Sopcast: P2P Internet TV*. 2016. URL: http://www.sopcast.com/ (cit. on p. 45).

[16r] *Stirling Number of the Second Kind*. 2016. URL: http://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html (cit. on p. 93).

[Tär+16] William Tärneberg, Amardeep Mehta, Eddie Wadbro, Johan Tordsson, Johan Eker, Maria Kihl, and Erik Elmroth. "Dynamic application placement in the Mobile Cloud Network". In: *Future Generation Computer Systems* (2016), (cit. on p. 23).

[16s] *XtreemFS - Fault-Tolerant Distributed File System*. 2016. URL: http://www.xtreemfs.org/ (cit. on p. 19).

[al15] Davide Vega et al. "A technological overview of the guifi.net community network". In: *Computer Networks* 93, Part 2 (2015). Community Networks, pp. 260–278 (cit. on p. 94).

[15a] *Avahi Service Discovery Tool*. http://avahi.org/. 2015 (cit. on p. 35).

[Bai+15] Roger Baig, Ramon Roca, Felix Freitag, and Leandro Navarro. "guifi.net, a crowdsourced network infrastructure held in common". In: *Computer Networks* 90 (Oct. 2015), pp. 150–165 (cit. on p. 2).

[15b] *CaracalDB*. https://github.com/CaracalDB/CaracalDB. 2015 (cit. on p. 37).

[Dub+15] D. J. Dubois, G. Valetto, D. Lucia, and E. Di Nitto. "Mycocloud: Elasticity through Self-Organized Service Placement in Decentralized Clouds". In: *2015 IEEE 8th International Conference on Cloud Computing*. June 2015, pp. 629–636 (cit. on p. 23).

[15c] *Etcd key-value store*. https://github.com/coreos/etcd. 2015 (cit. on p. 37).

[15d] *FEDERICA: Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures*. 2015. URL: http://www.fp7-federica.eu/ (cit. on p. 38).

[15e] *LXC: Linux Containers*. 2015. URL: https://linuxcontainers.org/ (cit. on p. 40).

[NLN15] Axel Neumann, Ester López, and Leandro Navarro. “Evaluation of mesh routing protocols for wireless community networks”. In: *Computer Networks* 93, Part 2 (2015). Community Networks, pp. 308–323 (cit. on pp. 4, 14).

[Nov+15] P. Novotny, R. Urgaonkar, A. L. Wolf, and B. Ko. “Dynamic placement of composite software services in hybrid wireless networks”. In: *IEEE Military Communications Conference (MILCOM 2015)*. Oct. 2015, pp. 1052–1057 (cit. on p. 25).

[15f] *OpenVZ Linux Containers*. http://openvz.org/. 2015 (cit. on p. 34).

[15g] *OpenWRT*. https://openwrt.org/. 2015 (cit. on p. 38).

[15h] *PeerStreamer: P2P Media Streaming*. http://peerstreamer.org/. 2015 (cit. on p. 37).

[Sel+15a] M. Selimi, F. Freitag, R. P. Centelles, A. Moll, and L. Veiga. “TROBADOR: Service Discovery for Distributed Community Network Micro-Clouds”. In: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications (AINA 2015)*. (CORE Rank B). Mar. 2015, pp. 642–649 (cit. on p. 63).

[Sel+15b] Mennan Selimi et al. “Integration of an Assisted P2P Live Streaming Service in Community Network Clouds”. In: *Proceedings of the IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom 2015)*. Vancouver, Canada: IEEE, Nov. 2015 (cit. on pp. 3, 71, 89).

[Sel+15c] Mennan Selimi, Amin M Khan, Emmanouil Dimogerontakis, Felix Freitag, and Roger Pueyo Centelles. “Cloud services in the Guifi.net community network”. In: *Computer Networks* 93.P2 (Dec. 2015). Q2, pp. 373–388 (cit. on pp. 2, 18, 87).

[SBL15] B. Spinnewyn, B. Braem, and S. Latré. “Fault-tolerant application placement in heterogeneous cloud environments”. In: *Network and Service Management (CNSM)*. Nov. 2015, pp. 192–200 (cit. on p. 23).

