UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

GOSSIP-BASED DATA DISTRIBUTION IN MOBILE AD HOC NETWORKS

Hugo Alexandre Tavares Miranda

DOUTORAMENTO EM INFORMÁTICA

(Engenharia Informática)

2007

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

GOSSIP-BASED DATA DISTRIBUTION IN MOBILE AD HOC NETWORKS

Hugo Alexandre Tavares Miranda

DOUTORAMENTO EM INFORMÁTICA

(Engenharia Informática)

Tese orientada pelo Prof. Doutor Luís Eduardo Teixeira Rodrigues

2007

This work was partially supported by the European Science Foundation (ESF) programme Middleware for Network Eccentric and Mobile Applications (MiNEMA), by project Probabilistically-Structured Overlay Networks (P-SON), POSC/EIA/60941/2004 and by project Middleware for Context-aware and Adaptive Systems (MICAS), POSC/EIA/60692/2004 through Fundação para a Ciência e Tecnologia (FCT) and FEDER and by PRODEP

Resumo Extendido

A tecnologia utilizada nas redes sem fios sofreu avanços significativos, que criaram as condições para o desenvolvimento de uma nova geração de dispositivos computacionais móveis, como os telefones celulares, os computadores portáteis e os assistentes digitais pessoais. Por sua vez, a popularidade crescente destes dispositivos aumentou o interesse nas aplicações de computação móvel que beneficiam da ubiquidade das redes de computadores.

Os dispositivos computacionais portáteis são tendencialmente de pequena dimensão e leves, abdicando de algumas funcionalidades. Em comparação com, por exemplo, os computadores de secretária, os dispositivos móveis apresentam menor memória e poder computacional. Adicionalmente, espera-se que nestes dispositivos ocorram frequentemente interrupções temporárias do acesso à rede, devido por exemplo à exaustão das baterias, ao deslocamento do dispositivo para uma zona onde não existe cobertura ou, simplesmente, porque o utilizador o solicitou.

As redes ad hoc móveis (MANETs, do inglês Mobile Ad hoc NETworks) são uma classe particular de redes sem fios caracterizadas pela ausência de infra-estrutura. Nas MANETs, a rede é exclusivamente composta pelos dispositivos participantes. Espera-se que estas redes venham a ser particularmente úteis para situações onde a instalação da infra-estrutura é excessivamente cara, demorada ou inadequada, por exemplo em locais afectados por catástrofes naturais, missões de busca e salvamento ou missões militares.

O desenvolvimento de aplicações para MANETs tem desafios muito particulares.

Ao contrário do que é comum nas redes fixas e nas redes sem fios infra-estruturadas, não é possível concentrar serviços como o armazenamento de dados, serviço de nomes, etc. em servidores especializados e com um elevado grau de fiabilidade. Pelo contrário, em MANETs estes serviços terão que ser conseguidos através da cooperação dos participantes. Um exemplo paradigmático é o encaminhamento de mensagens entre dispositivos e que é suportado por uma cadeia de dispositivos localizados entre ambos.

Nos últimos anos foram propostas diversas soluções para o problema do encaminhamento de mensagens. Com este problema resolvido, é agora o momento de abordar e resolver outras limitações ao desenvolvimento de aplicações para MANETs. Idealmente, as soluções devem ser desenvolvidas sob a forma de camadas de código intermédio, capazes de esconder das aplicações as dificuldades que estas redes acarretam.

A disseminação de dados é um dos serviços chave em MANETs. Este serviço permite por exemplo a equipas de busca e salvamento partilhar fotografias ou informação actualizada da zona afectada; a passageiros num aeroporto encontrar um parceiro para um jogo de xadrez ou a vendedores numa feira usar uma MANET para anunciar os seus produtos. Todos estes exemplos partilham um modelo comum de dados: os dados são produzidos por diferentes participantes e não são actualizados com frequência. Além disso, não é possível inferir um destinatário para os dados nem quais os dispositivos que estarão mais interessados.

Em MANETs, a concentração de dados num único dispositivo deve ser evitada devido: i) à baixa disponibilidade dos dispositivos; ii) à sobrecarga dos dispositivos que armazenariam os dados e iii) à mobilidade dos dispositivos, que resulta em interrupções frequentes da ligação à rede e na ocorrência de partições. A disponibilidade dos dados aumentaria se estes fossem replicados pelos participantes. Por outro lado, um algoritmo de distribuição das réplicas tem que evitar a redundância excessiva, uma vez que os dispositivos podem ter recursos limitados. Atendendo a que para os dispositivos que operam nas redes sem fios, quer a largura de banda quer a energia

são recursos escassos, o algoritmo deve também reduzir tanto quanto possível a quantidade de dados e informação de controlo necessária para proceder à disseminação.

Nas MANETs, as réplicas devem ser distribuídas de forma uniforme pelos dispositivos que formam a rede, evitando a sua agregação em regiões geográficas. Ou seja, quando um item de dados é solicitado por um dispositivo, a distância ao dispositivo que fornece a informação deve ser aproximadamente a mesma, independentemente da localização de ambos. Naturalmente, a distância real dependerá de múltiplos parâmetros, como o número de dispositivos, a memória disponibilizada por cada um e o número de itens anunciados.

A distribuição geográfica das réplicas pode oferecer uma contribuição valiosa para o desempenho das aplicações. A transmissão e recepção de dados contribuem significativamente para o consumo de energia dos dispositivos. Se as réplicas se encontrarem distribuídas geograficamente, um dispositivo que pretenda obter um qualquer item de dados encontra-lo-á com grande probabilidade num qualquer dispositivo próximo, permitindo a sua obtenção com um número menor de transmissões. Contribui-se desta forma para a redução do consumo de energia, da latência na obtenção de informação e da quantidade de tráfego gerado.

A disseminação de dados em MANETs, e em particular, a sua distribuição geográfica, são problemas que já foram abordados por diversos grupos de investigação. Contudo, a maioria do trabalho anterior assume que os dispositivos conhecem a sua localização geográfica. De notar que esta informação pode não estar disponível, como por exemplo no interior de edifícios, mesmo assumindo que os dispositivos móveis dispõem de equipamentos de localização geográfica (por exemplo receptores do sistema de posicionamento global, GPS). Este requisito diminui por isso consideravelmente o domínio de aplicação destes algoritmos.

A tese apresenta um conjunto de algoritmos que se complementam para oferecer um serviço integral de replicação dos dados por um subconjunto, geograficamente disperso, dos participantes na rede. Assumimos contudo que os dispositivos não são capazes de obter a sua localização geográfica ou que a informação disponível não apre-

senta precisão suficiente. A distribuição geográfica permite que os dispositivos obtenham a informação utilizando um número limitado de transmissões, contribuindo desta forma para a preservação dos recursos da rede.

As principais contribuições da tese são:

- um novo método para a selecção dos dispositivos mais adequados para proceder à retransmissão de mensagens que se pretendem difundidas por todos os participantes. O método utiliza a força do sinal recebido para seleccionar os dispositivos mais distantes dos emissores anteriores;
- Um algoritmo de replicação de dados para MANETs. O algoritmo dispõe as réplicas por forma a que qualquer dispositivo possa encontrar uma réplica dos dados dentro de um número predefinido de saltos;
- Quatro algoritmos para atenuar o impacto do movimento dos dispositivos na distribuição geográfica. Estes algoritmos apresentam um consumo diminuto de recursos, beneficiando do tráfego gerado pelas operações de obtenção de dados para avaliar a qualidade da disseminação e aplicar as correcções;
- Um algoritmo de agregação de dados a partir de uma condição especificada pelo utilizador e que deve ser satisfeita pelos dados a obter.

A tese começa por motivar o problema e explanar os requisitos do modelo no Capítulo 1. O Capítulo 2 apresenta o trabalho relacionado, dividido em dois blocos principais: algoritmos de disseminação de mensagens e algoritmos de distribuição de dados. As contribuições da tese para cada um destes aspectos são apresentadas e avaliadas respectivamente nos Capítulos 3 e 4. Este último apresenta e avalia ainda os algoritmos de atenuação do movimento dos dispositivos. O Capítulo 5 expõe uma aplicação dos algoritmos a uma versão do Session Initiation Protocol (SIP) especialmente desenvolvida para MANETs. As principais conclusões deste trabalho assim como as perspectivas para o trabalho futuro são apresentadas no Capítulo 6.

Abstract

Wireless networks are useful in many different scenarios. They allow to create emergency networks for catastrophe response, wide area surveillance networks in hostile environments, or simply permit users to share information, play on-line games, and surf the Web. Mobile ad hoc networks are a particular case of wireless networks characterised by the absence of a supporting infrastructure.

The thesis addresses the problem of building middleware services that permit to fully exploit the opportunities offered by mobile ad hoc networks. For that purpose, it is required to design algorithms that account for the limitations of mobile devices and that make a careful use of the scarce resources available in ad hoc networks. A central middleware service for mobile applications is data sharing. The thesis addresses the use of data replication as a technique to improve data availability and resource savings in mobile ad hoc networks. In particular, the thesis proposes the use of epidemic protocols to achieve these goals.

In this context, the thesis presents the following contributions. It presents and evaluates $i)$ an algorithm to reduce the number of transmissions required in a broadcast, $ii)$ an algorithms for the geographical distribution of replicas of data items, and $iii)$ algorithms to attenuate the impact of node movement in the geographical distribution. Finally, the thesis describes an application of the algorithms to build a concrete application, a version of the Session Initiation Protocol for wireless networks.

Palavras Chave

Redes ad hoc móveis,
Distribuição geográfica de dados,
Algoritmos de disseminação

Keywords

Mobile ad hoc networks,
Geographical data distribution,
Broadcast algorithms

Agradecimentos

Na realização deste trabalho contei com o apoio pessoal e/ou científico de um conjunto de pessoas e instituições sem as quais este trabalho não teria sido possível. Aproveito por isso para expressar aqui os meus agradecimentos àqueles que mais significativamente deram a sua contribuição.

Ao Gonçalo e ao Pedro agradeço a paciência com que aceitam as minhas ausências. À Cristina devo uma parte significativa destes resultados. Pela motivação que sempre me transmitiu, pela disponibilidade com que vai assumindo muitas das minhas tarefas, abdicando do seu tempo. Sem a sua energia e apoio não teria sido possível a realização deste trabalho.

Não teria sido possível chegar até aqui se não tivesse, ao longo dos tempos, sentido um apoio continuado no prosseguimento dos meus estudos. Agradeço por isso aos meus pais e irmãs a quem dedico genericamente esta tese.

Foram muitas as instituições que contribuiram para este trabalho. O programa MiNEMA, da European Science Foundation (ESF) suportou financeiramente os contactos com a Universidade de Helsínquia. Adicionalmente, tem servido como um fórum que me permitiu enriquecer a minha formação, criar e fortalecer laços com outros investigadores.

Localmente, contei com o apoio institucional do Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa e do Laboratório de Sistemas Informáticos de Grande Escala (LaSIGE). A ambos agradeço a disponibilização das condições para desenvolver o meu trabalho. Uma palavra especial para os membros

(passados e presentes) do DIALNP, que têm feito deste grupo um excelente ponto de partilha de ideias, entreajuda e convívio, contribuindo assim para um ambiente de estímulo à investigação. Um agradecimento especial ao Filipe Araújo pelo cuidado que pôs na revisão de parte desta tese.

O Professor Luís Rodrigues excedeu largamente o papel de orientador científico da tese. A sua experiência, permanente disponibilidade, dinâmica invejável e o empenho no acompanhamento dos trabalhos dos alunos e em proporcionar-lhes todas as condições necessárias, são por si só um dos mais importantes factores de motivação e fazem dele um modelo a seguir. Por fim, agradeço-lhe a oportunidade que me deu de alargar a minha formação para domínios não estritamente relacionados com a elaboração da tese. Estou certo que experiências como a organização de eventos, a participação na elaboração de propostas de financiamento e em comissões de programa, e o estabelecimento de contactos com outros investigadores serão também factores importantes para o desenvolvimento da minha carreira.

Lisboa, Junho de 2007

Hugo Alexandre Miranda

À Cristina

Ao Gonçalo e ao Pedro

Contents

List of Figures

List of Tables

1

Introduction

Advances in wireless network technology paved the way to the development of a new generation of communication devices, like cell phones, wireless enabled laptops and PDAs. The growing popularity of these devices has increased the demand for mobile computing applications that can benefit from ubiquitous connectivity.

A common requirement for wireless devices is that they should be portable, so that they can be easily carried by the users. Therefore, mobile devices tend to be small and lightweight, trading of some functionalities by size. In comparison with, for example, desktop computers, mobile devices present lower computing power and less memory. Mobile devices are expected to be temporarily disconnected, due to drained batteries, movement of the device to some location outside the network range or simply because the user has requested it.

Mobile Ad hoc NETworks (MANETs) are one particular class of wireless networks characterised by the lack of a support infrastructure. The network is composed only by the mobile devices. MANETs are particularly useful for situations where the deployment of an infrastructure is expensive or impossible like catastrophe scenarios, search and rescue, and military operations.

Application development is particularly challenging in MANETs. Contrary to what is typical in wired or wireless infrastructured networks, it is not possible to centralise in some reliable and powerful server functions such as data storage, naming service, etc. Instead, these services must be provided via the cooperation of the participating devices. A notorious example is packet routing between two devices not in

transmission range, which must be supported by a sequence of intermediate devices that forward the message.

Packet routing in ad hoc networks has received considerable attention in the last few years and a number of interesting protocols have been proposed. With the routing problem solved, it is time to identify and resolve other impairments to the development of distributed applications in MANETs. Ideally, solutions should be provided as middleware services, capable of masking the peculiarities of ad hoc networks to the application.

Data dissemination is one of the key applications of MANETs. Teams of a search-and-rescue operation may want to exchange photographs or up-to-date information of the disaster scene, participants on a digital flea market may use a MANET to advertise their products, and passengers waiting on an airport may try to find a partner for a chess game. All these scenarios share a common data model: data is produced by different nodes and is not frequently updated. Furthermore, there is not an *a priori* known target node for consuming the data, nor it is possible to infer which nodes are more likely to be interested on it.

The concentration of the data in a single node is unsuitable in MANETs due to $i)$ the low availability of the devices; $ii)$ the overload of the device storing the data; and $iii)$ the mobility of the devices, which results in frequent network disconnections and network partitions. Data availability may be improved if multiple replicas are distributed through the participants. However, a data dissemination algorithm for MANETs should balance the need to provide data replication with the need to avoid excessive data redundancy (as nodes may have limited storage capability). Since in wireless networks both bandwidth and battery power are precious resources, the algorithm should also minimise the amount of data and control packets required to disseminate data.

In MANETs, data replicas should be deployed as evenly as possible among all the nodes that form the network, avoiding clustering of information in sub-areas. That is, whenever a data item is requested by a node S, the distance to the node that provides

the reply should be approximately the same, regardless of the location of S. Naturally, the actual distance depends on multiple parameters, such as the number of nodes in the network, the amount of memory made available at each node, and the total number of data items stored in the network.

The geographical distribution of the replicas can provide a valuable contribution to the performance of the data dissemination applications. Packet transmission and reception have been identified as some of the most resource demanding operations for mobile devices (Feeney & Nilsson, 2001). If replicas are geographically distributed, nodes are more likely to contact at least one of them using a small number of messages, saving their battery reserves. Because replicas can be found close to any node, the query time is reduced, what contributes to improve latency and reduce the traffic.

Due to its importance, data dissemination in MANETs has been a subject widely studied. In particular, the development of data dissemination algorithms that aim at a geographical distribution of items is not novel. However, most of previous work assumes that nodes are aware of their geographical location. That is, these algorithms can only be applied if mobile nodes have some location device available or perform complex computations to determine their position. In addition, we note that at some sites (for example indoors), location information may not be available or it may not be possible to obtain it with sufficient accuracy. Therefore, the application domain of the existing geographical data distribution algorithms is severely limited.

1.1 Problem Statement and Objectives

The goal of this thesis is to devise algorithms that distribute replicas of data items by some of the mobile nodes in an ad hoc network. The thesis assumes that nodes are unable to retrieve their geographical location. Still, replicas should be geographically distributed so that any node in the network can retrieve the item from a location at most a few hops away.

The thesis assumes that all nodes cooperate to the successful execution of the algorithms. In particular, nodes are required to $i)$ make some storage space available to be managed by the algorithms; and $ii)$ send all the messages that are required, for example, by contributing to the propagation of the messages transmitted by other nodes and by replying to queries.

An ideal data distribution algorithm should provide consistent updates of the replicas, high availability and partition tolerance. Unfortunately, Gilbert & Lynch (2002) have proven the Brewer's conjecture, which stated that at most two of these properties can be achieved simultaneously. In this thesis, replicas are used to improve availability and tolerance to partitions. The lack of consistency support does not necessarily limit the utility of the algorithms: this section has already presented some examples of applications where data is not frequently updated and which do not depend of strict consistency requirements for their correct execution.

1.2 Contributions

The main contributions of this thesis are:

- A novel method for the selection of the most adequate nodes for retransmitting a message in a broadcast. The method uses the Received Signal Strength Indicator (RSSI) to select nodes that are more distant from the source of the previous retransmissions. The selection criteria improves previous work by requiring a lower number of retransmissions for delivering the message to a comparable number of nodes.

- A replication algorithm for mobile ad hoc networks. The algorithm places copies at a bounded number of hops of any node in the network so that data can be retrieved with a limited number of messages. In addition, the algorithm presents a moderate use of the memory of the devices. The algorithm performs these tasks without using location information.

- Four alternative shuffling algorithms to mitigate the impact of node movement in the geographical distribution. The algorithms relocate replicas in the background and use the feedback provided by messages used for other operations. The algorithms differentiate by the effort to preserve the number of replicas and by the type of messages they use.

- A data gathering mechanism to retrieve multiple items satisfying some condition that benefits of the geographical distribution of the replicas to minimise the number of transmissions.

1.3 Results

The thesis presents the following results:

- A middleware framework for replica distribution in MANETs. The framework provides services for geographical distribution of the replicas, data retrieval and shuffling.

- Extensive simulation results that confirm the properties claimed by the framework.

- A proof-of-concept application to a framework that adapts the Session Initiation Protocol (SIP) to mobile ad hoc networks.

1.4 Outline of the Thesis

In this thesis we present the components of a scalable middleware framework for data distribution and retrieval in ad hoc networks. The framework replicates data items so that nodes have one replica located in a bounded and predefined number of hops. A general overview of the framework is presented in Figure 1.1. The framework

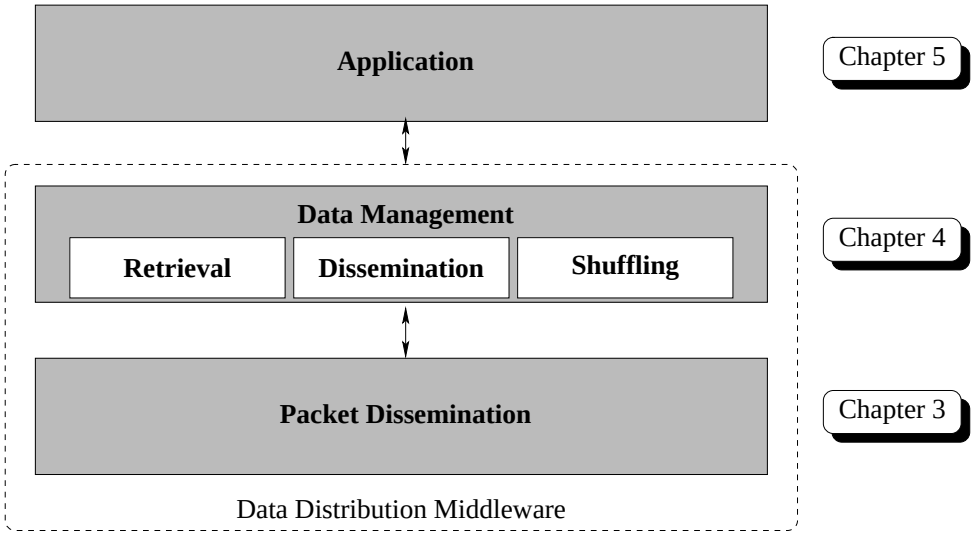


Figure 1.1: Architecture of data distribution middleware frameworks

is composed by a Data Management and a Packet Dissemination module. The chapter where each of the components is described is presented on the right.

The thesis is structured as follows. Chapter 2 presents a survey of the related work. Our work combines a novel broadcast algorithm with data management algorithms. Previously, these aspects have been addressed has two independent lines of research. The chapter reflects this separation by presenting two major sections, each covering some of the most significant contributions to the state of the art on these research trends. Each section is concluded with a comparison of the algorithms surveyed and an analysis focused on their limitations, which is used to motivate our work.

The broadcast algorithm is the focus of Chapter 3. The chapter begins with the description of the algorithm, named“Pampa”, and then proceeds to compare it both analytically and by the results of simulations with some of the related work.

Chapter 4 addresses the data management module. The chapter starts with a description of the dissemination, query and shuffling algorithms. The quality of the dissemination and the capability of the shuffling algorithms to keep an acceptable data distribution in spite of node movement are evaluated in separate in the second part of the chapter.

The framework was integrated in an independent project aiming to adapt the Session Initiation Protocol (SIP) to ad hoc networks. An overview of the project and of the contribution of the algorithms described in the thesis is presented in Chapter 5. The chapter also describes and evaluates the data gathering module of the project, that benefits of the dissemination capabilities of the framework to retrieve the data items satisfying some condition.

The most significant conclusions of the thesis and some directions for the continuation of this work are presented in Chapter 6.

Related Publications

Preliminary versions of portions of this dissertation have been published in the following:

- MIRANDA, HUGO, LEGGIO, SIMONE, RODRIGUES, LUÍS, & RAATIKAINEN, KIMMO. 2006a (Sept. 11–14). A power-aware broadcasting algorithm. *In: Proceedings of the 17th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'06)*. University of Oulu, Helsinki, Finland.

  This paper presents the Power-Aware Message Propagation Algorithm (PAMPA). The performance of PAMPA is estimated using a network simulator and compared with other message propagation algorithms.

- MIRANDA, HUGO, LEGGIO, SIMONE, RODRIGUES, LUÍS, & RAATIKAINEN, KIMMO. 2006b (Dec. 12–15). An algorithm for distributing and retrieving information in sensor networks. *Pages 31–45 of: Proceedings of the 10th International Conference on Principles of Distributed Systems (OPODIS 2006) - part II - Brief Announcements.*

  This brief announcement paper presents and evaluates a preliminary version of Power-Aware data DISsemination algorithm (PADIS), the data dissemination algorithm described in the dissertation.

- MIRANDA, HUGO, LEGGIO, SIMONE, RODRIGUES, LUÍS, & RAATIKAINEN, KIMMO. 2007 (Aug. 28–31). An algorithm for dissemination and retrieval of information in wireless ad hoc networks. *Proceedings of the 13th International Euro-Par Conference - European Conference on Parallel and Distributed Computing*. Lecture Notes in Computer Science. Rennes, France: Springer. (to appear).

  The most up-to-date version of PADIS is described in this paper. The performance of PADIS is estimated by varying multiple parameters using a network simulator.

- MIRANDA, HUGO, LEGGIO, SIMONE, RODRIGUES, LUÍS, & RAATIKAINEN, KIMMO. 2006c. *Global data management*. Emerging Communication: Studies in New Technologies and Practices in Communication, vol. 8. Nieuwe Hemweg 6B, 1013 BG Amsterdam, The Netherlands: IOS Press. Chap. Epidemic Dissemination for Probabilistic Data Storage, pages 124–145.

  This book chapter begins with a characterisation and overview of recent results on gossip protocols. It then proceeds to describe and evaluate a preliminary version of PADIS named PCache and the data gathering mechanism that is presented in this dissertation.

- LEGGIO, SIMONE, MIRANDA, HUGO, RAATIKAINEN, KIMMO, & RODRIGUES, LUÍS. 2006 (July 17–21). SIPCache: A distributed SIP location service for mobile ad-hoc networks. *In: Proceedings of the 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MOBIQUITOUS 2006)*.

  This short paper presents SIPCache, a Session Initiation Protocol (SIP) for ad hoc networks. The paper focus on the integration of the three components that result in SIPCache: dSIP, PADIS and the data gathering mechanism.

2

Related Work

The frequent disconnections, low reliability and limited resources of mobile devices make efficient data dissemination and data retrieval challenging problems in wireless networks. This chapter surveys research results on data dissemination protocols.

Wireless networks can be applied on a multitude of application scenarios, each characterised by different data access patterns and device capabilities. The chapter reflects this heterogeneity by surveying data dissemination protocols for sensor, infrastructured, hybrid and ad hoc networks. The goal is to identify approaches that, independently of the original application scenario, can be applied in MANETs. In addition, from the survey will emerge an interesting and useful approach for MANETs which, to the extent of our knowledge, has not been attempted before and will be the focus of the thesis.

This chapter is organised as follows. Section 2.1 identifies the most relevant components of a generic data replication service. Related work on these components, namely local storage management, data distribution and packet dissemination is presented respectively from Section 2.2 to Section 2.4. The most relevant conclusions extracted from the survey are summarised on Section 2.5.

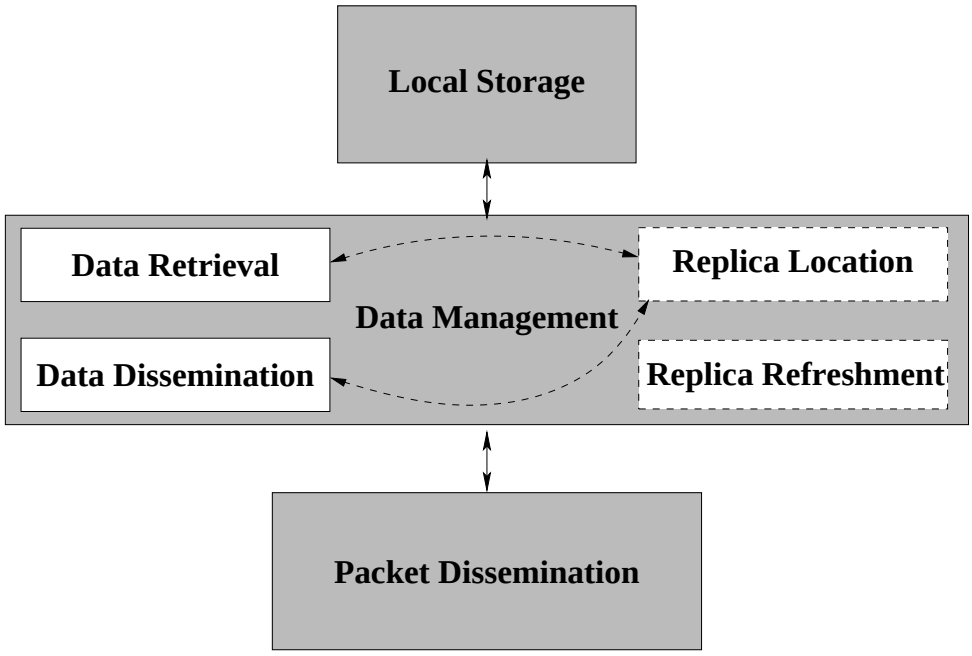


Figure 2.1: Generic architecture of data distribution protocols

2.1 General Architecture of Data Distribution Protocols

Any data distribution protocol provides two complementary services: data dissemination and data retrieval. The implementation of these services can be decomposed in several modules, each providing a basic functionality. A possible decomposition, which will be used throughout the rest of the chapter is presented in Figure 2.1.

The "local storage" module represents the storage space made available by each device. The literature frequently assumes that data produced by the node is kept in a separate region with unlimited size and that only data produced by other nodes is managed by the local storage module. Most protocols assume that nodes make limited storage space available for the storage space module. Section 2.2 describes the different storage space management policies presented in the literature.

The "data management" module aggregates all the functions related with the dissemination and retrieval of the data items. A myriad of algorithms have been proposed in the literature. They are compared in Section 2.3.

Conceptually, the "data management" module is decomposed in four submodules.

The algorithms supporting data replication are provided by the "data dissemination" submodule. The "replica refreshment" submodule is present only in some of the protocols surveyed. It is responsible for monitoring and correcting either the number or the location of the replicas. The "data retrieval" submodule issues queries for data items and collects and prepares replies.

Some protocols apply deterministic functions to decide the location or identity of the nodes hosting the replicas. We assume that these protocols use a "replica location" submodule. If available, the "Replica Location" submodule is used by the "data dissemination" and "data retrieval" submodules respectively to decide the location of the replicas and the destination of queries.

Although conceptually independent, the algorithms implemented in the "data dissemination", "data retrieval" and "replica location" modules are typically tightly coupled. For clarity, these modules are aggregated in the description of the related work.

The "packet dissemination" module exports the message reception and dissemination primitives that interface the underlying network. Depending of the algorithms implemented in the remaining modules, point-to-point, multicast and broadcast algorithms may be required. Section 2.4 addresses this module, putting particular emphasis on broadcast algorithms as they will be one of the building blocks of the algorithms presented in the thesis.

2.2 Local Storage Management

The "local storage" module manages a device's storage space made available for storing replicas of data items advertised by other nodes. When the storage space made available by a device is full, an algorithm must be applied to decide which data items should be kept and which should be discarded. This algorithm plays an important role in the overall performance of data distribution protocols. Local storage management has been previously addressed in different settings, such as operating systems,

databases and web caching. The Least Recently Used (LRU) algorithm is frequently cited as an example of a policy offering good performance in different settings. The literature has also shown that for wireless networks, it is possible to devise specialised algorithms that provide a better performance than more general solutions.

Two different approaches, specifically addressing the wireless environment have been proposed, namely: $i)$ algorithms accounting with the popularity of the data objects aim at reducing the average reply time by keeping the most popular items, so that queries can be replied faster. $ii)$ Other algorithms privilege data availability, giving preference to the diversity of the items in the collective storage space defined by the union of the individual storage space of the nodes in some neighbourhood.

The *Global-Cache-Miss initiated Cache Management* (GCM) and the *Motion-aware Cache Management* (MCM) (Wu & Tan, 2006) are examples of algorithms privileging the popularity of the items. Both algorithms were devised for infra-structured wireless networks with a centralised, resourceful data server that repeatedly broadcasts the data items. The goal of the algorithms is to reduce latency, by retrieving the data items either from their local cache or from the cache of the nodes in their direct transmission range (so that nodes do not have to wait for the next retransmission from the data server).

In GCM, a node checks if the data item is stored in its local cache and then repeatedly probes its 1-hop neighbours until either the item is delivered by one of its neighbours or broadcasted by the data server. Each node preferably populates its storage space with data items ranked according to the formula presented in Equation 2.1.

$$\text{NumAccess}\frac{\text{NumGCM}}{\text{NumLCM}} \tag{2.1}$$

where *NumAccess* is the number of requests for the object and *NumLCM* and *NumGCM* count respectively the number of failed retrievals for the object from its local cache and from the cache composed by all 1-hop neighbours. NumGCM is incremented with each periodic retransmission of the query without a reply. A low

NumGCM represents either an item that is popular in the cache of the neighbours or frequently broadcasted by the data server.

MCM ranks the candidates for replacement in local storage according to a "PIX" (P Inverse X) score, originally described in Acharya *et al.* (1995). PIX score is given by the ratio of the access probability (P) for the frequency with which the item is broadcast by the data server (X). An interesting aspect of MCM is that the decision to cache an item depends of the estimated availability of a replica stored on a node in proximity. The future distance between two nodes is calculated by comparing their position, direction and speed. Comparisons between GCM and MCM show that in general, GCM outperforms MCM although at expenses of additional energy consumption.

In (Sailhan & Issarny, 2003), nodes in an hybrid network cache web pages, making them available to other nodes. Each node ranks its cached web pages according to their popularity and access cost. *Popularity* estimates the probability of future accesses from past requests. The *AccessCost* estimates the cost of retrieving the web page from another node, which may possible be located several hops away. Finally, the system prefers to remove older and bigger documents.

Popularity is a relevant criteria when it is known that data will be available in a bounded time. Other algorithms give priority to data availability, and therefore, are more adequate for scenarios where such a bound does not exist. This is the most expected scenario in MANETs. For example, in the algorithm described in (Lim *et al.*, 2006), a data item is only cached if the object was sent by a node more than a predefined number of hops away. Cached data item are selected to be replaced by one of three *Time and Distance Sensitive* (TDS) algorithms. TDS combines the distance to the closest node known to also store a copy (δ) with the inverse of the time elapsed since the distance was measured (τ). τ is used to estimate the accuracy of the distance measurement. The three TDS algorithms are:

TDS_D given by $\delta + \tau$. Because $\delta \geq 1$ and $0 \leq \tau \leq 1$, it can be said that TDS_D selects the item with the lower δ and uses τ for tie-breaking;

TDS_T that selects the item with the lowest τ.

TDS_N which selects the lowest value of $\delta \times \tau$.

Simulation results concluded that in general, the three algorithms perform better than the traditional LRU (Least Recently Used) cache update policy. Of the different TDS policies, no clear winer can be found, as they present different results depending of the network settings. TDS algorithms have a limited application as they require nodes to be equipped with location devices.

2.3 Data Dissemination Algorithms

Intuition suggests that there is a tradeoff between the effort placed on the dissemination of the data and the effort required to retrieve it. In principle, a strategy that saves the resources during the dissemination, for example by placing the replicas in nodes close to the producer will require the consumption of more resources for data retrieval. This observation motivated the classification adopted in this section, which is depicted in Figure 2.2. At first, algorithms are arranged according to the level of awareness of the nodes on the location of the data. The term location is used in a broad sense encompassing both geographical location and node addresses. In this taxonomy, a node is aware of the location of some data item if it knows either a geographical coordinate or the address of a node and one of them can be used to retrieve the data item.

Algorithms that are unaware of the location of data items are arranged by their effort on the geographical distribution of the replicas in two classes: "location unaware" algorithms completely ignore the state of other nodes in the strategies for keeping data items in their storage space; "Improved location unaware" algorithms perform some form of leveraging of the replicas to prevent excessive redundancy in the neighborhood of each node.

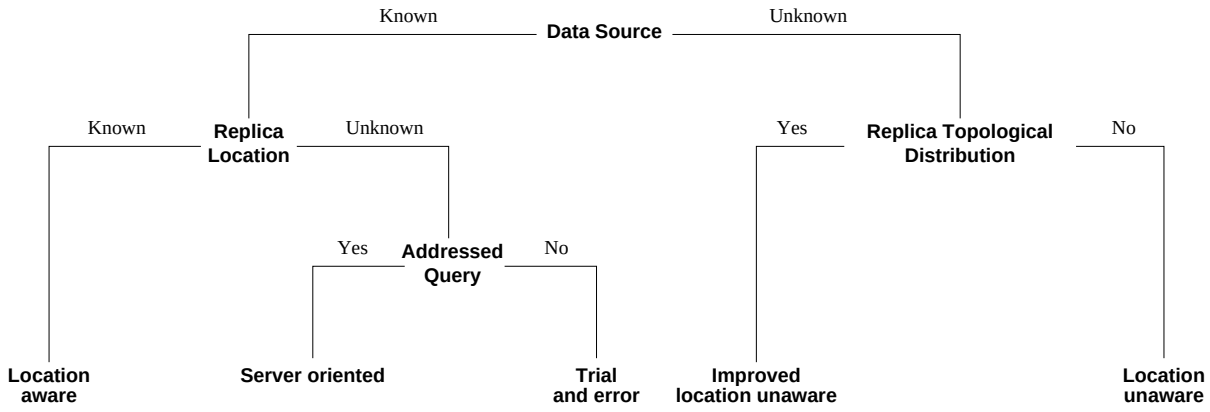


Figure 2.2: A taxonomy of data dissemination algorithms

On the opposite side of our classification are the algorithms that are aware of the location of every replica. These algorithms have been grouped in a class named "location aware". The algorithms that are aware of only one copy are arranged by the strategy used for performing the queries. The "server oriented" class groups the algorithms that address the query to the location of the known copy of the data. As the name implies, "trial and error" algorithms perform a preliminary step trying to find some replica in a more favorable location than the one that is known.

2.3.1 Location Unaware Algorithms

Location unaware algorithms typically use the less efficient dissemination procedures. Of the four algorithms described in this section, three perform on-demand replication, with nodes caching only the data that has been received in result of a query. Another common aspect of these algorithms is the absence of cooperative management of the available distributed storage space.

A simple location unaware algorithm was used to estimate the equilibrium point between the effort that should be placed on dissemination and retrieval (Krishnamachari & Ahn, 2006). The goal is to determine the most adequate number of replicas of a data item. In the algorithm, nodes disseminate data using point-to-point messages addressed to random nodes. Queries progress in successive expanding-ring search broadcasts with TTL increasing according to a dynamic programming se-

quence (Chang & Liu, 2004). The authors conclude that, in this setting, the optimal number of replicas of each item is proportional to the square root of its query rate. However, this result is of limited application because it assumes that the query ratio of some item will be known by the producer. In addition, it should be expected that in non-random deployment of the replicas, for example like those presented below, that take into account the popularity of the data items, queries can use different sequences of TTLs.

The *Simple Search* (SS) algorithm (Lim *et al.*, 2006) was devised for hybrid networks. It is assumed that nodes only keep items they have previously requested in their storage space. *Simple Search* improves access latency for the cases where the requested data is stored in some node closer than the data server, co-located with the base station.

In SS the queries are broadcasted to the network in a *request* message with a predefined value in a Time-To-Live field. All nodes that receive the *request* packet and do not store the object append their address to the request packet, decrement the TTL field and, if the TTL field permits it, retransmit the packet. Instead of retransmitting the request, nodes capable of replying to the request, including the data server, send a point-to-point *ack* message to the source of the request. The *ack* message follows the path accumulated in the *request* message. The querying node requests the object to the source of the first *ack* message received using a *confirm* message and discards the remaining. The data is delivered by the node that receives the *confirm* message.

In contrast with *SS*, the *Rumour Routing*, *Static Access Frequency* and *Autonomous Gossipping* algorithms (described below) assume that nodes are simultaneously the producers and consumers of data. Each assumes, however, a different networking model. The *Rumour Routing* algorithm relies on the stability of the network links between the nodes and is therefore more adequate for sensor networks without node movement. *Autonomous Gossipping*, instead, relies on the temporary links established by moving nodes to disseminate data. *Static Access Frequency* considers that nodes are in transmission range for some time interval.

Rumour Routing (Braginsky & Estrin, 2002) does not replicate data. Instead, it im-

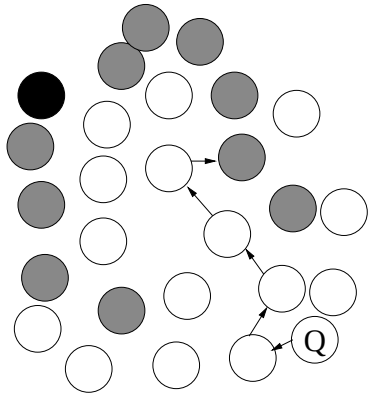


Figure 2.3: Intersection of trails and queries in rumour routing

plements an algorithm to facilitate the location of the producer of each data item. The algorithm is based on the assumption, justified by the authors with Monte-Carlo simulations, that the probability of two lines in a bounded rectangular region to intersect is of approximately 69%. In addition, the probability of one line to intersect at least one of five others is of 99.7%. Although it is impossible to define straight lines in sensor networks, authors show that the algorithm still provides acceptable results. The algorithm creates trails pointing to the node that owns the data. The trails are left on the nodes by agents performing random walks. The agents are initialised by the producer node. To prevent loops, agents record the path of the nodes that have been visited. The agents accumulate the state of the storage space and trails previously recorded in the nodes they visit. Information learned in previous visits is made available by the agents to the nodes in the following steps of the random walk so that multiple trails can be created and updated. Figure 2.3 shows in gray the nodes that store a trail for a data object stored at the node in black.

A node performing a query begins by locally verifying if it knows a trail for the data item. If no trail is found, the query is successively forwarded using a random walk until a node in a trail is encountered or the TTL of the query expires. Nodes in a trail forward the query to the respective source node. The Monte Carlo simulations suggest that if a sufficient number of trails is defined, there is a good probability that any query message forwarded in a random walk will intersect one of the trails created by the agents. As a last resort, and if no reply is received after some number of retries,

the algorithm broadcasts the query.