[15i] *Sweep*. http://wiki.clommunity-project.eu/pilots:sweep. 2015 (cit. on p. 37).

[15j] *Syncthing*. http://syncthing.net/. 2015 (cit. on p. 37).

[15k] *TincVPN*. http://tinc-vpn.org/. 2015 (cit. on p. 35).

[Urg+15] Rahul Urgaonkar et al. "Dynamic service migration and workload scheduling in edge-clouds". In: *Performance Evaluation* 91 (2015). Special Issue: Performance 2015, pp. 205–228 (cit. on pp. 25, 111).

[Wan+15a] Shiqiang Wang et al. "Dynamic service placement for mobile micro-clouds with predicted future costs". In: *IEEE International Conference on Communications (ICC)*. June 2015, pp. 5504–5510 (cit. on p. 25).

[Wan+15b] Shiqiang Wang, Rahul Urgaonkar, Murtaza Zafer, Ting He, Kevin Chan, and Kin K. Leung. "Dynamic Service Migration in Mobile Edge-Clouds". In: *CoRR* abs/1506.05261 (2015) (cit. on p. 25).

[15l] *Zero Configuration Networking (Zeroconf)*. http://www.zeroconf.org/. 2015 (cit. on p. 57).

[BML14a] L. Baldesi, L. Maccari, and R. Lo Cigno. "Live P2P streaming in CommunityLab: Experience and insights". In: *13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*. June 2014 (cit. on p. 20).

[BML14b] Luca Baldesi, Leonardo Maccari, and Renato Lo Cigno. "Improving P2P Streaming in Community-Lab Through Local Strategies". In: *10th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. Larnaca, Cyprus, Oct. 2014, pp. 33–39 (cit. on p. 20).

[Cab14] Guillem Cabrera Añon. "Service allocation methodologies for contributory computing environments". PhD thesis. Barcelona, Spain: Universitat Oberta de Catalunya. Escola de Doctorat, 2014 (cit. on p. 24).

[Cat+14] Simon Caton, Christian Haas, Kyle Chard, Kris Bubendorfer, and Omer F. Rana. "A Social Compute Cloud: Allocating and Sharing Infrastructure Resources via Social Networks". In: *IEEE Transactions on Services Computing* 7.3 (July 2014), pp. 359–372 (cit. on p. 17).

[14a] *Community-Lab: Community Networks Testbed by the CONFINE Project*. http://community-lab.net/. 2014 (cit. on p. 37).

[14b] *CONFINE Project: Community Networks Testbed for the Future Internet*. 2014 (cit. on p. 20).

[Dit+14] Andreas Dittrich, Björn Lichtblau, Rafael Rezende, and Miroslaw Malek. “Modeling Responsiveness of Decentralized Service Discovery in Wireless Mesh Networks”. English. In: *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. Ed. by Kai Fischbach and UdoR. Krieger. Vol. 8376. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 88–102 (cit. on p. 21).

[Gha+14] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna. “Replica Placement in Cloud through Simple Stochastic Model Predictive Control”. In: *2014 IEEE 7th International Conference on Cloud Computing*. June 2014, pp. 80–87 (cit. on p. 22).

[Jan+14] Minsung Jang, Karsten Schwan, Ketan Bhardwaj, Ada Gavrilovska, and Adhyas Avasthi. “Personal clouds: Sharing and integrating networked resources to enhance end user experiences”. In: *33rd Annual IEEE International Conference on Computer Communications (INFOCOM'14)*. Toronto, Canada: IEEE, Apr. 2014, pp. 2220–2228 (cit. on p. 18).

[Moe+14] H. Moens, B. Hanssens, B. Dhoedt, and F. De Turck. “Hierarchical network-aware placement of service oriented applications in Clouds”. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. May 2014, pp. 1–8 (cit. on p. 23).