The *Static Access Frequency* (SAF) algorithm (Hara, 2001) tries to improve data availability in the presence of network partitions by predicting future access by the applications to the data items. SAF progresses in rounds. At the beginning of each round, nodes rank the data objects by access frequencies and try to populate their storage space with the data items with the highest ranks. Storage space must be occupied by the designated objects. If some object cannot be retrieved, the storage space will remain free until the object is retrieved from the network. SAF is a simple algorithm that does not require coordination between the nodes. This uncooperative behaviour will likely replicate the most popular items a large number of times, decreasing the number of items that could be retrieved from the neighbours.

SAF was extended to address the case where the access pattern of the nodes discloses some correlation between different data objects (Hara *et al.*, 2004) and to handle periodic updates of data items (Hara, 2002). For clarity, the improved versions of the algorithms (respectively C-SAF and E-SAF) are discussed together with other algorithms proposed by the same authors in Section 2.3.2.

Autonomous Gossipping (Datta *et al.*, 2004) uses ecological and economical principles as an alternative to classical access frequency definitions. Autonomous Gossipping considers each host to be an habitat providing storage space for a limited number of data items. The habitat may be more or less favourable to the data item, depending on the relevance of the data item to the host.

Both hosts and data items have a profile, defining respectively the topics of relevance for the host and the topics covered by the data item. A function *similarity* numerically rates the proximity of the profiles of hosts and data items. In addition, data items carry an attribute describing the geographical locations where the item is supposed to reside and a scalar called the *associated utility*, measuring the utility of the item to the host. The utility of an item can be incremented for example when the item is used by some application and decremented otherwise. Hosts advertise a goal zone to let data items learn about their intended destination.

The system partially implements a "survival of the fittest" model. Opportunistic gossipping, defined by the temporary interaction made available by the movement of the nodes, is used for data items to evaluate the conditions of their hosts, and consider their migration for more favourable habitats. Inside each host, data items compete for their survival, knowing that the hosts attempt to maximise the accumulated *associated utility* of the data items they host. When an opportunity appears, data items decide to stay, migrate or replicate to another host with a more suitable profile, according to the following policies:

Migration A data item decides to migrate when the host provides adverse conditions, defined by a *similarity* below some threshold and low utility. The decision to migrate is taken without the data items becoming aware if the migration will provide better conditions.

Replication A data item may decide to replicate if it has an high utility and *similarity* above a threshold. To prevent an excessive population of some data item, and again following the ecological model, the utility of the original data item and its replica will be lower than its value before replication.

Replica reconciliation occurs when a data item finds a copy of itself in the host to where it has migrated. The utility of the re-conciliated data item is higher than the utility of both replicas.

Migration anyway Independently of the friendliness of its current habitat, a data item may decide to migrate if the current host is outside its target location (if any). Migration is attempted to hosts whose goal zone is the same of the data item.

Simulation results (Datta *et al.*, 2004) confirmed that autonomous gossipping is capable of delivering messages to a large proportion of the interested nodes. However, it leaves some questions unanswered. Intuition suggests that an high threshold may result in nodes becoming excessively judicious about the hosted data items what may endanger the message dissemination. This can be compensated either by increasing

the size of the habitat or the frequency of contacts so that the probability of finding an interested node is increased. However, simulation results presented are not sufficient to evaluate this tradeoff.

2.3.2 Improved Location Unaware Algorithms

Although unaware of their precise location, improved location unaware algorithms use some form of topological information to distribute the replicas. This section presents three examples. The first degrades the accuracy of the information with the distance to the source. In the remaining, nodes in proximity cooperate to improve data availability by coordinating the content of their storage space.

The concept of *non-uniform information* is introduced in Tilak *et al.* (2003) to describe one application model for sensor networks where the relevance of the information degrades with the distance from the source to the consumer. In the *non-uniform information* model it is assumed that instead of delivering data to powerful sink nodes, fixed over the network, sensor nodes will deliver data to "passing by" sink nodes. Examples are military applications, where the presence of enemies nearby is more relevant than learning about a distant fight.

Two categories of algorithms are presented in the paper to limit the propagation of the information: deterministic and randomised. In both, the goal is to progressively reduce the number of forwarders of every message by successively applying at each hop a mechanism that only propagates some of the messages it receives. In the *Filtercast* deterministic algorithm, each node forwards every n^{th} message received from the same source. However, if all nodes are configured with the same constant, all will propagate the same message from each source. If this synchronisation is removed, the accuracy of the information propagated is improved. This is addressed in the *RFiltercast* deterministic algorithm, where although forwarding every n^{th} message, each node initially peaks a random number that will decide which of the n messages is transmitted. Simulation results showed that using *RFiltercast*, nodes transmit more messages

than with *Filtercast*. Both deterministic algorithms present some scalability problems as they require that each node keep a counter for each source.

Randomised algorithms are stateless and therefore scale better with the size of the network. In the *unbiased* algorithm, each packet is retransmitted with an independent probability. The *biased* algorithm, in turn, improves the probability of retransmission for packets with an higher Time-To-Live field, what corresponds to an higher proximity to the source.

Non-uniform information algorithms effectively reduce the number of retransmitted messages and therefore, contribute to increase the network lifetime by saving node's batteries. However, the lack of semantic interpretation of the filtered data raises concerns of the utility of these algorithms.

The *Dynamic Access Frequency and Neighbourhood* (DAFN) and the *Dynamic Connectivity based Grouping* (DCG) algorithms (Hara, 2001) predict the items that will be accessed by each node in the future from previous accesses. Like in SAF (described in Section 2.3.1), the estimation is used to improve data availability in the presence of network partitions. Both algorithms progress in rounds. At each round, nodes re-evaluate their access pattern to the data and negotiate with their neighbours the data items that should be stored so that data availability is improved for the group.

In DAFN, nodes negotiate with their 1-hop neighbours. Each round begins with the broadcast of a message by every node. The message notifies the neighbours about the presence of the node and includes information about the node's access frequency to the data items. Nodes then follow a deterministic algorithm to populate their storage space. For each 1-hop neighbouring set of nodes, the algorithm is sequentially applied from the one with the lowest ID to the one with the highest. If two neighbouring nodes intend to store the same data item, the algorithm dictates that:

- If one of the nodes is the producer of the object, the other node will not store it;
- If none of the nodes is the producer of the object, the item is stored by the node with the highest access frequency.

The storage space made available by duplicate elimination will be filled by the next objects in the node's rank. If some node cannot retrieve an object from the network, the space will be temporarily filled with some other object until the node is able to retrieve a replica of the preferred object.

Although DAFN improves replica distribution over SAF, it still allows some redundancy between nodes two or more hops away. However, it should be noted that the major goal of replication is to improve availability. Therefore, some tradeoff must be found between the number of replicas of each data item that can be reached by the nodes and the probability of, in the future, none of the nodes storing them to be reachable. DCG estimates this probability by counting the number of links that must be broken between any two nodes. Instead of using 1-hop neighbours, DCG groups nodes connected by at least two disjoint paths and removes replica redundancy among them. Each group estimates their access frequency to the data items by summing the access frequency of all its members. Data items are allocated to the node with the highest access frequency for the item.

DCG is the most effective of SAF, DAFN and DCG algorithms in the removal of redundancy. However, it is also the one requiring the more power consuming operations. In dense networks, i.e. those where a large number of nodes can be reached, DCG will require the transmission and forwarding of a large number of messages. Another benefit of replication, not considered in DCG, is the improvement of the access latency to the data. DCG may increase latency in large scale networks because it will likely increase the average distance of each node to the replicas of the items.

Simulations assumed a random movement of the nodes and experimented different access patterns to the data. DCG showed to be the best performing one for every scenario at the expenses of the highest number of messages. The most interesting results are presented with increasing levels of randomness in the queries. The more random the queries, the more the performance of the three algorithms approximates. However, DCG continuously increases traffic while the traffic in DAFN tends to be reduced and in SAF it is constant and significantly lower. Finally, simulations con-

firm that the performance of all the algorithms is highly by the number of neighbours and the storage space at each node. In extreme cases, any of the algorithms performs equally. The availability of the data can be differentiated only for the intermediary cases. However, the same conclusions do not apply to the traffic, being even impossible to establish a correlation between the traffic and the data availability results.

SAF, DAFN and DCG were extended respectively to C(Correlation)-SAF, C-DAFN and C-DCG to address the case where the access patterns discloses some correlation between independent data items (Hara *et al.*, 2004). The strength of correlation is defined for each node and pair of items as the frequency of access to the two items by the node. It is assumed that the strength of correlation between each two data items is known and does not change. In these algorithms, the access frequency is replaced by the data priority, estimated by each node from the access frequency of the item and from the sum of the strength of the correlations of the item with all other items.

Evaluation was performed by comparing the original methods with those using correlation in a network where nodes request data items with a correlation predefined by the authors. The performance of each algorithm relative to each other mimics the results presented in the original version. In this scenario, correlation-aware methods provide better accessibility than non-correlation methods while increasing the traffic at the nodes.

E(xtended)-SAF, E-DAFN and E-DCG handle periodic updates of the data objects (Hara, 2002). The algorithms assume that items are periodically updated with a known and constant rate. The access frequency metric for each item is replaced by a *PT* value representing the average number of access requests which are issued for an item until its next update. *PT* is given by $p_{ij}.\tau_j$, where p_{ij} is the access frequency of node N_i to item D_j and τ_j denotes the time remaining until the next update of item D_j. Preference is given to replicate data objects with higher *PT* values as they represent the objects that will be valid for more time and with an higher access ratio.

Simulation results show that when the frequency of the updates is significantly higher than the frequency of the rounds, the rapid invalidation of the caches attenuates

the gains of the extended algorithms. However, the frequency of the rounds can not be excessive because previous results (Hara, 2001) showed that it significantly increases traffic.

2.3.3 Trial and Error Algorithms

In the "trial and error" and "owner oriented" classes of algorithms, nodes are aware of the ID or location of the primary copy of the data but not of the replicas. These classes are distinguished by the effort put by the querying node in retrieving the data from the replicas. In general, trial and error algorithms weight the cost of addressing the query to the server or perform a preliminary round probing a limited number of nodes for the data and select the most advantageous option. In some algorithms, and in the absence of failures, the preliminary round always succeeds and is the default mode of operation. The examples below show the application of trial and error algorithms in two distinct types of networks: hybrid networks and sensor networks where nodes are equipped with location devices.

A simple example of a trial and error algorithm for hybrid networks is 7DS (Papadopouli & Schulzrinne, 2001). In 7DS, each node can be in one of two modes: infrastructured or ad hoc. If the node is in range of a base station, queries are directly addressed to it. Otherwise, queries are broadcasted to the node's 1-hop neighbours and replied by those having the resource in their local cache. Resources are only cached by nodes that have previously requested them.

The 1-hop query propagation of 7DS limits the probability of successful retrieval of the data. The algorithm described in (Sailhan & Issarny, 2003) removes this limitation. It aims specifically at caching web information in hybrid networks but could be easily extended for other application domains. In this algorithm, nodes create profiles of other nodes. A profile describes the interests of a node and are created by observing the queries the node sends.

The algorithm is tightly integrated with the Zone Routing Protocol (ZRP) (Haas,

1997). The ZRP combines reactive and proactive mechanisms. Each node pro-actively collects routing information for the nodes in its neighbourhood (the "zone" of the node). Routing information for nodes outside its zone is collected on-demand.

The algorithm operates in two phases: replica location and data retrieval. Replicas are located by successively performing a sequence of steps from the one with the lowest cost for retrieving the data to the one with the highest cost. The steps performed are:

1. If an access point is located in its zone, the data is retrieved from the access point;
2. The node broadcasts a query to the nodes in its zone;
3. The node sends point-to-point queries to some nodes outside the zone but at a lower distance than the data server. The nodes selected are those known to have a profile matching the requested web page, ordered according to a rank that weights the node's profile and the distance, in hops, between them.
4. The query is sent to the closest access point.

Only positive replies are sent to the querying node. If several replies are received, what may happen in Steps 2 and 3, nodes are ranked by weighting the distance (in hops), the freshness of the data and the resources available on the node for each reply. The resource is requested to the most suitable node.

Double rulings algorithms are an example of trial and error algorithms for sensor networks where the devices are aware of their location. These algorithms use sphere projections to determine the location of the replicas so that they form a circle as depicted in Figure 2.4. The work described in (Sarkar *et al.*, 2006) uses stereographic projections.

In this algorithm, the area of the sensor network is considered as a finite region of the plane. Since the stereographic projection maps a plane in a sphere, the sensor network is mapped in a region of the sphere. The rationale for this algorithm comes from

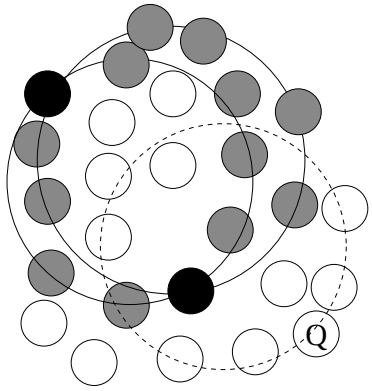


Figure 2.4: Storage and query circles in Double Ruling storage

noticing that in a sphere, any two great circles intersect. Double rulings algorithms use one such great circle for determining the location of the replicas and another for forwarding the queries.

In double ruling all nodes can derive the location of the primary copy from a mapping of the key or type of the data in a geographical coordinate inside the network perimeter. This mapping is usually referred as Geographical Hash Tables (GHTs) due to its similarities with Distributed Hash Tables (DHTs), which map the key of a data item on an address.

Every data object is stored along the projection on the plane of the great circle passing by the producer and the hash of the data type given by a GHT. Storage is performed by nodes forwarding the object along the circle. A property of the storage circle is that, independently of the location of the source, it necessarily includes the hash point and its antipode point in the sphere. The nodes closer to these locations perform aggregation functions that speed up and reduce the amount of energy required to retrieve data summaries. Figure 2.4 shows two storage circles and represents in black these two points.

Data summaries can be retrieved by addressing the queries using any geographical routing algorithm (like GPSR (Karp & Kung, 2000) or other) to the hash point or to its antipode. Data can be retrieved by having the query to follow any closed curve that separates the two hash points, knowing that the curve finds a replica of every data

object stored.

The trial and error classification of this algorithm comes from noticing that any data item will eventually be found if the query is forwarded along a query curve. A query curve is depicted in a dashed circle in Figure 2.4. The curve is determined by an algorithm that first rotates the sphere so that the hash point is placed at the top of the sphere and then deriving a latitude circle over the sphere. It is shown in the paper that, if the distance between the producer of the item and the querying node is d, the distance to the closest replica is in the order of $O(d)$. Double ruling algorithms have the desirable property of avoiding traffic concentration at the hash points, thus improving fairness and load balancing.

The Geographical Location Service (GLS) (Li *et al.*, 2000) was designed to facilitate the mapping of node identities on their current location but can be applied in other domains. All nodes have an unique identity, selected from a circular identity space. The goal of the algorithm is to facilitate the retrieval of the approximate location of any node from its identity, so that a geographical routing algorithm like GPSR (Karp & Kung, 2000) can be used for packet forwarding.

The algorithm assumes that all nodes are aware of a geographical partition of the networked region. A simple example is to partition the network in squares, as depicted in Figure 2.5. An hierarchy is established with squares of order n being successively grouped in squares of order $n+1$ with sides that double the size of the squares of order n. A fundamental requirement of the algorithm is that each square of order n is part of precisely one square of order $n+1$. Figure 2.5 shows 16 order 1 squares, grouped in 4 order 2 squares and in one order 3 square.

The algorithm geographically distributes the information by the network. Some node S, updating its location, registers the information with progressively lower density of replicas as the size of the squares increases. The location of S will be stored at each square of order 1 in the square of order 2 where S is located. One copy will be stored at each of the remaining squares of order 2 that share the same order 3 square, etc. In practice, S stores 3 replicas at each level of the hierarchy. Figure 2.5 depicts

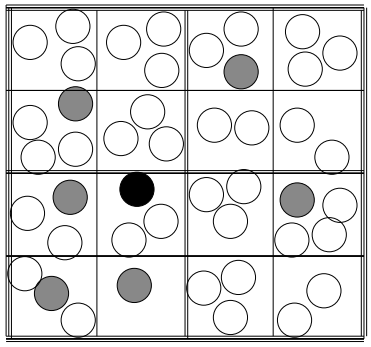


Figure 2.5: Network region partitioning in GLS

in gray a possible deployment of the replicas. Replicas of the location of S are stored at the node with the least identity greater than the identity of S on the target region. However, the effective identity of the nodes storing the replica is unknown to S. Instead, the packet is forwarded to some point in the target square using geographical routing. The first node in the respective square that receives the packet will use the same algorithm followed by queries (described below) to find the target node.

A node issuing a query for S and not locally storing S's location will forward it to the node with the closest identity of which it stores the location. Nodes successively perform a similar operation until a reply to the query is found. It should be noted that this algorithm converges by approximating the query to some node storing the replica. It was shown that the number of forwardings is not more than the order of the square on which the querying node and S are co-located (Li *et al.*, 2000).

2.3.4 Owner Oriented Algorithms

In owner oriented algorithms, queries are addressed to the node known to hold the item. This section presents two variations of owner oriented algorithms. One implements a transparent caching mechanism in hybrid networks. In the second, queries are always replied by the owner node. Replicas are made available to tolerate faults and node movement.

In the *Cache the Data* (CacheData) protocol (Yin & Cao, 2006) requests are always

addressed to the access point of the hybrid network. Nodes cache data when receiving it as a result of a query or after repeatedly forwarding the same data item to different next hops. Subsequent queries for the same data item are replied by the node instead of forwarded to the access point. Notice that the decision to cache is based on the next hop of the reply not the destination. This prevents multiple nodes in a frequently used route from caching the same data item.

The data-centric storage (DCS) algorithm described in (Ratnasamy *et al.*, 2002; Ratnasamy *et al.*, 2003) relies on the Greedy Perimeter Stateless Routing (GPSR) (Karp & Kung, 2000) geographical routing protocol. GPSR uses two routing algorithms. The preferred one is *greedy forwarding*, where each node forwards the messages to the node in its 1-hop neighbourhood that is closer to the target location. The *perimeter forwarding* mode is activated whenever *greedy forwarding* can not be used. In particular, when some node N is not at the destination but can not find any node closer to the target location to forward the packet. In this case, the packet enters the *perimeter forwarding* mode and begins to circle the region until it is able to resume *greedy forwarding*. If it is not possible to resume *greedy forwarding*, the packet is again delivered to N after circling the target location using the closest nodes.

The *home node* for some data object is the node closer to the target location given by the GHT mapping.[1] It is the node responsible for storing the primary copy of the data. The *home perimeter* for some data object is composed by the nodes that forward a packet, sent by the *home node*, addressed to the target location in *perimeter forwarding* mode. Nodes in the home perimeter store a backup copy of the object. These concepts are illustrated in Figure 2.6. The target location is represented with an X. The nodes composing the home perimeter are depicted in gray and the home node in black.

Periodically, the home node initiates a *refresh algorithm* which consists in sending a *Refresh Packet* addressed to the target location of the data object. *Refresh Packets* carry the data objects stored by the node. GPSR ensures that *Refresh Packets* will traverse the home perimeter of the object. In the absence of node movement or failure, *Refresh*

[1] Geographic Hash Tables (GHTs) were introduced in Section 2.3.3.

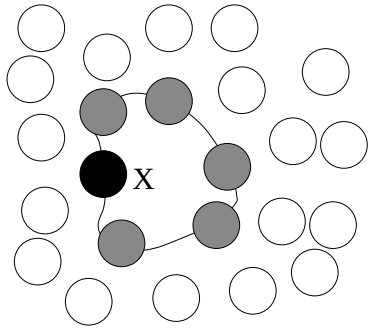


Figure 2.6: Home node and perimeter in Data-Centric Storage

Packets are equally used to signal the availability of the home node.

Nodes receiving a *Refresh Packet* enter or update the data object in their caches. Therefore, nodes that recently joined the *home perimeter* are implicitly included in the list of replicas. Additionally, each node in the home perimeter compares its distance to the target location with the distance of the home node to the target location. If closer, the node elects himself as the new home node. Others (including the previous home node) will be notified by an updated *Refresh Packet*. It should be noted that this procedure implicitly addresses the case where the home node moves away from the home perimeter.