[Ryd+14] M. Ryden et al. “Nebula: Distributed Edge Cloud for Data Intensive Computing”. In: *IEEE International Conference on Cloud Engineering (IC2E), 2014*. Mar. 2014, pp. 57–66 (cit. on p. 71).

[SF14] M. Selimi and F. Freitag. “Tahoe-LAFS Distributed Storage Service in Community Network Clouds”. In: *4th IEEE International Conference on Big Data and Cloud Computing (BDCloud'14)*. Dec. 2014, pp. 17–24 (cit. on p. 37).

[Sel+14] M. Selimi, F. Freitag, R. Pueyo Centelles, and A. Moll. “Distributed Storage and Service Discovery for Heterogeneous Community Network Clouds”. In: *7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'14)*. Dec. 2014, pp. 204–212 (cit. on p. 32).

[Veg+14] D. Vega, R. Meseguer, G. Cabrera, and J.M. Marques. “Exploring local service allocation in Community Networks”. In: *10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'14), IEEE*. Oct. 2014, pp. 273–280 (cit. on p. 25).

[ZLL14] Han Zhao, Xinxin Liu, and Xiaolin Li. “Towards efficient and fair resource trading in community-based cloud computing”. In: *Journal of Parallel and Distributed Computing* 74.11 (Aug. 2014), pp. 3087–3097 (cit. on p. 18).

[Ala13] Amr Alasaad. “Content sharing and distribution in wireless community networks”. English. In: Univeristy of British Columbia, PhD Thesis, 2013 (cit. on p. 20).

[Bra+13] Bart Braem et al. “A case for research with and on community networks”. In: *ACM SIGCOMM Computer Communication Review* 43.3 (July 2013), pp. 68–73 (cit. on p. 38).

[CNE13a] Llorenç Cerdà-Alabern, Axel Neumann, and Pau Escrich. “Experimental Evaluation of a Wireless Community Mesh Network”. In: *Proceedings of the 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. MSWiM '13. Barcelona, Spain: ACM, 2013, pp. 23–30 (cit. on pp. 4, 49, 79, 84).

[CNE13b] Llorenç Cerdà-Alabern, Axel Neumann, and Pau Escrich. “Experimental Evaluation of a Wireless Community Mesh Network”. In: *Proceedings of the 16^{th} ACM International Conference on Modeling, Analysis, Simulation of Wireless and Mobile Systems*. MSWiM '13. Barcelona, Spain: ACM, 2013, pp. 23–30 (cit. on pp. 42, 61).

[Che+13] Y. F. Chen, S. Daniels, M. Hadjieleftheriou, P. Liu, C. Tian, and V. Vaishampayan. “Distributed storage evaluation on a three-wide inter-data center deployment”. In: *2013 IEEE International Conference on Big Data*. Oct. 2013, pp. 17–22 (cit. on p. 18).

[Esp+13] Christian Esposito, Massimo Ficco, Francesco Palmieri, and Aniello Castiglione. “Interconnecting Federated Clouds by Using Publish-Subscribe Service”. In: *Cluster Computing* 16.4 (Apr. 2013), pp. 887–903 (cit. on p. 17).

[GSF13] Mark Gall, Angelika Schneider, and Niels Fallenbeck. “An Architecture for Community Clouds Using Concepts of the Intercloud”. In: *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, Mar. 2013, pp. 74–81 (cit. on p. 17).

[Gra+13] R. Gracia-Tinedo, M. S. Artigas, A. Moreno-Martínez, C. Cotes, and P. G. López. “Actively Measuring Personal Cloud Storage”. In: *2013 IEEE Sixth International Conference on Cloud Computing*. June 2013, pp. 301–308 (cit. on p. 18).

[KBF13] Amin M Khan, Ümit Cavus Büyükşahin, and Felix Freitag. “Towards Incentive-Based Resource Assignment and Regulation in Clouds for Community Networks”. In: *Economics of Grids, Clouds, Systems, and Services*. Vol. 8193. Lecture Notes in Computer Science. Zaragoza, Spain, Sept. 2013, pp. 197–211 (cit. on p. 11).

[LaC13] Katrina et al. LaCurts. “Choreo: Network-aware Task Placement for Cloud Applications”. In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC ’13. Barcelona, Spain: ACM, 2013, pp. 191–204 (cit. on pp. 5, 22).

[Oli+13] João Oliveira et al. “Can Peer-to-Peer live streaming systems coexist with free riders?” In: *P2P’13*. 2013, pp. 1–5 (cit. on p. 20).

[Pun+13] Magdalena Punceva, Ivan Rodero, Manish Parashar, Omer F Rana, and Ioan Petri. “Incentivising resource sharing in social clouds”. In: *Concurrency and Computation: Practice and Experience* (Mar. 2013) (cit. on p. 17).

[RC13] A. Russo and R.L. Cigno. “Pullcast: Peer-assisted video multicasting for wireless mesh networks”. In: *10^{th} Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. Mar. 2013 (cit. on p. 20).

[AL12] M. Alicherry and T.V. Lakshman. “Network aware resource allocation in distributed clouds”. In: *Proceedings of INFOCOM, IEEE*. Mar. 2012, pp. 963–971 (cit. on p. 23).

[BMT12] Ozalp Babaoglu, Moreno Marzolla, and Michele Tamburini. “Design and implementation of a P2P Cloud system”. In: *27th Annual ACM Symposium on Applied Computing (SAC ’12)*. New York, USA, Mar. 2012, pp. 412–417 (cit. on p. 17).

[Cha+12] Kyle Chard, Kris Bubendorfer, Simon Caton, and Omer F. Rana. “Social Cloud Computing: A Vision for Socially Motivated Resource Sharing”. In: *IEEE Transactions on Services Computing* 5.4 (Jan. 2012), pp. 551–563 (cit. on p. 17).

[DP12] Salvatore Distefano and Antonio Puliafito. “Cloud@Home: Toward a Volunteer Cloud”. In: *IT Professional* 14.1 (Jan. 2012), pp. 27–31 (cit. on p. 17).

[KIH12] Adrian Klein, Fuyuki Ishikawa, and Shinichi Honiden. “Towards Network-aware Service Composition in the Cloud”. In: *Proceedings of the 21st International Conference on World Wide Web*. WWW ’12. Lyon, France: ACM, 2012, pp. 959–968 (cit. on p. 23).

[NLN12] A. Neumann, E. Lopez, and L. Navarro. “An evaluation of BMX6 for community wireless networks”. In: *8th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 I*. Oct. 2012, pp. 651–658 (cit. on p. 85).

[SD12] K. Shima and N. Dang. *Indexes for Distributed File/Storage Systems as a Large Scale Virtual Machine Disk Image Storage in a Wide Area Network*. 2012 (cit. on p. 19).

[SG12] Moritz Steiner and Bob et al. Gaglianello. “Network-aware Service Placement in a Distributed Cloud Environment”. In: *Proceedings of the ACM SIGCOMM 2012 Conference*. SIGCOMM ’12. Helsinki, Finland: ACM, 2012, pp. 73–74 (cit. on p. 23).

[Tra+12] S. Traverso et al. “Experimental comparison of neighborhood filtering strategies in unstructured P2P-TV systems”. In: *IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*. Sept. 2012, pp. 13–24 (cit. on p. 20).

[Tse+12] F. H. Tseng, C. Y. Chen, L. D. Chou, and H. C. Chao. “Implement a reliable and secure cloud distributed file system”. In: *2012 International Symposium on Intelligent Signal Processing and Communications Systems*. Nov. 2012, pp. 227–232 (cit. on p. 19).

[Veg+12] Davide Vega, Llorenc Cerda-Alabern, Leandro Navarro, and Roc Meseguer. “Topology patterns of a community network: Guifi.net”. In: *1st International Workshop on Community Networks and Bottom-up-Broadband (CNBuB 2012), within IEEE WiMob*. Barcelona, Spain, Oct. 2012, pp. 612–619 (cit. on p. 60).