Nodes in the home perimeter may also initiate the *refresh algorithm* if they do not receive a *refresh packet* after a predefined amount of time. This procedure triggers the replacement of a failed home node and will trigger the election of a new one.

An interesting case occurs when a home node issues a *refresh packet* for a data object but never receives it. This implicitly indicates to the node that it is no longer included in the *home perimeter* what may occur either because the node moved away or because some other node became closer to the target location. After a predefined timeout without receiving any *refresh packet*, nodes purge the data object from their caches.

2.3.5 Location Aware Algorithms

In location-aware algorithms, nodes know the location of both the primary copy of the data and its replicas. This section presents three examples of these algorithms. The first two are designed for hybrid networks. The third extends DCS by geographically distributing the replicas and uses a deterministic algorithm to derive their location.

The *Cache the Data Path* (CachePath) (Yin & Cao, 2006) algorithm was designed for hybrid networks. Nodes store the replies to their queries. Intermediary hops that forward the reply record the key of the item and the destination of the message.

Queries are always addressed to the access point. However, a node forwarding a query may redirect it, if it is aware of a replica of the item in a node located closer than the access point. Nodes rely on the underlying routing protocol to learn the distance (in hops) to the access point and to the caching node as well as to forward the messages.

In comparison with CacheData (presented in Section 2.3.4), CachePath requires less storage space because only an address, not the object is stored. In addition, the algorithm implements some mechanisms to prevent nodes from storing useless information. Nodes that are closer to the access point than to the destination of a reply do not record information because it is more cost efficient to forward the query to the access point. Also, because node movement can rapidly invalidate distance information, addresses are cached only if the replica is stored up to a maximum predefined distance. Nodes that redirect queries are responsible for confirming the availability of the data item or to retransmit the query to the access point if the object is not provided.

Analytical comparisons showed that CachePath and CacheData present advantages over a baseline method where all queries are forwarded to the data centre. CacheData outperforms CachePath when the storage space available at the nodes is larger, when data items are frequently updated or the topology changes. In the opposite situations, CachePath is preferable. HybridCache (Yin & Cao, 2006) selects the most favourable of CachePath and CacheData on a per-query basis according to three rules:

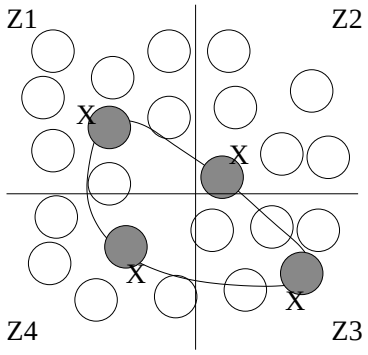


Figure 2.7: Monitor nodes in Resilient-Data-Centric Storage

- CacheData is used if the data item is small;
- CacheData is used if the object is frequently updated. CachePath is used otherwise;
- CachePath is used when the gains (in hops) are significant, i.e. when the distance to the data centre is high in comparison with the distance to the caching node.

Simulation results showed that HybridCache outperforms CacheData and CachePath.

Storing the replicas of each data item in proximity is considered undesirable because such a system is unable to tolerate localised failures which may result for example from interference (Ghose *et al.*, 2003). Resilient Data-Centric Storage (R-DCS) (Ghose *et al.*, 2003) improves data availability by geographically partitioning the network in zones and storing at most one replica on each zone (see Figure 2.7). R-DCS implements a two layer architecture, with some nodes storing replicas of the data and others keeping summaries of the cached information. R-DCS extends the GHT function with a zone parameter so that a different location, within the zone, is returned for each zone. Like in DCS (described in Section 2.3.4), the node closer to the target location in each zone is the home node for the object. In R-DCS, this node is named the "monitor node".

Data storage is optional for monitor nodes and depends of its available resources. A monitor node that simultaneously stores data is called a replica node. At a minimum,

any monitor node is required to store a map containing:

- A list of the zones containing replica nodes;
- A list of the zones containing monitor nodes;
- Event summaries, to facilitate summary-mode queries;

Data objects are stored by forwarding them to the corresponding monitor node in the source's zone. If the monitor node is not storing data, it will forward the object to the closest replica node. All queries are also addressed to the monitor node in the region of the source. Depending on the type of the query, the monitor node will either reply directly (if it is a summary query) or forward the request to all other monitor nodes so that each can reply with the information stored locally.

Monitor nodes coordinate to keep updated versions of the map using a virtual ring (depicted in Figure 2.7). Each monitor node receiving the map updates it with the information it collected since the last visit and forwards the map to the next node in the ring.

2.3.6 Discussion

This section surveyed different algorithms for data dissemination in wireless networks. As depicted in Table 2.1, the algorithms cover a large number of combinations of assumptions concerning the capabilities of the devices, their movement, the supporting infrastructure and the predictability of the accesses to each data item.

The table shows some correlations among different algorithms. For example, the absence of proactive dissemination of replicas can be found in all algorithms developed for hybrid networks (those with a small number of producers) but also in others developed for wireless networks without infrastructure support. With the exception of Rumour Routing, all these algorithms create copies of the data items by storing them on the nodes performing the queries. Replication is justified, for example, to improve

Class	Protocol	Node Movement	Location Awareness	Access Prediction	Producers	Replica Refresh/ Leveraging	Geograph. Distr.	Message type	
								Proactive Dissem.	Query
Location Unaware	Simple Search	•			$<< n$			–	B
	Rumour Routing				n			rw	rw
	*-SAF	•		•	n	○		–	
	Aut. Gossipping	•		•	n	○		opp	–
Improved	Non-Unif				n		local/degrad.	B	–
	*-DAFN	•		•	n	○	coordination	–	
	*-DCG	•		•	n	○	coordination	–	
Trial & Error	7DS	•			$<< n$			–	p2p/B
	Sailhan et al.	•			$<< n$			–	p2p
	Double rulings		•		n		deterministic	p2p	p2p
	GLS	•	•		n	○	deterministic	p2p	p2p
Owner	CacheData	•			$<< n$		local	–	p2p
	DCS	•	•		n	•		p2p	p2p
Loc. Aware	CachePath	•			n			–	p2p
	R-DCS	•	•		n	•	deterministic	p2p	p2p

•: feature of the algorithm ○: implicitly provided
n: approx. all nodes in the network, $<< n$: small number of nodes
rw: random-walk, B: broadcast, p2p: point-to-point message, opp: opportunistic, –: not applicable

Table 2.1: Comparison of the data dissemination algorithms surveyed

availability (Hara, 2001) and to reduce of the number of transmissions (Sailhan & Issarny, 2003).

Replica refreshment can be used to ensure that an adequate number of replicas exist or that they are at the correct location. Some protocols offer this feature as a side effect of the protocol's normal operation. DCS and R-DCS are the only two protocols that implement specific measures to ensure that replicas are kept at the correct location. They send periodic messages, which are also used as keep alive messages, and that trigger a reconfiguration in case of failure.

The geographic distribution of the replicas should be interpreted differently for algorithms with and without proactive replica dissemination. Proactive algorithms distribute the replicas by the networked region. Non-proactive algorithms are mostly motivated by the savings in the resources of the nodes and aim to prevent that many replicas are stored in a neighborhood. The non-uniform information algorithm (Tilak *et al.*, 2003) uses a particular case of geographic dissemination because it is the only one that degrades the quality of the information with the distance to the source.

In the algorithms surveyed, proactive and geographic distribution of replicas is an exclusive of the algorithms that benefit of location information. Geographical distribution is motivated, for example, by the improved resilience to localised failures, like interference (Ghose *et al.*, 2003). The three algorithms providing this feature apply deterministic functions (GHTs) to determine the location of the data. Deterministic functions provide two advantages: $i)$ they avoid expensive coordination operations among the nodes to prevent redundancy, and $ii)$ they allow all operations to be implemented with point-to-point messages, which are expectedly less energy consuming than broadcasts. A requirement typically neglected in the literature and affecting these algorithms is that, to be useful, GHTs must first learn the topology of the entire network so that all data keys can be mapped on coordinates inside the network.

Two methods are used for distributing replicas in non proactive replication algorithms. In *CacheData* (Yin & Cao, 2006), the node selected for storing a replica is the node located at the injunction of different routes to the access point. This implicitly

prevents multiple nodes in a single but frequently used route from redundantly storing the same data item. Nodes in DAFN and DCG (Hara, 2001) periodically initiate rounds aimed specifically at negotiating the content of the cache of each node.

Point-to-point messages are the preferred method for message dissemination in algorithms where the location of at least one copy is known. It should be noted that point-to-point messages hide an implicit cost: these messages are typically delivered to some underlying routing protocol which is responsible for finding the route to the destination, or, in some cases, the most adequate next hop. To perform these operations, many routing protocols developed for networks where nodes are unaware of their location, like DSR (Johnson *et al.*, 2001) and AODV (Perkins & Royer, 1999) may occasionally use broadcasts.

Location unaware algorithms cannot address point-to-point messages to a specific node. However, even in this case, broadcast is used by a limited number of algorithms. Alternatives are to rely on sequences of random point-to-point messages within neighbours (random walks) or communicate only when new nodes become in range (Opportunistic Gossipping). Node discovery is typically performed using limited broadcasts.

2.4 Packet Dissemination

The goal of the packet dissemination module is to propagate messages according to the requirements of the query and dissemination algorithms. As shown in the previous section (see Table 2.1), the preferred packet dissemination algorithm is point-to-point what is typically provided by some routing protocol. Point-to-point routing protocols for multi-hop wireless networks have received considerable attention in the literature. The interested reader is referred to (Liu & Kaiser, 2003) for an extensive survey on the subject.

Random-walks, opportunistic communication and broadcast have also been referred in the previous section as alternative mechanisms used for message propaga-

tion. However, random walks and opportunistic communication are too restrictive and show to be useful only in a limited number of scenarios. Both are probabilistic, best effort mechanisms, that can not assure a timely message delivery or a successful delivery of the messages even if it is possible to establish a communication channel between two endpoints.

Broadcast does not suffer from the problems mentioned above but it is typically avoided due to its considerable costs when compared with point-to-point messages. However, due to the decentralised nature of Mobile Ad hoc NETworks (MANETs), many services, like the location of some resource, are typically implemented using broadcasts. Besides the data distribution protocols mentioned in the previous section, examples of other applications implemented using broadcasts are route discovery (Johnson *et al.*, 2001; Perkins & Royer, 1999), reputation systems (Perich *et al.*, 2004) and the distribution of code updates for sensors (Levis *et al.*, 2004).

This section focus on algorithms aimed to reduce the cost of broadcast operations. We note that reducing the cost of broadcast operations indirectly helps to improve the performance of the data distribution protocols that use point-to-point message delivery as many routing protocols use broadcast for determining the route for a node.

2.4.1 Broadcast Algorithms

The number of nodes required to retransmit a message so that it gets delivered to every participant depends of several factors like the transmission range of the devices, the location of the source, the size of the region covered by the nodes or their geographical distribution. Furthermore, these factors may be permanently changing, even between consecutive messages. In some cases, the underlying infrastructure may provide the tools to efficiently broadcast messages. This is the case, for example, of spanning trees provided by multi-cast routing protocols for ad hoc networks. However, in many cases, maintaining such structure is too costly or even impossible due to the high mobility of the nodes. This section will focus on alternatives that do not

```
1: msgSet← {}

2: upon event RECEIVE(m) do
3:     if m ∉ msgSet then
4:         DELIVER(m)
5:         msgSet ← msgSet ⋃{m}
6:         SEND(m)
7:     end if
8: end upon
```

Figure 2.8: Flooding algorithm

require an underlying infrastructure.

The most common implementation of broadcast is by flooding the network. In flooding, each node keeps a log of the messages recently received. When a node receives a message for the first time, the node adds the unique identification of the message to its log and retransmits it. Figure 2.8 shows an algorithm implementing flooding. Flooding creates a large number of redundant transmissions. Many nodes receive multiple copies of the message, each transmitted by a different node. Therefore, it wastes a non-negligible amount of bandwidth and power. Independently of the contribution of each retransmission, it consumes resources at the sender. Furthermore, receivers also spend a non-negligible amount of energy at the reception (Feeney & Nilsson, 2001) and CPU time to decide if the message should be discarded or retransmitted. Finally, flooding may generate collisions and contention because most of the Medium Access Control (MAC) protocols for ad hoc networks, like the one used in the IEEE 802.11 (IEEE 802.11, 1999), lack a coordination function able to completely avoid collisions of concurrent transmissions.

In a flooding procedure, the redundant reception of messages can not be completely avoided. Elsewhere (Tseng *et al.*, 2002), it was shown that a retransmission necessarily overlaps at least 39% of the region already covered by a previous transmission. The motivation for devising algorithms that improve the performance of flooding comes from noticing that this value increases significantly with the number of retransmissions listened by a node. The same paper shows that after listening to four trans-

Uses:
p: probability of retransmission

```
1: msgSet← {}

2: upon event RECEIVE(m) do
3:     if m ∉ msgSet then
4:         DELIVER(m)
5:         msgSet ← msgSet⋃{m}
6:         if RANDOM(0,1)< p then
7:             SEND(m)
8:         end if
9:     end if
10: end upon
```

Figure 2.9: GOSSIP1(p) algorithm

missions of the same message, a retransmission will overlap on average more than 95% the region covered by the previous ones.

2.4.2 Probabilistic Algorithms

A simple approach to reduce the number of retransmissions of a broadcast is to have each node to forward the first received copy of every message according to some probability. This section describes a family of probabilistic algorithms named GOSSIP, all presented in (Haas *et al.*, 2002).

The simpler implementation of a probabilistic algorithm is GOSSIP1(p), where each node, after receiving a message for the first time retransmit it with some probability $p \leq 1$. GOSSIP1(p) is depicted in Figure 2.9. It should be noted that for $p = 1$, GOSSIP1(p) becomes the flooding algorithm described previously.

GOSSIP1(p) presents some limitations that are better explained in a simplified model assuming that the algorithm progresses in rounds. In this model, the source of the message initiates the broadcast in round $r = 0$. A node receiving a message for the first time in round r will possibly transmit it in round $r + 1$. Nodes receive a transmission in the same round where it was transmitted.

An interesting property of probabilistic algorithms is that for the majority of the broadcasts, either a large or a small proportion of the nodes receive the broadcast. This property is usually referred as a bimodal behaviour and was studied in the context of probabilistic algorithms for broadcasting in wireless networks in (Haas *et al.*, 2002). Let t_r be the number of nodes that retransmit a message in round r and n_r the number of nodes that received it for the first time in the same round. In GOSSIP1(p), t_{r+1} is expected to be close to $n_r \times p < n_r$. Given that p is a constant, the probability of having some nodes to retransmit the message in round $r + 1$, i.e. $t_{r+1} > 0$ increases with n_r which in turn depends of t_r. Therefore, the broadcast is more likely to be delivered to every node if, in all rounds, t_r is high. The bimodal behaviour of GOSSIP1(p) comes from the broadcast being initiated by a single node, so $t_0 = 1$. If the number of neighbouring nodes (n_r) or the probability of retransmission (p) is also small, there is the risk that after a few rounds, the broadcast dies because $n_r \times p \approx 0$. However, with some probability, it is also possible that n_r increases rapidly in the first few rounds, ensuring the delivery of the message to a large proportion of the nodes.

To avoid the bimodal behaviour one should therefore improve the operation of the first few rounds. Because the number of neighbours (n_r) does not depend of the algorithm, an higher t_r can only be achieved with an higher probability of retransmission p. GOSSIP1(p, k) extends GOSSIP1(p) by setting $p = 1$ in the first k rounds of a broadcast. In the remaining rounds, nodes retransmit with probability $p < 1$. The algorithm is depicted in Figure 2.10.

Experimental results confirmed that for $p > 0.7$ and $k = 4$, GOSSIP1(p, k) does not exhibit bimodal behaviour, meaning that only approximately 70% of the nodes are required to propagate a broadcast to deliver it to a large proportion of the nodes with high probability. For lower values of p, the delivery ratio of GOSSIP1(p, k) decays rapidly. The problem was attributed to the irregular distribution of the nodes. In regions with low density, the conditions that motivated GOSSIP1(p, k) may appear as a result of a small number of nodes listening to the message for the first time, even after a large number of rounds.

Uses:
p: probability of retransmission
k: min number of hops

```
1: msgSet← {}

2: upon event RECEIVE({m,hops}) do
3:     if m ∉ msgSet then
4:         DELIVER(m)
5:         msgSet ← msgSet⋃{m}
6:         if hops < k ∨ RANDOM(0,1) < p then
7:             SEND({m,hops+1})
8:         end if
9:     end if
10: end upon
```

Figure 2.10: GOSSIP1(p, k) algorithm

The problem can be addressed by monitoring the network so that an higher probability of retransmission is applied in regions where the node density is low. GOSSIP2(p_1, k, p_2, n) is depicted in Figure 2.11. In this algorithm, p_1 and k retain the meaning described for GOSSIP1(p, k). $p_2 > p_1$ specifies the probability of retransmission for nodes receiving the message from a node with less than n neighbours. Simulations showed that for the same scenarios, GOSSIP1(0.8,4) presented a comparable performance to GOSSIP2(0.6,4,1,6). However, GOSSIP2 required on average less 13% transmissions than GOSSIP1.

An undesirable requirement of GOSSIP2 is that nodes must be aware of their neighbours. Due to node movement, the number of neighbours can only be known if all nodes periodically transmit some control message to advertise their presence. However, the periodic sending of messages should be avoided in wireless networks because it increases bandwidth and battery consumption. RAPID (Drabkin *et al.*, 2006) reduces this burden by having nodes to periodically broadcast an heartbeat message only if no message has been sent for a predefined period of time. RAPID algorithm is outlined in Figure 2.12.

In contrast with the previous algorithms, in RAPID the probability of retransmis-

Uses:
k: min number of hops
n: min number of neighbours
p_1: probability of retransmission in the regular case
p_2: probability of retransmission in special case

```
1: msgSet← {}

2: upon event RECEIVE({m,hops,nn}) do
3:     if m ∉ msgSet then
4:         DELIVER(m)
5:         msgSet ← msgSet⋃{m}
6:         rnd←RANDOM(0,1)
7:         if hops < k ∨ (nn < n ∧ rnd < p2) ∨ (nn ≥ n ∧ rnd < p1) then
8:             myNeigh←COUNTNEIGHBOURS
9:             SEND({m,hops+1,myNeigh})
10:        end if
11:    end if
12: end upon
```

Figure 2.11: GOSSIP2(p_1, k, p_2, n) algorithm

Uses:
NNEIGH: current estimated number of neighbours
β: reliability factor

```
1: msgSet← {}

2: upon event RECEIVE(m) do
3:     if m ∉ msgSet then
4:         DELIVER(m)
5:         msgSet ← msgSet⋃{m}
6:         p← min{1, β/NNEIGH}
7:         if RANDOM(0,1)< p then
8:             SEND(m)
9:         end if
10:    end if
11: end upon
```

Figure 2.12: RAPID algorithm

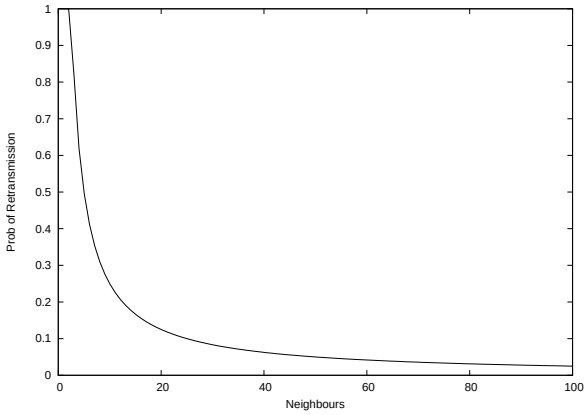


Figure 2.13: Probability adaptation function in RAPID ($\beta = 2.5$)

sion is independently defined by each node. Nodes retransmit with a dynamic probability given by its number of neighbours and a reliability factor β related with the number of nodes that should perform a retransmission in a one hop neighbourhood. As Figure 2.13 shows, the probability of retransmission decreases with the number of neighbours, to prevent an excessive number of redundant retransmissions to occur.

2.4.3 Counter-based Algorithms

Results from GOSSIP2(p_1, k, p_2, n) and RAPID show that probabilistic algorithms do not adapt well to variations in node density because the number of retransmissions is proportional to the number of nodes in the region. However, even without the exchange of dedicated control messages, it is possible to retrieve some information from the network. Counter-based algorithms try to improve the efficiency of the broadcast by requiring each node to check the usefulness of their retransmission. The test is used both for preventing, to some degree, redundant retransmissions and to ensure a sufficient number of retransmissions.