[Wir+12] Hanno Wirtz, Tobias Heer, Martin Serror, and Klaus Wehrle. “DHT-based localized service discovery in wireless mesh networks.” In: *MASS*. 2012, pp. 19–28 (cit. on p. 21).

[B+11] R. Birke, E. Leonardi, M. Mellia, et al. “Architecture of a network-aware P2P-TV application: the NAPA-WINE approach”. In: *Communications Magazine, IEEE* 49.6 (June 2011), pp. 154–163 (cit. on pp. 46, 51).

[Cou+11] A.P. Couto da Silva, E. Leonardi, M. Mellia, and M MEO. “Exploiting Heterogeneity in P2P Video Streaming”. In: *IEEE Transactions on Computers* 60 (May 2011), 667–679 (cit. on p. 20).

[MG11] Peter Mell and Timothy Grance. “The NIST Definition of Cloud Computing”. In: *NIST Special Publication* 800.145 (2011) (cit. on pp. 16, 32).

[VP11] H. Verespej and J. Pasquale. “A Characterization of Node Uptime Distributions in the PlanetLab Test Bed”. In: *2011 IEEE 30th International Symposium on Reliable Distributed Systems*. Oct. 2011, pp. 203–208 (cit. on p. 86).

[Yi+11] Sangho Yi, Emmanuel Jeannot, Derrick Kondo, and David P. Anderson. “Towards Real-Time, Volunteer Distributed Computing”. In: *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011)*. Newport Beach, USA: IEEE, May 2011, pp. 154–163 (cit. on p. 17).

[Aga+10] Sharad Agarwal et al. “Volley: Automated Data Placement for Geo-distributed Cloud Services”. In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. San Jose, California: USENIX Association, 2010, pp. 2–2 (cit. on p. 22).

[BD10] Gautier Berthou and Jim Dowling. “P2P VoD Using the Self-organizing Gradient Overlay Network”. In: *Proceedings of the 2nd International Workshop on Self-organizing Architectures*. SOAR '10. Washington, DC, USA: ACM, 2010, pp. 29–34 (cit. on p. 37).

[BRC10] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. “InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services”. In: *Algorithms and Architectures for Parallel Processing*. Lecture Notes in Computer Science 6081 (Mar. 2010), pp. 20–31 (cit. on p. 17).

[DS10] A Dittrich and Felix Salfner. “Experimental responsiveness evaluation of decentralized service discovery”. In: *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. Apr. 2010, pp. 1–7 (cit. on p. 21).

[Her10] Klaus Herrmann. “Self-organized Service Placement in Ambient Intelligence Environments”. In: *ACM Trans. Auton. Adapt. Syst.* 5.2 (May 2010), 6:1–6:39 (cit. on p. 22).

[RG10] Aditya Rajgarhia and Ashish Gehani. “Performance and Extension of User Space File Systems”. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. SAC '10. Sierre, Switzerland: ACM, 2010, pp. 206–213 (cit. on p. 42).

[10] “Towards Transactional Load over XtreemFS”. In: *CoRR* abs/1001.2931 (2010). Withdrawn. (cit. on p. 19).

[Wit10] Georg Wittenburg. “Service Placement in Ad Hoc Networks”. PhD thesis. Berlin, Germany: Department of Mathematics and Computer Science, Freie Universität Berlin, 2010 (cit. on p. 24).

[Beb+09] Adam L. Beberg, Daniel L. Ensign, Guha Jayachandran, Siraj Khaliq, and Vijay S. Pande. “Folding@home: Lessons from eight years of volunteer distributed computing”. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. Rome, Italy, May 2009 (cit. on p. 17).

[Cap+09] Justin Cappos, Ivan Beschastnikh, Arvind Krishnamurthy, and Tom Anderson. “Seattle: a platform for educational cloud computing”. In: *40th ACM Technical Symposium on Computer Science Education (SIGCSE ’09)*. Chattanooga, USA, Mar. 2009, pp. 111–115 (cit. on p. 17).

[Eli+09] Fotios A. Elianos, Georgia Plakia, Pantelis A. Frangoudis, and George C. Polyzos. “Structure and evolution of a large-scale Wireless Community Network”. In: *2009 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks & Workshops*. IEEE, June 2009 (cit. on p. 31).

[LF09] Andrea Lancichinetti and Santo Fortunato. “Community detection algorithms: A comparative analysis”. In: *Phys. Rev. E* 80 (5 Nov. 2009), p. 056117 (cit. on p. 94).

[MB09] Alexandros Marinos and Gerard Briscoe. “Community Cloud Computing”. In: *Cloud Computing*. Lecture Notes in Computer Science 5931 (2009), pp. 472–484 (cit. on p. 17).

[WW08] Zooko Wilcox-O’Hearn and Brian Warner. “Tahoe: The Least-authority Filesystem”. In: *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*. StorageSS ’08. Alexandria, Virginia, USA: ACM, 2008, pp. 21–26 (cit. on pp. 18, 36, 39, 40).

[DMQ07] C. Dabrowski, K. Mills, and S. Quirolgico. “Understanding failure response in service discovery systems”. In: *Journal of Systems and Software* 80.6 (2007), pp. 896–917 (cit. on p. 21).

[Lee+07] Jae Woo Lee et al. “z2z: Discovering Zeroconf Services Beyond Local Link”. In: *Globecom Workshops, 2007 IEEE*. Nov. 2007, pp. 1–7 (cit. on p. 21).

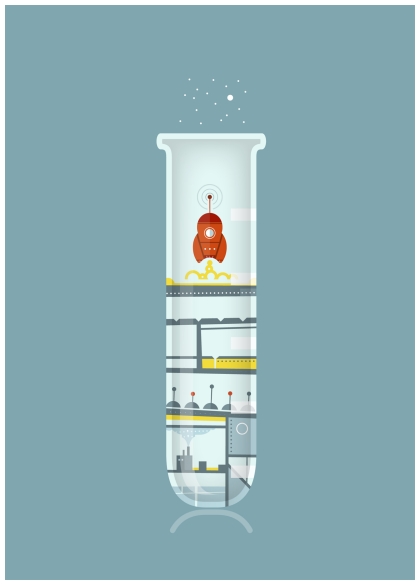
[CG06] Celeste Campo and Carlos García-Rubio. “DNS-Based Service Discovery in Ad Hoc Networks: Evaluation and Improvements”. English. In: *Personal Wireless Communications*. Ed. by Pedro Cuenca and Luiz Orozco-Barbosa. Vol. 4217. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 111–122 (cit. on p. 21).

[And04] David P Anderson. “BOINC : A System for Public-Resource Computing and Storage”. In: *Fifth IEEE/ACM International Workshop on Grid Computing*. Pittsburgh, USA, Nov. 2004, pp. 4–10 (cit. on p. 16).

[Oh+04] Chang-Seok Oh et al. “A Hybrid Service Discovery for Improving Robustness in Mobile Ad Hoc Networks”. In: *IEEE International Conference on Dependable Systems and Networks (DSN 2004)*. 2004 (cit. on p. 21).

[Chu+03] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. “PlanetLab: An Overlay Testbed for Broad-Coverage Services”. In: *ACM SIGCOMM Computer Communication Review* 33.3 (July 2003), pp. 3–12 (cit. on p. 17).

[03] *Optimized Link State Routing Protocol (OLSR), RFC 3626*. 2003. URL: `https://tools.ietf.org/html/rfc3626` (cit. on p. 4).



This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Minion Pro, designed by Robert Slimbach in 1990 inspired by late Renaissance-era type and issued by Adobe in 2000. The headlines and captions are set in variations of Myriad Pro, a humanist sans-serif typeface designed by Robert Slimbach and Carol Twombly in 1990 and issued by Adobe in 2000. The above illustration was created by Ben Schlitter and released under CC BY-NC-ND 4.0. A template that can be used to format a PhD dissertation with this look & feel has been released under the permissive AGPL license, and can be found online at github.com/aminmkhan/Dissertate or from its lead author, Jordan Suchow, at suchow@post.harvard.edu.