Enhanced RAPID (Drabkin *et al.*, 2006) is a counter-based three step algorithm. As depicted in Figure 2.14, after receiving a message for the first time, the algorithm waits for a small random delay, while monitoring the network. The retransmission is cancelled either by listening a retransmission by another node or by the same dynamic

Uses:
NNEIGH: current estimated number of neighbours
β: reliability factor
d_s: small maximum timer delay
d_l: large maximum timer delay

1: msgSet$\leftarrow$ {}

2: **upon event** RECEIVE(msg) **do**
3: **if** $msg \notin$ msgSet **then**
4: DELIVER(msg)
5: msgSet $\leftarrow$ msgSet$\bigcup\{msg\}$
6: d$\leftarrow$RANDOM(0,d_s)
7: p$\leftarrow \min\left\{1, \frac{\beta}{\text{NNEIGH}}\right\}$
8: SETALARM(d,{msg,p})
9: **else**
10: CANCELALARM(msg)
11: **end if**
12: **end upon**

13: **upon event** ALARM({msg,p}) **do**
14: **if** RANDOM(0,1)<p **then**
15: SEND(msg)
16: **else**
17: d$\leftarrow$RANDOM(0,d_l)
18: SETALARM(d,{msg,1.0})
19: **end if**
20: **end upon**

Figure 2.14: Enhanced RAPID algorithm

probability function used in RAPID (see Section 2.4.2). However, a node that decides not to retransmit continues to monitor the network for an additional random time. This second monitoring period has a larger interval. The node will retransmit with probability 1 if it does not listen at least one retransmission of the message during this period.

Like in GOSSIP1(p, k), in GOSSIP3(p, k, m) when a node receives a message for the first time, it will retransmit it with probability p, with $p = 1$ if the source is less than k hops away. However, each node that decided not to retransmit is required to confirm that a sufficient number (m) of retransmissions occurs so that the broadcast does not

Uses:
k: min number of hops
m: min number of retransmissions
p: probability of retransmission
d: timer delay

```
1: msgSet← {}

2: upon event RECEIVE({msg,hops}) do
3:     if msg ∉ msgSet then
4:         DELIVER(msg)
5:         msgSet ← msgSet⋃{msg}
6:         if hops < k ∨ RANDOM(0,1) < p then
7:             SEND({msg,hops+1})
8:         else
9:             SETALARM(d,{msg,hops})
10:        end if
11:    else
12:        C_msg ← C_msg + 1
13:    end if
14: end upon

15: upon event ALARM({msg,hops}) do
16:    if C_msg < m then
17:        SEND({msg,hops+1})
18:    end if
19: end upon
```

Figure 2.15: GOSSIP3(p, k, m) algorithm

die. The node will retransmit if it does not listen to m retransmissions within a short period of time. GOSSIP3(p, k, m) is depicted in Figure 2.15. The late retransmission reduces the probability of the broadcast to die because of an inadequate selection of the probability p. Experiments performed by the authors showed that in a particular scenario, GOSSIP3(0.65,4,1) outperformed slightly GOSSIP1(0.75,4) although requiring 8% less transmissions.

The "counter-based scheme" (Tseng *et al.*, 2002) is presented in Figure 2.16. In this algorithm, a node receiving a message for the first time waits for a random time t before retransmitting. During the wait period, the node counts the number of retransmissions it listens. The node will retransmit if, when the timer expires, the number of

Uses:
m: min number of retransmissions
maxDelay: maximum delay

```
1: msgSet← {}

2: upon event RECEIVE(msg) do
3:     if msg ∉ msgSet then
4:         DELIVER(msg)
5:         msgSet ← msgSet⋃{msg}
6:         delay←RANDOM(0,maxDelay)
7:         SETALARM(delay,msg)
8:     else
9:         C_msg ← C_msg + 1
10:    end if
11: end upon

12: upon event ALARM(msg) do
13:    if C_msg < m then
14:        SEND(msg)
15:    end if
16: end upon
```

Figure 2.16: Counter-based scheme

retransmissions listened is below a predefined threshold value.

Like in GOSSIP3(p, k, m), nodes in the "counter-based scheme" count the number of retransmissions listened to decide if they should retransmit. However, in the "counter-based scheme" the transmissions are distributed over some time period. This may be advantageous for wireless networks as it reduces the number of collisions in multiple access MAC protocols like the IEEE 802.11 (IEEE 802.11, 1999). These algorithms differ also in the expected number of retransmissions, which in GOSSIP3(p, k, m) depends of the network density but is a fixed constant in the "counter-based scheme".

In the "Hop Count-Aided Broadcasting" (HCAB) algorithm (Huang *et al.*, 2006), messages carry an hop count field (HC) set to zero by the source and incremented at every retransmission. As depicted in Figure 2.17, nodes receiving the message for the first time start a random timer and record the value of HC. The message will be retrans-

Uses:
maxDelay: maximum delay

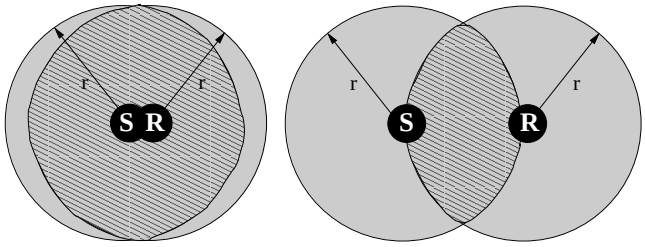
```
1: msgSet← {}

2: upon event RECEIVE({msg,HC}) do
3:     if msg ∉ msgSet then
4:         DELIVER(msg)
5:         msgSet ← msgSet⋃{msg}
6:         HC_msg ← HC
7:         delay←RANDOM(0,maxDelay)
8:         SETALARM(delay,msg)
9:     else
10:        if HC > HC_msg then
11:            CANCELALARM(msg)
12:        end if
13:    end if
14: end upon

15: upon event ALARM(msg) do
16:    SEND({msg,HC_msg + 1})
17: end upon
```

Figure 2.17: Hop Count-Aided Broadcasting algorithm

mitted by the node if, when the timer expires, no message with an HC higher than the first was received. Implicitly, nodes in HCAB try to assure that each transmission is forwarded by some node, hopefully covering additional regions of the network.

2.4.4 Distance-Aware Algorithms

When nodes are equipped with omni-directional antennas, each transmission performed by a node S is propagated in every direction. The signal strength fades with the distance to the source by a factor that depends of different physical aspects, such as obstacles traversed by the signal. The transmission of a message by S will be received by a node R if at R's location the signal strength is above the minimum receiving threshold of R's interface. A simplified theoretical model commonly used in the literature assumes that nodes are homogeneous. That is, all nodes transmit with the same power



(a) Small additional coverage (b) Maximum additional coverage

Figure 2.18: Overlap of the coverage of the transmission range between two nodes

and have an equal minimum receiving threshold. It is also assumed that there is a monotonously decreasing function that maps the distance to the source in the signal strength.

In this model, the receivers of any message transmitted by some node S are those lying within a circle centred at S and with a radius that depends of the transmission power, the fading of the signal and the minimal reception threshold. If all these factors are known, it is possible to map the power with which each message is received by some node on its distance to the source. This section uses the model above to present some algorithms that use the distance between the nodes as a criteria for the selection of the nodes that will retransmit a message.

A node R that retransmits a message must have received it from some node S. Therefore, R must be within transmission range of S. Since it is assumed that nodes use omni-directional antennas, part of the coverage provided by the retransmission of R will overlap with the region previously covered by the transmission from S. Figure 2.18 illustrates this by presenting two nodes S and R and their transmission ranges as circles of radius r.

For the theoretical model presented above, it was shown in (Tseng *et al.*, 2002) that the additional coverage provided by a retransmission may increase between 0% and 61% the coverage of the previous retransmission. Figures 2.18(a) and 2.18(b) show the additional coverage provided with a retransmission by R and the overlapped region. As it can be seen, the gains depend of the location of the node performing the

Uses:
maxPwr: threshold transmission power
maxDelay: maximum delay

```
1: msgSet← {}

2: upon event RECEIVE(msg,P) do
3:     if msg ∉ msgSet then
4:         DELIVER(msg)
5:         msgSet ← msgSet ⋃{msg}
6:         P_msg ← P
7:         delay←RANDOM(0,maxDelay)
8:         SETALARM(delay,msg)
9:     else
10:        P_msg ← max {P_msg, P}
11:    end if
12: end upon

13: upon event ALARM(msg) do
14:    if P_msg < maxPwr then
15:        SEND(msg)
16:    end if
17: end upon
```

Figure 2.19: Distance-based scheme

retransmission: a node closer to the source (Figure 2.18(a)) contributes with a smaller additional coverage and a node more distant to the source (Figure 2.18(b)) with an higher additional coverage.

Preventing nodes too close to a previous sender (and therefore providing a negligible contribution to the coverage) from retransmitting is addressed in the "distance-based scheme" described in (Tseng *et al.*, 2002) and depicted in Figure 2.19. After receiving a message, nodes set a timer for a random time. During this period, nodes compute M, the maximum received signal strength (RSSI) of the original message and of all copies. A node will retransmit a message only if when the timer expires M does not exceed a predefined threshold.

Given that in the wireless media the signal strength decays with the distance to the sender, it can also be said that this model prevents nodes within a minimal dis-

tance from a previous sender from forwarding the message. Like in the counter-based algorithms presented in Section 2.4.3, the "distance-based scheme" adapts well to different network densities: the number of transmissions does not grow with the average number of neighbours.

In the "Self-Adaptive Probability Broadcasting" (SAPB) (Huang *et al.*, 2006), the probability of a node to retransmit a message is given by an aggregation of three metrics. Figure 2.20 shows that like in many other algorithms described above, each node sets a timer for a random time after receiving a message for the first time. When the timer expires, the probability of retransmission will be given by:

$$P = K.f(n).g(S_x) \tag{2.2}$$

where K is a constant defined so that $0 < P \leq 1$, n is the number of transmissions received and S_x is the maximum of the Received Signal Strength Indicator (RSSI) of all copies of the message received during the time interval. Functions $f(n)$ and $g(S_x)$ should be monotonously decreasing so that the probability of retransmission decreases with the number of retransmissions listened and with their RSSI. SAPB combines the three major topics that have been addressed in the other proposals discussed: a random selection of the nodes, a bound on the number of retransmissions in the neighbourhood of each node and a bound on the distance to the source of previous transmissions.

2.4.5 Discussion

A summary of the key characteristics of each algorithm discussed in this section is presented on Table 2.2. The column "bounded transmissions" shows which algorithms impose an absolute limit on the number of retransmissions in the neighbourhood of each node. This is an important aspect as it evaluates the adaptation of the algorithms to different node densities. Of the surveyed algorithms, only the Enhanced RAPID and "Counter-based scheme" prevent a node from retransmitting if it listens

Uses:
K: constant multiplicative factor
f: function over the number of retransmissions
g: function over maximum transmission power
maxDelay: maximum delay

```
1:  msgSet← {}

2:  upon event RECEIVE(msg,P) do
3:      if msg ∉ msgSet then
4:          DELIVER(msg)
5:          msgSet ← msgSet⋃{msg}
6:          N_msg ← 1
7:          S_msg ← P
8:          delay←RANDOM(0,maxDelay)
9:          SETALARM(delay,msg)
10:     else
11:         N_msg ← N_msg + 1
12:         S_msg ← max {S_msg, P}
13:     end if
14: end upon

15: upon event ALARM(msg) do
16:     p ← K.f(N_msg).g(S_msg)
17:     if RANDOM(0,1) < p then
18:         SEND(msg)
19:     end if
20: end upon
```

Figure 2.20: Self-Adaptive Probability Broadcasting Algorithm

to more than a predefined number of retransmissions. The "distance-based scheme" indirectly implies an upper bound on the number of retransmissions because a transmission from a node implicitly prohibits those closer than a predefined distance from retransmitting. A lower bound can be defined as a configuration parameter (this is the case in GOSSIP3(p, k, m) and "counter-based scheme") or be implicit (like in HCAB) by having the nodes to retransmit if, when its timer expires, the retransmissions listened do not satisfy some condition.

The "source aware" column shows the algorithms that identify a potential threat to the broadcast at the early stages of the dissemination. Indirectly, this is also addressed

	Algorithm	Bounded transmissions		Source aware	Progress	Collision avoidance	Random selection of nodes
		Upper	Lower				
Probab	GOSSIP1(p)						•
	GOSSIP1(p, k)			•			•
	GOSSIP2(p_1, k, p_2, n)			•			•
	RAPID						•
Count	Enhanced RAPID	•	•	○		•	•
	GOSSIP3(p, k, m)		•	•			•
	Count-based	•	•	○		•	•
	HCAB		○	○	•	•	•
Dist	Distance-based	○			•	•	•
	SAPB					•	•

•: feature of the algorithm ○: implicitly provided

Table 2.2: Comparison of the broadcast algorithms surveyed

by all algorithms imposing a lower bound on the number of retransmissions because this bound is also applicable in the neighbourhood of the source.

The "Distance-based scheme" and HCAB are the two algorithms that use the expectations of additional coverage in the selection of the nodes that will retransmit. The former by estimating the distance to the source of the previous retransmissions while the later uses a counter that compares the original reception of the message at the node with other retransmissions performed in the neighbourhood.

Algorithms marked in the "collision avoidance" column are those that incorporate a mechanism to separate the retransmissions performed by the nodes, using a random delay locally decided at each node. This separation is fundamental in some medium access protocols to prevent contention and an undesirable number of collisions resulting from the simultaneous retransmission of the message by multiple nodes in a neighbourhood as a result of the reception of the same message. It should be noted that any of the other algorithms can be easily adapted to address this issue by applying some random delay before performing the retransmission of the message.

Although with some variations, all the algorithms surveyed use randomisation to select the nodes that will perform the retransmission. The random selection may be explicit, like in GOSSIP algorithms where nodes peek a random number to decide if they

retransmit, or can be implicit. In these, after receiving a message, each node chooses a random delay, and decides to retransmit at the end of the delay if some criteria has not been satisfied by previous retransmissions. Therefore, a node that randomly selected a smaller delay is more likely to be required to retransmit.

Contrary to the remaining, RAPID, Enhanced RAPID and GOSSIP2($p1, k, p2, m$) adapt their probability of retransmission to the local node density. The former two using a linear variation and the later with a fixed threshold. This is an interesting feature, which, although not explicitly bounding the number of retransmissions attempts to adapt to the network conditions in the neighbourhood of each node.

HCAB and Enhanced RAPID prevent all nodes listening to one retransmission from sending. This is a restrictive threshold that is based on the results presented in (Tseng *et al.*, 2002) that concluded that very limited additional coverage is gained with the second retransmission. As Section 3.3 will show, simulations of HCAB do not provide a delivery ratio comparable with other algorithms using an higher threshold. However, it should be noted that RAPID runs in the background a gossip algorithm allowing nodes to learn and retrieve messages they did not receive with the dissemination algorithm.

2.5 Summary

Data distribution protocols can be decomposed in different modules. The decomposition proposed in this chapter uses three modules: local storage space management, data management and packet dissemination. From the survey of the related work for each of these topics we extract the following conclusions.

Cache management policies that have proven to be successful in other domains are not the most adequate to wireless networks. However, no main trend in customised solutions has been found. Apparently, the most adequate policy is likely to depend of the application scenario and of the particularities of the wireless networking environment.

Data management uses replication algorithms to improve resilience to faults like disconnection, interference and device failure. In addition, replication contributes to reduce the number of messages and bandwidth required for data retrieval.

Two major types of replication algorithms have been found. In proactive replication, data is pushed to a subset of the nodes in the network, regardless of their interest on that particular item. In non-proactive algorithms, nodes store the data they have previously requested or forwarded and make it available to other nodes.

A particular variant tries to evenly distribute the replicas over the entire networked region. It was argued that geographically distribution of the replicas provides the biggest tolerance to failures. In scenarios where the access pattern of the nodes to the data is not correlated with their location, geographical distribution can equally contribute to reduce the access latency and the number of messages required to retrieve the data. To the extent of our knowledge, proactive geographical distribution of the data has only been pursued in the context of wireless networks where the nodes are aware of their location. In networks where devices do not have location information available, geographical distribution has been addressed in a limited scope, where neighbouring nodes cooperate to prevent an excessive redundancy of the data.

To replicate data, nodes exchange messages. The survey identified four message passing mechanisms: point-to-point, random walk, opportunistic gossipping and broadcast. A correlation between point-to-point message passing and dissemination algorithms were nodes are aware of the location of at least one copy of the items was found. In the remaining algorithms no clear trend can be identified.

Broadcast has been widely used by protocols for wireless networks and is one of the cornerstones of different point-to-point routing protocols for MANETs. Therefore, a reduction on the number of messages transmitted on each broadcast contributes to the reduction of the cost of different data dissemination protocols, including some of those using point-to-point messages. Different proposals have recently emerged to reduce the cost of broadcast operations. The survey showed that although implementing different policies, all randomly select some of the neighbours of the source of the

previous transmission.

3

A Power-Aware Broadcasting Algorithm

Section 2.4 presented different algorithms for message dissemination that generate a much smaller number of retransmissions than flooding. The goal of these algorithms is to devise, in run-time, a subset of participants that are required to retransmit so that every node in the network receives at least one copy of the message. Due to the movement of the nodes and to the different location of the sources, this subset may be different for each message.

An important factor to improve the performance of broadcast algorithms for MANETs and which has been previously neglected in the literature is the location of the nodes performing a retransmission with respect to the source of the previous transmissions. We provide a simple case study supporting this claim.

Figure 3.1 represents a region of a MANET with a source S of a broadcast message and its neighbours. It is assumed that the network follows the simplified networking model described in Section 2.4.4. In brief, that all nodes transmit with the same power and are capable of receiving a message if the signal strength at the receiver is above some minimum threshold. The figure assumes that the transmission range of all nodes is r and that the minimal power threshold used by the "distance-based scheme" described in Section 2.4.4 is r'. The transmission range of nodes S, B, C and F is represented by circles.

The region covered by a retransmission of a broadcast initiated by S grows with the distance of the retransmitting node to S. Elsewhere (Tseng *et al.*, 2002), it was shown

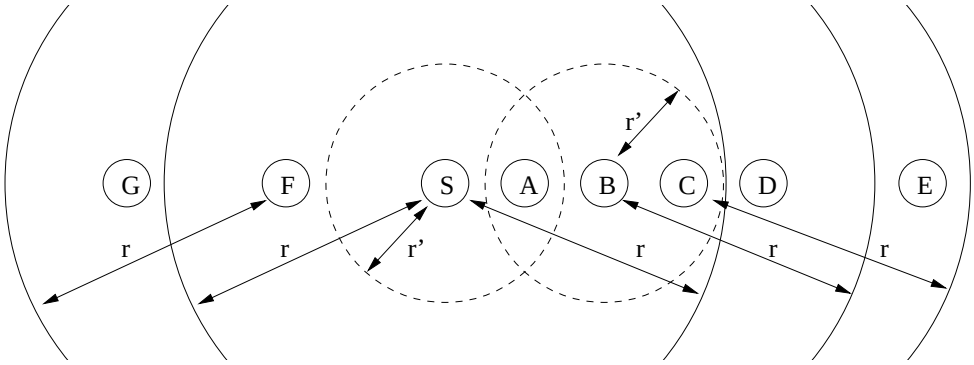


Figure 3.1: Deployment and transmission range of some nodes

that if all nodes transmit with equal power, a retransmission can increase between 0% and 61% the space covered by the previous transmission, depending on the location of the retransmitting node. In this example, the most efficient retransmissions would be performed by nodes C and F. The minimal number of transmissions required to deliver a message broadcasted by node S to all other nodes is three: the message is delivered to nodes A, B, C and F by the first transmission and nodes C and F could retransmit to deliver it respectively to D and E, and G. We emphasise that, in runtime, nodes do not have access to the information required for following the same rationale presented above.

Section 2.4.5 showed that all algorithms surveyed in the related work randomly select the nodes that perform a retransmission. Applying these algorithms to the scenario depicted in Figure 3.1 shows that for all algorithms the random selection of the nodes may result in at least four, instead of the optimal three retransmissions. With exception of the "distance-based scheme", all algorithms will randomly select one of the nodes A, B or C to perform the first retransmission. The transmission by node A, for example, satisfies the conditions of all these algorithms but provides a minimal contribution to increase the number of nodes that received the message. Depending on the algorithm and on its configuration parameters, a retransmission by node A would either: i) require a retransmission by node D in order to have the message delivered to node E, or ii) be useless, if nodes B or C retransmit as well. Therefore, if node A is selected, the message can not be delivered to every node in the figure with less than

four transmissions.

The "Distance-based scheme" is configured with a fixed threshold. This threshold can not be too close to the maximum transmission range to prevent the elimination of good candidates for retransmission in the cases where nodes are not uniformly distributed. In the example of Figure 3.1, the threshold is represented using dashed circles with radius r' and excludes node A from the list of candidates for retransmission of a message sent by S. However, the "distance-based scheme" does not rank differently the nodes located outside the circle defined by r'. After the transmission by node S, both nodes B and C are candidates to retransmit. This algorithm would require four transmissions if the timer of node B expires before the timer of node C. It should be noted that the retransmission by B would even inhibit node C from retransmitting.

This chapter proposes a novel algorithm to reduce the resources consumed by the nodes and the bandwidth required by broadcasts in MANETs. This is a challenging problem because we want to minimise the signalling overhead and we do not want to enforce the use of special hardware (e.g. nodes are not required to use a GPS receiver to become aware of their location). Like in the "distance-aware scheme" (Tseng *et al.*, 2002), our algorithm only assumes that nodes are able to retrieve the power with which each message is received. The algorithm, named Power-Aware Message Propagation Algorithm (Pampa) is distinguished from the previous proposals by removing some of the randomness associated with the decision on the nodes that will retransmit a message. Simulation results, to be presented in Section 3.3 experimentally confirm that the selection of the nodes can be improved to prevent the occurrence of these cases.

3.1 Pampa

As mentioned before, the increment in the coverage provided by a retransmitting node may vary between 0% and 61%. This ratio grows with the distance to the source. The key idea of Pampa is to run a fully distributed algorithm that makes nodes more distant to the source to retransmit first, instead of relying on the random selection of

Uses:
m: min number of retransmissions
delay(P): function over the signal strength P

```
1:  msgSet← {}

2:  upon event RECEIVE(msg,P) do
3:      if msg ∉ msgSet then
4:          DELIVER(msg)
5:          msgSet ← msgSet ⋃{msg}
6:          C_msg ← 0
7:          del← delay(P)
8:          SETALARM(del,msg)
9:      else
10:         C_msg ← C_msg + 1
11:     end if
12: end upon

13: upon event ALARM(msg) do
14:     if C_msg < m then
15:         SEND(msg)
16:     end if
17: end upon
```

Figure 3.2: Pampa algorithm

the nodes. In an ideal environment, and independently of the node's distribution, this would ensure that each retransmission would be providing the highest additional coverage possible, what would be achieved by the other algorithms only in the fraction of the cases where the more distant node is randomly selected for retransmission.

The algorithm of Pampa is depicted in Figure 3.2. When receiving a message for the first time, the algorithm stores the message and sets a timer for a delay d, given by a function *delay* to be addressed later. During this period, the node counts the number of retransmissions listened. The message is transmitted if when the timer expires, the node did not listen to a sufficient number of retransmissions.

Central to Pampa is a function *delay* which gets the received signal strength (RSSI) of a transmission and outputs a delay. This function is expected to map an increasing distance to the source (corresponding to a smaller RSSI) in a smaller return value. Be-

cause the RSSI will be different for each node, the function *delay* will return a different value for each node receiving the same transmission. Implicitly, the function orders the nodes according to the distance to the source, with nodes more distant to the source expiring their timers first. It should be noted that the function is fully distributed: the algorithm is triggered exclusively by the transmission of the broadcast message and it does not require any coordination between the nodes. Like in the "counter-based scheme" (Tseng *et al.*, 2002), the algorithm prevents excessive redundancy by having nodes to count the number of retransmissions listened. However, Pampa bias the delay such that the nodes refraining from transmitting are usually those that are closer to the source.

3.1.1 *Delay* Assignment

The selection of a good *delay* function is key to the performance of Pampa. We estimate that a *delay* function that varies linearly with the distance to the source would provide the best results. However, such a function would require complex computations unsuitable to be performed by mobile devices for each received message.

In our tests, we defined two simpler *delay* functions that multiply the RSSI by a constant k to return the number of seconds that the node should wait before retransmitting. Analysis of an adequate value for k was performed using the Two Ray Ground propagation model as defined in the *ns–2* network simulator version 2.28. We have found 300×10^6 to be an adequate value for obtaining distinct wait times for nodes close to each other. The behaviour of the function $\mathit{delay}(p) = 300 \times 10^6 \times p$ for the Two Ray Ground propagation model is presented in Figure 3.3.

As expected, the function follows the logarithmic decay of the reception power of a message. For short distances, the function returns excessively large delay values. However, nodes at these distances from the source have a large probability of not being required to retransmit. A careful implementation of the algorithm can free the resources consumed by the messages on hold as soon as the threshold number of re-

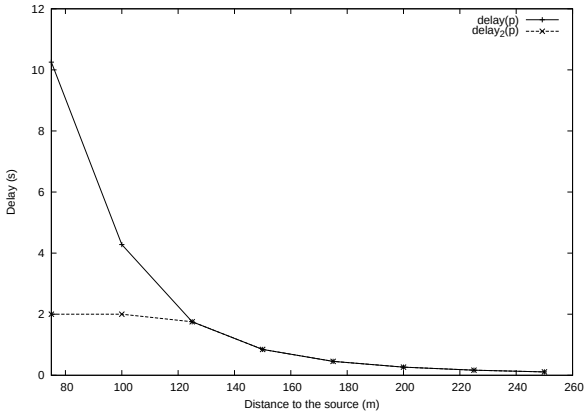


Figure 3.3: Function *delay*

transmissions is listened. Section 3.3.2.1 will show that these excessive delays do not significantly influence the propagation of the message.

However, to better assert its impact in the performance of Pampa, a second delay function $delay_2$ was defined. This function, also depicted in Figure 3.3 defines a maximum delay of two seconds with an additional random variation of 500ms to prevent an excessive number of collisions.

3.2 Comparison with Related Work

In Pampa, the instant at which each node forwards a message is locally determined from its distance to the sender. In the absence of abnormal effects on the signal propagation, Pampa assures that the first nodes to perform a retransmission are those that provide the higher possible additional coverage. In the example presented in Figure 3.1, node C would be the first to retransmit, delivering the message to both nodes D and E. Although slightly later, node F would also be required to retransmit and therefore, guarantee the coverage of node G. In this example, Pampa reduces the probability of requiring four instead of the minimal three transmissions for delivering the message to every node.

Table 3.1 extends to Pampa the comparison of the related work of Table 2.2 (see

	Algorithm	Bounded transmissions		Source aware	Progress	Collision avoidance	Random selection of nodes
		upper	lower				
Probab	GOSSIP1(p)						•
	GOSSIP1(p, k)			•			•
	GOSSIP2(p_1, k, p_2, n)			•			•
	RAPID						•
Count	Enhanced RAPID	•	•	○		•	•
	GOSSIP3(p, k, m)		•	•			•
	Counter-based	•	•	○		•	•
	HCAB		○	○	•	•	•
Dist	Distance-based	○			•	•	•
	SAPB					•	•
	Pampa	•	•	○	•	•	

•: feature of the algorithm ○: implicitly provided

Table 3.1: Comparison of the broadcast algorithms surveyed with Pampa

page 52). Of those compared, Pampa is the only algorithm that does not randomly select the nodes performing the retransmission, selecting instead, those that are more distant to the original source of the transmission. Also, Pampa presents an interesting combination of features from counter-based and distance-aware algorithms. It imposes an upper and lower bound on the number of retransmissions, therefore, being independent of the node density. The selection of the nodes that are more distant from previous retransmissions tries to extend the coverage to more distant locations, thus putting an effort on the selection of the nodes that favour the progress of the broadcast.

3.3 Simulation Results

We have implemented the "counter-based scheme" (Tseng *et al.*, 2002), "distance-based scheme" (Tseng *et al.*, 2002), HCAB (Huang *et al.*, 2006) and Pampa algorithms in the *ns–2* network simulator v. 2.28. Each implementation was tested with different parameters. For the "counter-based" and Pampa, we tested different thresholds for the number of times that the same message is received after which a retransmission is discarded. This threshold is shown as the number following the name of each algorithm

in the captions of the figures. In addition, Pampa was tested with the two *delay* functions introduced in Section 3.1.1. In the figures that follow, Pampa with the unbound *delay* function is identified as "Pampa" while Pampa using function $delay_2$ is depicted as "Pampa$_2$". The "distance-based scheme" was tested with different threshold distances, identified (in meters) in the captions of the figures. HCAB does not have any configuration parameters.

Each algorithm had some parameters immutable for all simulations. The maximum random delay used by the "counter-based scheme", "distance-based scheme" and HCAB was set to $0.75s$. As described in Section 3.1.1, both Pampa and Pampa$_2$ multiply the Received Signal Strength Indication (RSSI) by 300×10^6. The maximum delay in Pampa$_2$ is $2s$ and the additional jitter is randomly selected between 0 and $500ms$.

3.3.1 Test-bed

Different node densities have been experimented by changing the size of the simulated space. Eight simulated regions were tested: $250m \times 250m, 500m \times 250m, 1000m \times 500m, 1000m \times 1000m, 2000m \times 1000m, 1500m \times 1500m, 2000m \times 1500m$ and $2000m \times 2000m$, providing ratios between $625m^2$/node and $40000m^2$/node. All simulations are run with 100 nodes. Nodes were configured to emulate a 914MHz Lucent WaveLAN DSSS radio interface running an IEEE802.11 protocol at 2Mb/s. Network cards present a transmission range of 250m using the Two Ray Ground propagation model.

Each test combines different traffic sources and movement of the nodes. Traffic in each test is composed of 1000 messages, generated at a pace of one message per second. The source of each message is selected at random. The size of each message is 1000 bytes.

At the beginning of each test, nodes are uniformly deployed over the simulated region. In one set of tests, nodes do not move for the entire duration of the simulation. These tests have been named "Speed 0". In the remaining two sets nodes move using

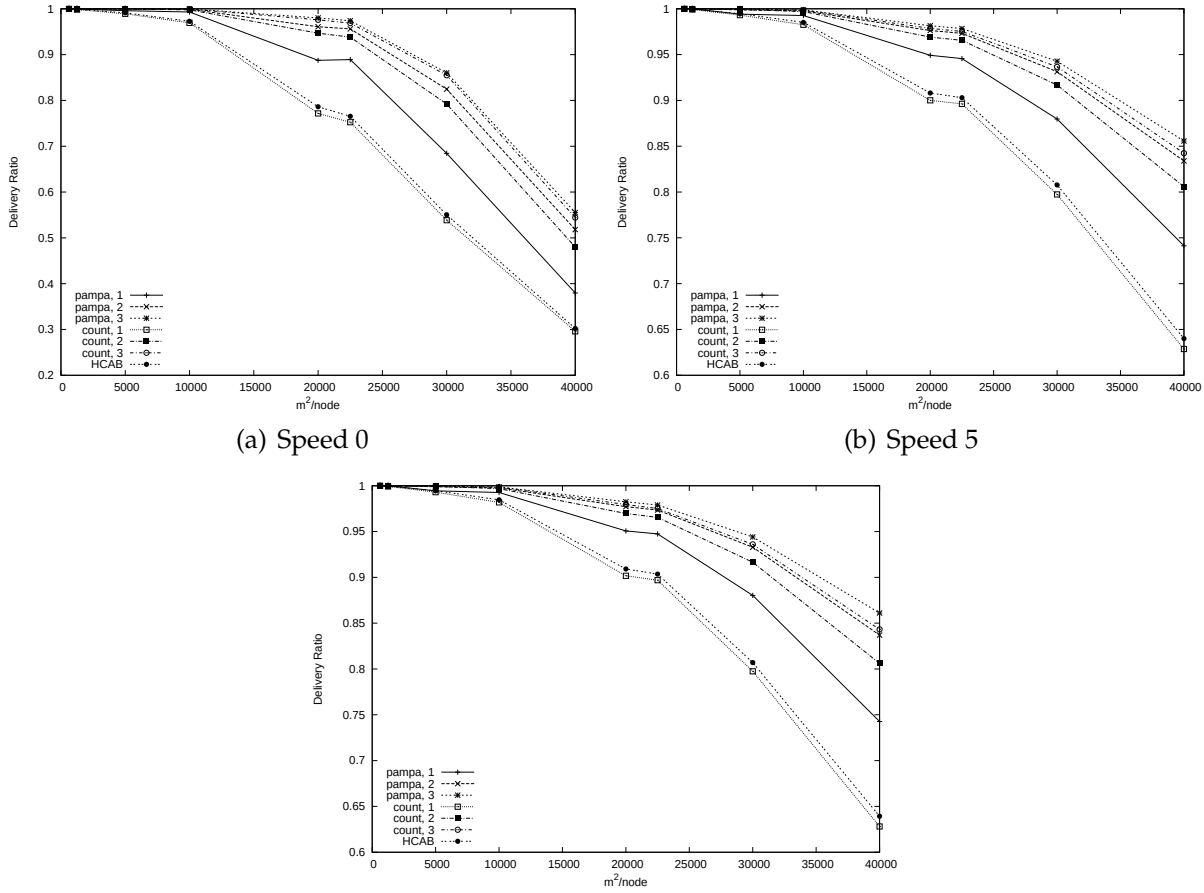
the Random Waypoint Movement Model (Johnson & Maltz, 1996). In this movement model nodes select a random location in the simulated region and move in a straight line to it at a speed randomly defined between a minimum and a maximum value. When the location is reached, nodes stop for a predefined amount of time and initiate a new cycle by selecting another random location. In our tests, minimum and maximum speeds are respectively $4m/s$ and $6m/s$ for one test set and $9m/s$ and $11m/s$ for the other. Nodes never stop. These sets have been named respectively "Speed 5" and "Speed 10".

For each simulated region and speed, 100 different tests were defined and experimented with each of the algorithms. Each point in the figures presented below averages the result of the 100 runs. The evaluation will focus on four metrics: the proportion of nodes receiving each message; the number of transmissions used; the latency of the delivery and the number of hops required to deliver the message to the nodes.

3.3.2 Delivery Ratio

This section evaluates the efficiency of the algorithms. The metric used is the average of the proportion of the nodes that receive each message. Figure 3.4 compares the performance of the counter based algorithms with Pampa. A comparison between the three plots of the figure shows that the performance of all algorithms improves with the movement of the nodes. This behaviour is attributed to a reduced number of partitions, which results from the concentration of nodes at the centre of the simulated space. This is a well-known effect of the random way-point movement model (Bettstetter *et al.*, 2003). This conclusion is supported by counts of the average number of nodes receiving each message in the $2000m \times 2000m$ tests. The values were of 4.30, 5.91 and 5.92 respectively for tests in Speeds 0, 5 and 10.

The figure shows that for high densities, all the algorithms are capable of delivering every message to all nodes. As the area of the simulation increases, so does the average distance between the nodes and the gains provided by Pampa become more clear. This



(a) Speed 0

(b) Speed 5

(c) Speed 10

Figure 3.4: Delivery Ratio of Counter Based algorithms

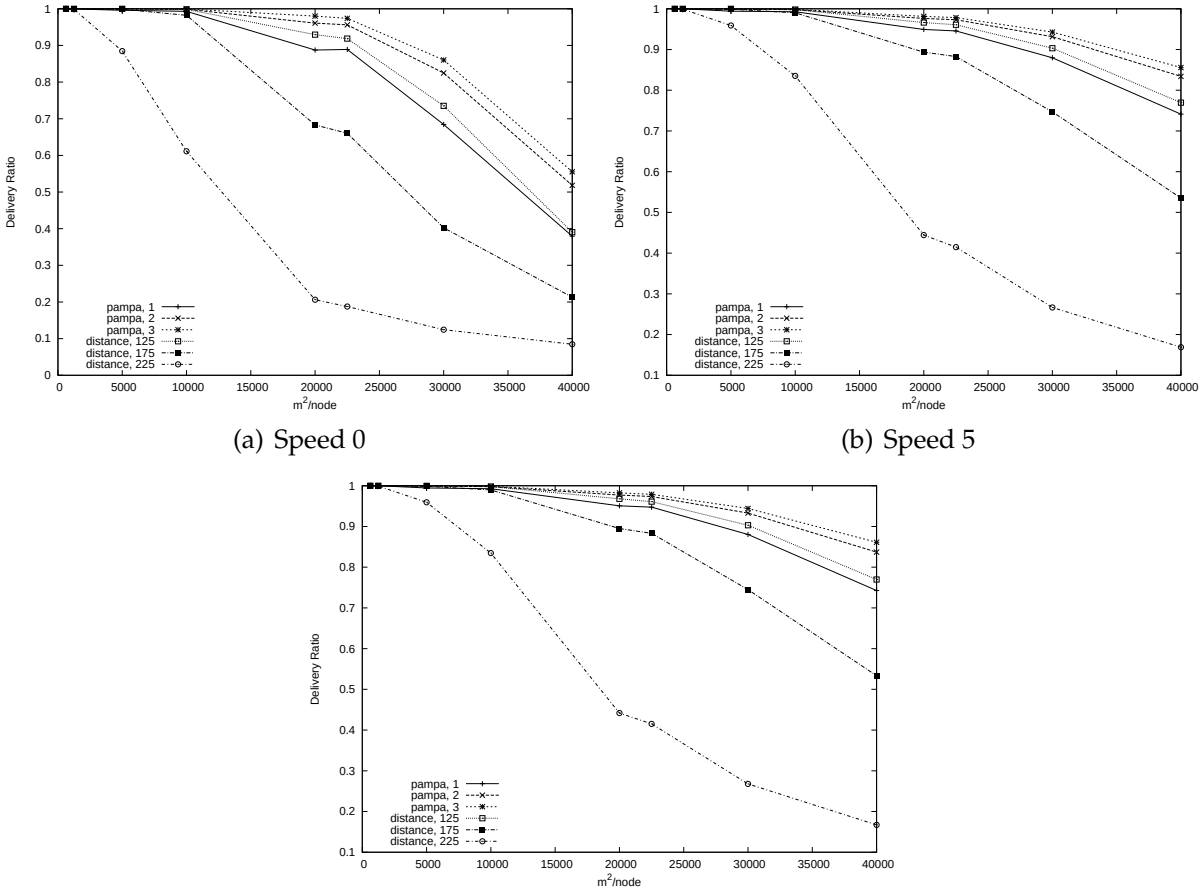
becomes more evident in the cases where the message threshold is lower. For a threshold of one message, the delivery ratio of the "counter-based scheme" begins to decay at a much faster pace than Pampa. We attribute this behaviour to the randomness associated with the node selection in the "counter-based scheme". In Pampa, the nodes forwarding the message have an higher probability of reaching more distant locations. When the simulated space is of $2000m \times 2000m$ $(40000m^2/\text{node})$ network partitions begin to affect message dissemination. The benefits of using Pampa can be more clearly observed in these extreme conditions: for the same thresholds, Pampa always presents an higher delivery ratio. The unique node selection criteria of Pampa helps to have the messages delivered to distant nodes improving its delivery ratio. HCAB presents a delivery ratio comparable to the "counter-based scheme" with threshold one.

The delivery ratio of the "distance-based scheme" with thresholds corresponding to 125m, 175m and 225m (respectively, 50%, 70% and 90% of the transmission range) is compared with Pampa in Figure 3.5. The figure shows that the "distance-based" scheme is particularly sensitive to its threshold. The "distance-based scheme" exhibits less tolerance to lower node densities than Pampa. Again, this behaviour can be partially attributed to the random selection of nodes. In sparse networks, each message is received by a small number of nodes. The propagation of the broadcast is affected if either, all nodes receiving the message are closer to the source than the minimal threshold or if the node whose timer expires first is one of the nodes closer to the source. It should be noted that in this case, the retransmission by this node is likely to prevent the retransmission from other nodes whose additional contribution to the coverage space could be higher. Due to their poor performance, the rest of the evaluation omits results for the "distance-based scheme" with thresholds corresponding to 225m and 175m.

Figure 3.6 presents a more conclusive comparison between the two best performing algorithms presented before: Pampa and the "counter-based scheme". Figures 3.6(a), 3.6(b) and 3.6(c) divide the delivery ratio of Pampa by the delivery ratio of the "counter-based scheme" for the same threshold. Results clearly show the benefits of using Pampa. In lowest densities, Pampa is capable of delivering up to 30% more messages than the "counter-based scheme". The "counter-based scheme" outperforms Pampa only in a few densities and speeds and always with marginal gains.

It is interesting to notice that the gains increase for lower thresholds. This behaviour is attributed to the fact that, with higher thresholds, the probability of the "counter-based scheme" to select the most adequate nodes for retransmission increases, while Pampa tends to select these nodes even when the threshold is low. This conclusion is supported by the results presented in Figure 3.6(d) which divide the delivery ratio with a threshold of 4 by the delivery ratio with a threshold of 3. The figure shows that the "counter-based scheme" is the one that most benefits from the increased redundancy.

The overlap between the lines for Speeds 5 and 10 is due to the similar number



(a) Speed 0

(b) Speed 5

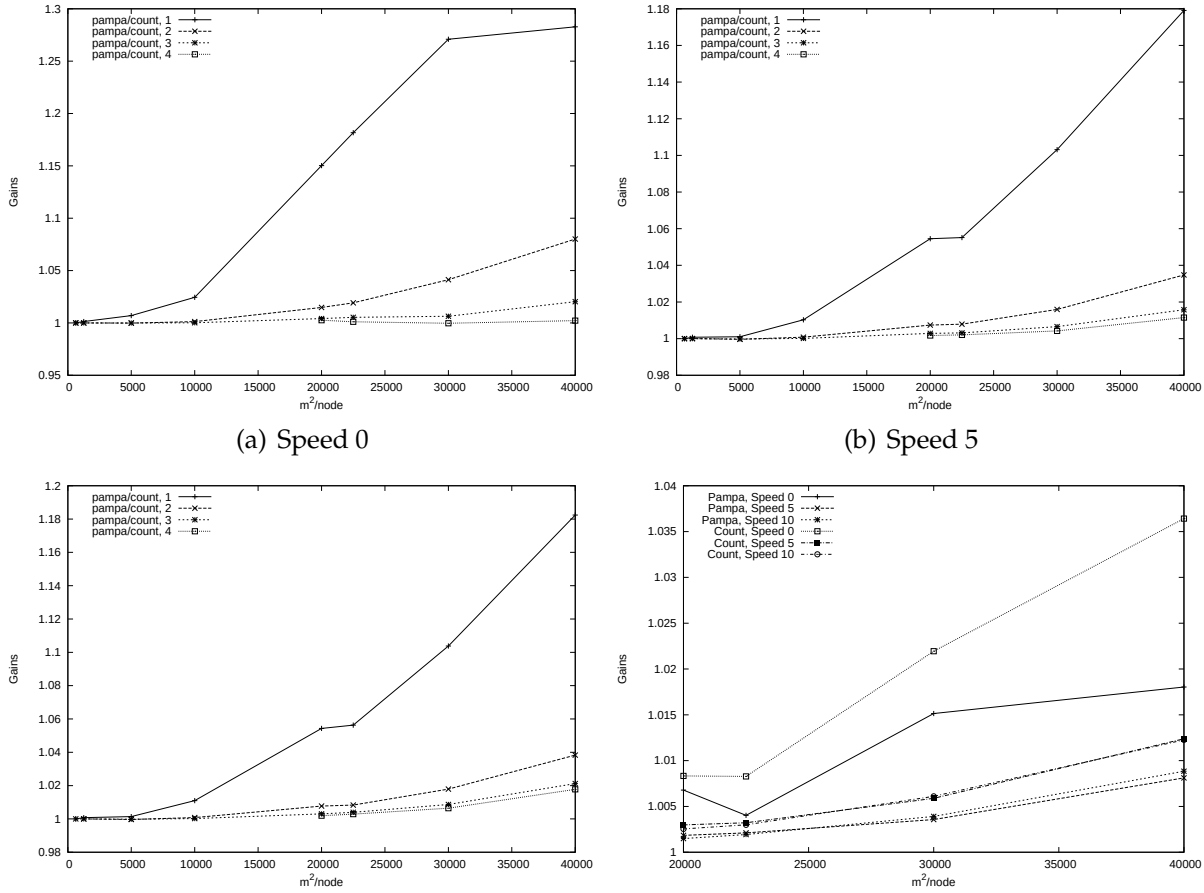
(c) Speed 10

Figure 3.5: Delivery Ratio of Distance Based algorithms

of neighbours counted in these simulations. A second conclusion to be derived from Figure 3.6(d) is that independently of the network density and speed, in general, the increment of the threshold above 3 provides only marginal gains in the delivery ratio (below 4%). This is an important conclusion, given that it provides an upper bound on the number of retransmissions and therefore, on the battery consumption required for broadcast delivery. However, results presented before have shown that smaller thresholds present considerably lower delivery ratios when the density increases.

3.3.2.1 Impact of the bounded delay

A similar comparison between Pampa and Pampa$_2$ is presented in Figure 3.7. Recall from Section 3.1.1 that the only difference between these algorithms is that in

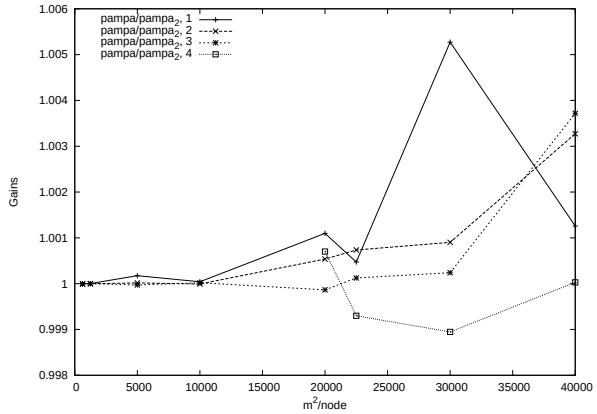


(a) Speed 0 (b) Speed 5

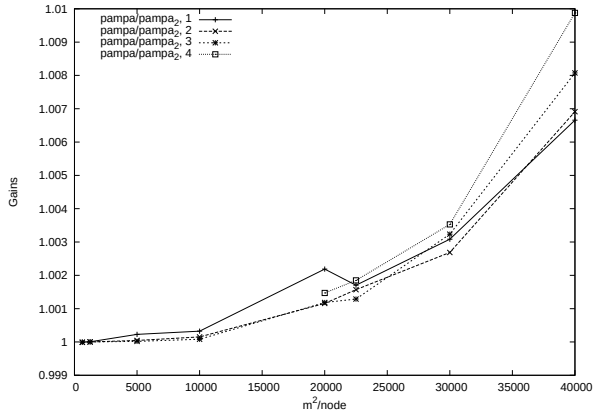
(c) Speed 10 (d) Gains using threshold 4 over threshold 3

Figure 3.6: Comparison of the delivery ratio of Pampa with the "counter-based scheme"

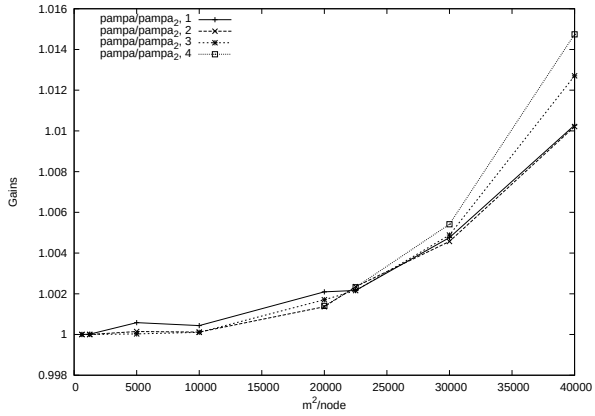
$Pampa_2$, nodes closer than 120m (approximately) to the source are randomly sorted by the jitter delay. Therefore, $Pampa_2$ should consume less memory of the mobile devices than Pampa because messages will be holden for a smaller period of time. The figure shows that in general, Pampa slightly outperforms $Pampa_2$ by small margins that do not exceed 1.6%. Such a small value shows that $Pampa_2$ can be used as a less resource demanding version of Pampa without significantly reducing its performance. The highest improvements of Pampa are for the less dense networks what confirms the relevance of the node sorting operation performed by Pampa, even when nodes are close to the source.



(a) Speed 0



(b) Speed 5



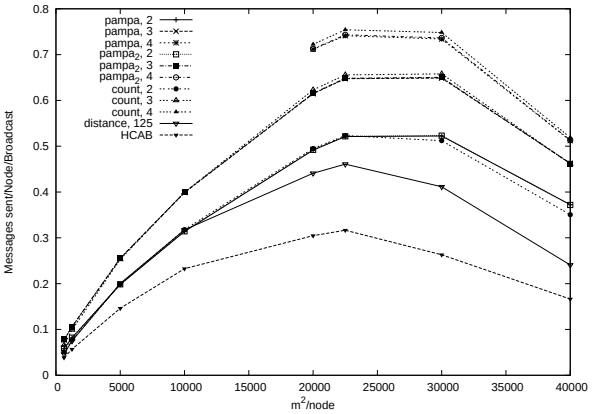
(c) Speed 10

Figure 3.7: Comparison of the delivery ratio of Pampa with Pampa$_2$

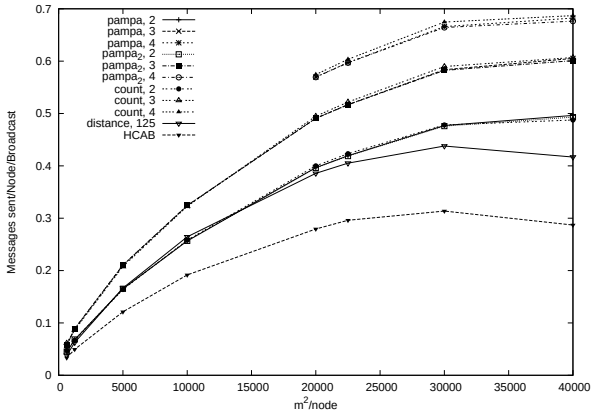
3.3.3 Number of Transmissions

The ratio of messages sent per node and per broadcast is depicted in Figure 3.8. To make the figures more readable, we omit to depict the results for the algorithms that exhibit very low reliability values.

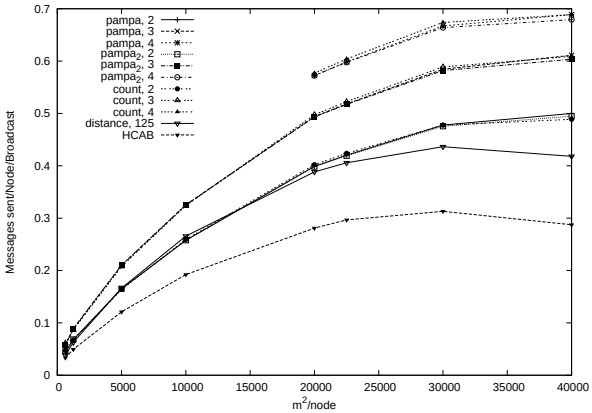
All algorithms exhibit a similar pattern, even if the absolute values differ. The decay in the number of messages, depicted in Figure 3.8(a), is again attributed to network partitions which are less likely to occur when nodes move. Nodes in a partition other than the one where the sending node is located are unable to receive the message and therefore, can not retransmit it. Pampa and the "counter-based scheme" use the same algorithm at each node for deciding whether to retransmit or not. Therefore, for the same threshold, both algorithms present a similar number of messages. However,



(a) Speed 0



(b) Speed 5



(c) Speed 10

Figure 3.8: Transmissions ratio

it was shown before that Pampa achieves an higher delivery ratio, what justifies the small increment in the number of messages transmitted for low node densities.

The figure also confirms the advantages of the counter based and distance aware algorithms over pure probabilistic ones. The GOSSIP algorithms presented in (Haas *et al.*, 2002) required a probability of retransmission above 60% to ensure an high probability of delivery to every node. Our experiments suggest that such a high retransmission rate is only required for sparse networks, where the number of neighbours is low.

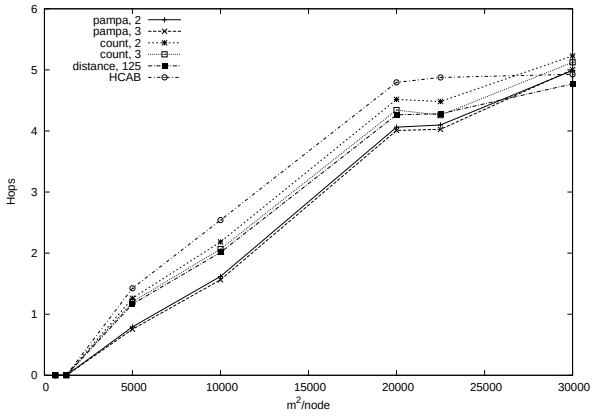


Figure 3.9: Hops to the source of the message (Speed 0)

3.3.4 Coverage

In principle, the criteria used for selecting retransmitting nodes in Pampa should improve the utility of each retransmission, measured by the number of nodes receiving a message for the first time with each retransmission. To confirm this intuition, we have counted, for each message, the number of hops travelled until it is delivered for the first time to each node. The results are presented in Figure 3.9, which omits the results for the networks with $2000m \times 2000m$ because it was considered that the low delivery ratio presented does not allow the interpretation of this metric. The figure shows that Pampa delivers messages in less hops than the remaining algorithms, confirming the utility gains expected for Pampa.

The figure also shows that contrary to the "counter based scheme", the number of hops in Pampa is independent of the threshold. This behaviour is attributed to the randomness of the "counter-based scheme": when its threshold is lower, the probability of having the most adequate nodes selected is reduced and some nodes will only receive the message after an additional retransmission. This conclusion is also supported by the good performance of the "distance-based scheme" in this metric. By prohibiting nodes too close to the source from retransmitting, the algorithm inherently favours the selection of the nodes that are capable of providing the highest coverage. On the contrary, the random selection of the nodes that retransmit in HCAB does not provide any guarantee that the node that retransmits will provide a good additional coverage.

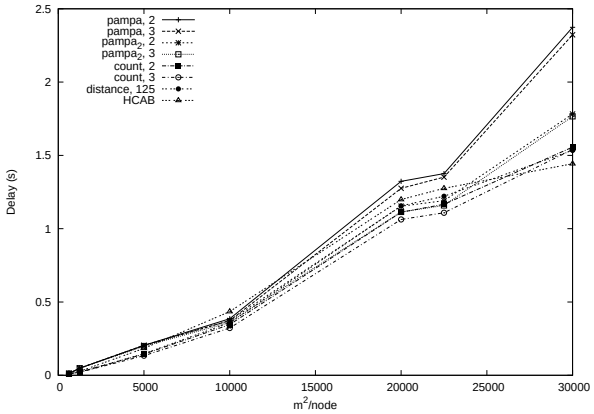


Figure 3.10: Average latency (Speed 0)

3.3.5 Delay

To conclude the evaluation, Figure 3.10 presents the average time interval between the initial transmission of a message and its delivery to each node. Results show that for high densities, messages in Pampa have a latency comparable with the remaining algorithms.

In sparse networks, the difference between Pampa and the remaining algorithms becomes significant. However, as shown before, it is in these networks that Pampa more significantly outperforms the remaining algorithms. Results for Pampa$_2$ highlight an implicit trade-off of Pampa: an excessive delay can be mitigated using a *delay* function that puts an upper bound on the delay of each nodes. However, as shown in Section 3.3.2.1, a lower latency implies a small degradation in the delivery ratio.

3.4 Summary

In Mobile Ad Hoc Networks, broadcasting a message to every node is an operation that consumes a non-negligible amount of resources at all participants. However, broadcast is a basic mechanism often required by protocols at different levels of the network stack. The most simple implementation of broadcast consists in having each node to retransmit each message after receiving it for the first time. This imple-

mentation, usually referred as flooding, creates a large redundancy of messages in the network and unnecessarily wastes resources at the participating nodes.

This chapter presented a new algorithm that uses information locally available at each node to reduce the redundancy of the broadcast operation. The novelty of the algorithm, named Pampa, is the ranking of the nodes according to their distance to the source. Pampa does not require the exchange of control messages or specialised hardware.

The algorithm was compared with previous proposals and it was shown to improve their performance, particularly in more adverse conditions like sparse networks, at the expenses of a small increase in the latency of the delivery.

4

Replica Management

Chapter 2 surveyed different data dissemination and retrieval algorithms designed for a large variety of networking models. We have observed that some dissemination algorithms attempt to place the replicas in a geographically distributed manner. There are two main reasons to promote a geographical distribution of the replicas:

Improved resilience to failures. If replicas are geographically distributed, the probability of having all of them being affected by some localised malfunction of the network, such as interference, is reduced. If the network becomes partitioned, geographical distribution may also increase the probability of having replicas available in the different partitions of the network.

Decreased cost of data retrieval. Assuming that there is no relation between the location of the nodes and the data they query, geographical distribution of the replicas reduces both the cost (in number of messages) and the time to access the data, because the probability of finding the data item in some close neighbour of the client increases.

Algorithms that promote a geographical distribution of replicas have been designed under different assumptions. Some algorithms provide a limited form of geographical distribution by preventing replicas from being close to each other (Hara, 2001). Others effectively distribute the replicas but require the nodes to be aware of their location. These alternatives have been discussed in Section 2.3.6, where we had the opportunity to note the absence of protocols that perform a proactive geographical distribution of the replicas for scenarios where the nodes are not aware of their

location. This chapter proposes to fill the gap by presenting a protocol with these characteristics.

In our work, we assume that either: $i)$ there is no predictable (biased) access pattern to the objects or; $ii)$ that this access pattern cannot be derived a priori or even during the lifetime of the system. Therefore, we aim at distributing data items as evenly as possible among all the nodes that form the network, avoiding clustering of information in sub-areas. An uniform dissemination of data items should leverage access latency to any item from any node in the network, i.e, whenever a data item is requested by a node S, the distance to the node that provides the reply should be approximately the same, regardless of the location of S. Naturally, the actual distance depends on multiple parameters, such as the number of nodes in the network, the amount of memory made available at each node, and the number of data items.

The chapter is organised as follows. Section 4.1 formalises the assumptions of the system model used by the algorithm. The protocol is described in four sections. Section 4.2 introduces the configuration parameters and the auxiliary modules. The dissemination algorithm, named Power-Aware data DISsemination algorithm (PADIS) is presented in Section 4.3. An algorithm for data retrieval well suited to the dissemination algorithm is described in Section 4.4. Section 4.5 discusses different algorithms for correcting the data distribution in the presence of node movement. Section 4.6 analytically compares our work with the algorithms surveyed in the related work. Evaluation was performed using a network simulator. The results are presented in Section 4.7. Finally, Section 4.8 summarises the results of the chapter.

4.1 System Model

Our work makes no assumption on the distribution of the nodes or on the size of the network (both in number of nodes and dimension). Devices communicate using omni-directional antennas and are able to establish bidirectional links. That is, if some node A is able to receive messages from one neighbour B, then node B is also able to

receive messages transmitted by node A. Nodes are not aware of their location or of the location of their neighbours.

To prevent trivial solutions to the data dissemination problem, we require the network to be large enough, avoiding all nodes from being within direct communication of each other. To be able to deliver data to every participating node, we assume that the network density is such that partitions are rare.

Nodes are producers and consumers of a (*a priori* unknown) number of data items, composed of a key and a value. The key is required to uniquely identify each data item. The algorithm does not interpret the content or structure of the key but assumes that the application provides a function allowing to determine if two keys are equal. Data items are short so that a few can fit in a single frame at the link layer level. Nodes always store the items they produce in a separate region of the memory with unlimited size.

Nodes are required to cooperate. Nodes must adhere to the protocol by $i)$ forwarding messages, $ii)$ keeping the items they produce in memory, $iii)$ making some storage space permanently available for keeping data advertised by other nodes and $iv)$ reply whenever listening for a query for a data item they store. Typically, the storage space at each node is enough for keeping only a fraction of the total amount of data produced.

The system model above shares many assumptions with those presented for some of the frameworks surveyed in the related work. A remarkably similar set of assumptions is presented for SAF, DAFN, and DCG, although we remove the assumptions on the predictability of the data access (Hara, 2001) and on the correlation of the queries (Hara *et al.*, 2004).

4.2 Initialisation

Bootstrap of the dissemination and query algorithms require the setting up of a number of variables depicted in Figure 4.1. The same figure also presents the configu-

```
1: procedure INIT
2:     addr←GETLOCALADDR
3:     lmsgid← 0                                          ▷ Local message counter
4:     recvdMsgs← {}      ▷ Records the messages previously received by the node
5:     STORAGE.INIT
6:     netDiameter←GETNETWORKDIAMETER
7:     qTTL←1                                             ▷ ttl for first query
8:                                          ▷ Configuration parameters constants
9:     firstQTimeout←2.5  ▷ Timeout before considering that no reply was received
10:    maxQryRetries← 5                    ▷ Number of retries before giving up
11:    prevQWeight← 0.5                    ▷ Weight of past queries for qTTL update
12:    occupThresh← 0.7                    ▷ Minimal level to bias Pampa hold time
13:    holdFactor←2.0                                     ▷ Bias limit multiplier
14:    DbC←2                                              ▷ Distance between Copies
15: end procedure

16: function CREATEMSGID          ▷ Generation of unique identifiers for messages
17:     id←(addr,lmsgid)
18:     lmsgid←lmsgid+1
19:     return id
20: end function
```

Figure 4.1: Initialisation procedure of the algorithms

ration parameters with the values used for the simulations presented in Section 4.7.

The algorithms interface with modules named PAMPA, STORAGE and TIMER providing respectively an implementation of the Pampa broadcast algorithm, data storage and an alarm service. Their interfaces are depicted in Figure 4.2. Pampa has been described in detail in Chapter 3. The purpose and functionality of the other modules is self-explanatory.

Data dissemination is triggered by some node, who is said to be the producer of the item. Dissemination and retrieval are implemented using three types of messages. In the dissemination, nodes cooperate to provide an adequate distribution of the replicas of new or updated versions of data items. Pampa is used to broadcast REGISTRATION and QUERY messages. Nodes that receive the QUERY message and store the corresponding value send a point-to-point REPLY message to the source of the query.

Pampa is not used as a black-box. Instead, every time Pampa decides to forward

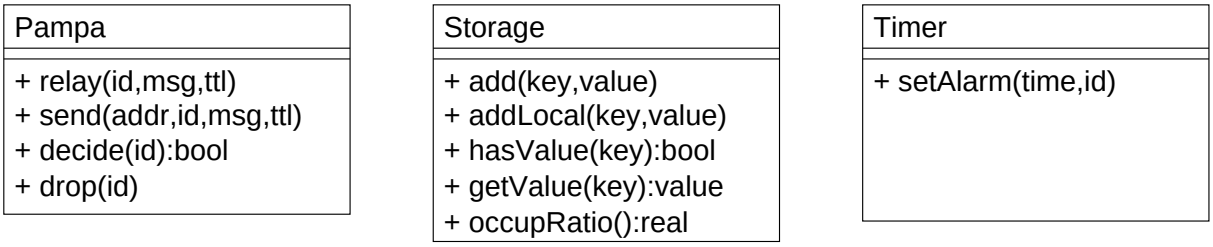


Figure 4.2: Auxiliary modules of the distribution and query algorithms

or drop a message, it first issues a call-back: the dissemination and query algorithms may then update the contents of the message, or overrule Pampa's decision on the fate of the message. In particular the algorithms may opt to force the transmission of a message that would be otherwise dropped by Pampa.

The module PAMPA exports the methods RELAY for retransmitting a message; DROP to discard a message; and SEND for point-to-point delivery to a node in transmission range.[1] The Pampa's decision to forward or drop a message is retrieved using the method DECIDE. Additionally, PAMPA notifies the algorithms by delivering events when new messages or duplicates arrive. The notification includes the suggested hold time for the message, i.e. the time the receiver should wait for other retransmissions before relaying the message.

Data is kept in the STORAGE module. The module allows to differentiate data generated at the node (using the ADDLOCAL method) and which can not be erased from data provided by other nodes (the ADD method) which can be overwritten by later entries if the storage space fills. Nodes always try to keep their storage space full, occupying all free space before beginning to overwrite other entries. Each data item is stored together with a flag *replace* whose purpose will be described later.

[1]Point-to-point message passing is associated to PAMPA only to keep on this module all network related activities.

4.3 Data Dissemination

To briefly introduce its rationale, we present in Figure 4.3 an illustrative example of one run of PADIS. The figure depicts in black the nodes that store a replica of the item. Nodes that forwarded a REGISTRATION message but did not store the data item are depicted in gray.

The dissemination begins with the broadcast of a REGISTRATION message by the producer of the data item. The item is stored at the producer and included in the message (Figure 4.3a). REGISTRATION messages carry a *Time From Storage* (TFS) field which records the distance (in number of hops) from the node transmitting the message to the known closest copy of the item. The TFS that would be putted in a message forwarded by the node as part of the propagation algorithm is depicted at its centre. Notice that this value may change with the reception of other copies.

Figures 4.3b and 4.3c show the progress of the dissemination. Nodes use Pampa to propagate the broadcast. Pampa provides two important contributions for the efficiency of the algorithm: $i)$ a small number of nodes is required to retransmit the message; and $ii)$ it selects for retransmission the nodes that are geographically more distant from the previous transmitters, an advantage that will become more clear with the explanation of the algorithm. Recall from Chapter 3 that Pampa imposes a small delay on all nodes before they retransmit a message. During this delay nodes count the retransmissions they listen. In PADIS, each node is also required to keep track of the value of the TFS field of the retransmissions it has received. The minimum TFS observed is kept in a variable named *mTFS*. When forwarding a REGISTRATION message, a node sets the TFS field to *mTFS*+1, accounting with the additional hop needed to reach the closest copy of the item.

Central to our algorithm is a constant *Distance Between Copies* (DbC). The *DbC* dictates the maximum value of the TFS field and, implicitly, the degree of replication of the items. This example uses *DbC*=2. Figure 4.3d shows that a node with *mTFS*=2 at the end of the hold period stores a copy of the item and retransmits the message. The

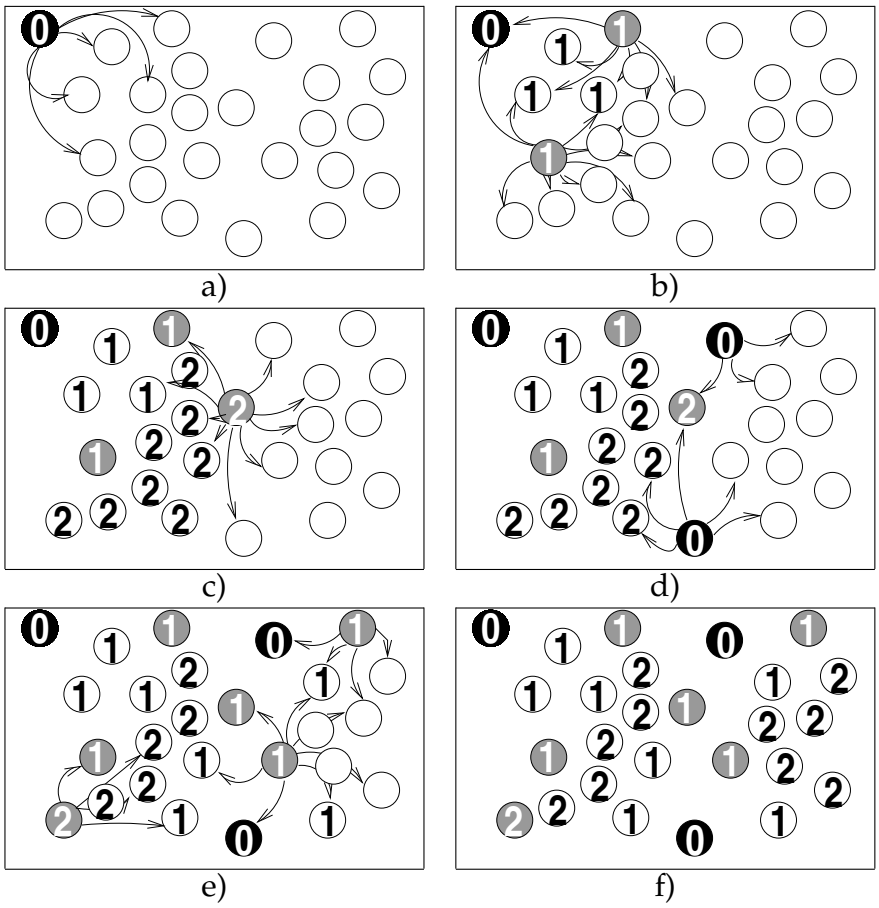


Figure 4.3: Example of dissemination of an item

TFS of the message is reset to 0 to let other nodes learn about the newly stored copy and update their *mTFS* variables accordingly (see for example Figure 4.3e).

The final state of the system after the dissemination of the item is depicted in Figure 4.3f. Although only a small number of nodes have stored the item, a replica is stored at no more than *DbC* hops away from any of the nodes.

4.3.1 Power-Aware data DISsemination algorithm

Dissemination of data items is initiated by the source node with the broadcast of a REGISTRATION message using Pampa. The dissemination algorithm is depicted in Figure 4.4. Function REGISTER (Figure 4.4, l. 1-9), shows that after receiving a call to advertise some data, the node prepares a REGISTRATION message and uses Pampa to

propagate it. In REGISTRATION messages, the *time from storage* (TFS) field indicates the distance (in number of hops) from the sender to the closest node that is known to have stored the items. Therefore, the source node sets the TFS field to zero to indicate that the record is stored locally.

When a message is received for the first time, nodes initialise *mTFS* with the value of the message's TFS field. The message is then placed on hold for a period of time which will be dictated by Pampa and biased by the storage space available at the node (l. 11-20). Section 4.3.3.1 addresses the algorithm used for determining the hold period.

Retransmissions listened by the node during the hold period are used by Pampa to decide if the message should be retransmitted and by PADIS to update *mTFS* to the lowest observed value of the TFS field (l. 23). The algorithm tries to keep a replica of each item at most *DbC* (*Distance Between Copies*) hops away of every node in the system. *DbC* is a configuration parameter that should be set according to the network conditions.[2] We assume that *DbC*> 0, given that *DbC*$= 0$ dictates a trivial solution to the dissemination problem which consists in having the items stored at every node.

When the hold period expires (l. 28-44), the nodes decide if the message is retransmitted and if the item is stored. The decision takes as input the following parameters: i) the output of the Pampa's algorithm (the return value of the call to the DECIDE method), that accounts with the number of retransmissions listened; and ii) the *mTFS* computed during the hold period. There are three possible outcomes:

- *The message is dropped and the item is not stored* if *mTFS*<*DbC* and Pampa decides not to retransmit. The combination of these conditions suggests that the dissemination has been ensured by other nodes and that a copy of the item is stored at an acceptable distance.

- *The message is retransmitted but the item is not stored* if *mTFS*<*DbC* but Pampa decides to retransmit. The message is retransmitted with TFS=*mTFS*+1 to let other

[2]Section 4.7 shows how different values of *DbC* affect the performance of the algorithm.

1: **procedure** REGISTER(data)
2: STORAGE.ADDLOCAL(data)
3: tfs$\leftarrow 0$
4: ttl$\leftarrow$netDiameter
5: mid$\leftarrow$CREATEMSGID
6: msg$\leftarrow$(REG,data,tfs)
7: PAMPA.RELAY(mid,msg,ttl)
8: recvdMsgs$\leftarrow$recvdMsgs$\cup\{$mid$\}$
9: **end procedure**

10: **upon event** PAMPA.RECEIVE(mid,(REG,data,tfs),ttl,holdTime) **do**
11: **if** mid$\notin$ recvdMsgs **then**
12: $\text{decided}_{\text{mid}} \leftarrow$false
13: recvdMsgs$\leftarrow$recvdMsgs$\cup\{$mid$\}$
14: $\text{mTFS}_{\text{mid}} \leftarrow$tfs
15: $\text{data}_{\text{mid}} \leftarrow$data
16: $\text{ttl}_{\text{mid}} \leftarrow$ttl
17: **if** tfs=DbC $\wedge$ STORAGE.OCCUPRATIO$>$occupThresh **then**
18: $\text{holdTime} \leftarrow \text{holdTime} \times \left(1 + \frac{\text{STORAGE.OCCUPRATIO} - \text{occupThresh}}{1 - \text{occupThresh}} \times \text{holdFactor}\right)$
19: **end if**
20: TIMER.SETALARM(holdTime,mid)
21: **else**
22: **if** $\neg\text{decided}_{\text{mid}}$ **then**
23: $\text{mTFS}_{\text{mid}} \leftarrow \min\{\text{mTFS}_{\text{mid}}, \text{tfs}\}$
24: **end if**
25: PAMPA.DROP(mid)
26: **end if**
27: **end upon**

28: **upon event** TIMER.ALARM(mid) **do**
29: **if** $\text{mTFS}_{\text{mid}} \geq$DbC **then**
30: STORAGE.ADD(data_{mid})
31: tfs$\leftarrow 0$
32: forward$\leftarrow$true
33: **else**
34: tfs$\leftarrow \text{mTFS}_{\text{mid}} + 1$
35: forward$\leftarrow$PAMPA.DECIDE(mid)
36: **end if**
37: **if** forward $\wedge \text{ttl}_{\text{mid}} > 0$ **then**
38: msg$\leftarrow$(REG,data_{mid},tfs)
39: PAMPA.RELAY(mid,msg,$\text{ttl}_{\text{mid}} - 1$)
40: **else**
41: PAMPA.DROP(mid)
42: **end if**
43: $\text{decided}_{\text{mid}} \leftarrow$true
44: **end upon**

Figure 4.4: Data dissemination algorithm

nodes learn about the additional hop to the closest known replica of the data item.

- *The message is retransmitted and the item is stored* when *mTFS=DbC*, regardless of the outcome of Pampa's DECIDE method. The message is retransmitted with TFS=0.

4.3.2 Geographical Distribution of the Replicas

The principal goal of PADIS is to put a replica a bounded number of hops away from any node in the network. If replicas are within a bounded number of hops, the retrieval of any data item should be possible using a limited number of messages and therefore requiring modest power and bandwidth consumption.

This section discusses different limits for the distance of the items to any node in the network. It assumes an ideal execution environment with a perfect wireless media, without message loss, collisions, or attenuation of the signal strength not resulting exclusively of the distance between the source of the message and the receivers. Therefore, Pampa is expected to always select for retransmission the nodes that are more distant to the source of the message. Node deployment is such that Pampa is able to deliver every message to every node in the network. In particular, it is assumed that there are no network partitions. Nodes do not move. The relocation of the replicas to mitigate the effect of node movements in these properties is the focus of Section 4.5.

Message propagation follows the Unity Disk Graph (UDG) model. In brief, UDG considers that each transmission is delivered to all nodes within a circle with a radius of 1 unit centred at the source. Because the transmission radius is equal for every node, if some node n receives a transmission performed by node n' then n' will also receive the transmissions performed by node n.

In this analysis, the algorithm is divided in two periods. The hold period is triggered by the reception of the first transmission advertising the data item. The duration

of the hold period is dictated by Pampa. At the end of the hold period, nodes enter the decision period, where the local outcome of the algorithm is decided.

The maximum number of hops of distance between any node and a replica is dictated by the *DbC* constant. We show it by emphasising the combination of two aspects of the algorithm:

1. The *mTFS* variable of each node is related with the distance, in hops, from the node to some copy of the item. In particular, if *mTFS*=n, then a copy is located $n + 1$ hops away of the node.

   We begin the proof by noting the relation between *mTFS* and the value of the TFS field of the messages the node receives during the hold period. *mTFS* is initialised with the TFS field of the first transmission of the message received (l. 14) and updated on the reception of every retransmission (l. 23). The update will always be to the minimum between the previous value of the variable and of the TFS field of the retransmission. Therefore, *mTFS* will necessarily be: $i)$ equal to the value of the TFS field of some retransmission it listened; and $ii)$ lower or equal to the value in the TFS field of every retransmission it listened.

   In a direct application of the previous results, we note that if *mTFS*=0 for some node p, then at least one of p's neighbours forwarded the registration with TFS=0. Lines 1 to 9 and 29 to 32 in Figure 4.4, show the two only cases where registrations are broadcast with TFS=0. These correspond respectively to the broadcast initially performed by the producer and by other nodes that also store the item. Therefore, if *mTFS*=0 then some node one hop away has stored the item.

   The result can be extended by induction for other values of *mTFS*. Notice that if the item is not stored, nodes that retransmit the message set TFS=*mTFS*+1 (l. 34). This transmission puts an upper bound on the value of the *mTFS* of the receivers, which are the one hop neighbours of the node. The bound is one unit above the *mTFS* of the sender, thus correctly reflecting the additional hop required to reach the replica.

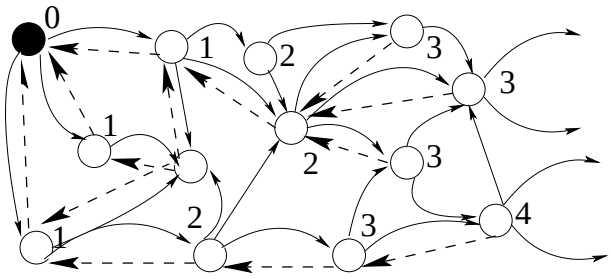


Figure 4.5: Recursive reverse shortest path to a replica

In practice, this result shows that it is possible to define a path from any node q to a copy of the item located *mTFS*+1 hops away. The path is defined by recursion, starting at step 0 with node q. Let r be the node added on step $i, i \geq 0$. The recursive step $i + 1$ adds to the path one of r's neighbours, selected from those that transmitted with a TFS value equal to r's *mTFS*.

The recursion is exemplified in Figure 4.5. Solid arrows in the figure represent the dissemination of a data item, with the number close to each node showing the value of the TFS field in the message it broadcasts. The dashed arrows represent the reverse path that can be followed by the nodes to reach the copy at the expected distance. Note that a node broadcasting a message with TFS=n needs to make n reverse hops to reach the node that stored the data item.

2. A node with *mTFS*=*DbC* stores the item and retransmits with TFS=0.

   This feature trivially results from lines 29 to 32 of Figure 4.4.

The second aspect imposes an upper bound on the possible values of TFS, which can not be greater than *DbC*. However, as shown by the first, this value is closely related to the number of hops from each node to some copy. Therefore, it can be concluded that the *DbC* constant effectively imposes a maximum distance from each node to a copy of the data item: at the end of the hold period, all nodes with a known distance to a copy greater than *DbC* store the item.

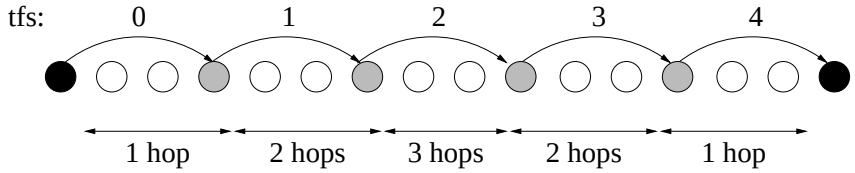


Figure 4.6: An example of data propagation in the algorithm.

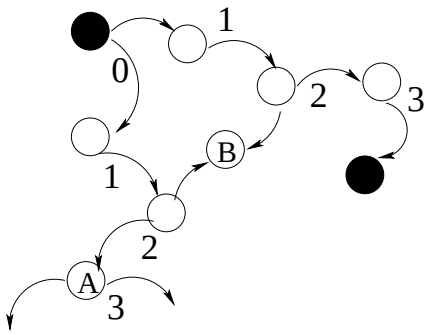


Figure 4.7: Counter-examples to the general expected distance rule

4.3.2.1 Expected Maximum Distance

In general, the upper bound presented above can be further reduced by approximately half. For the majority of the nodes, a copy will be found not more than $\left\lceil \frac{\text{DbC}+1}{2} \right\rceil$ hops away. The rationale is intuitively demonstrated by observing Figure 4.6. The figure shows the increasing TFS field of the retransmissions for a configuration with *DbC*$= 4$ and the distance at which the nodes will be able to find a copy. Nodes that forwarded the item are depicted in gray and nodes that forwarded and stored the item in black. Notice that for each node, the closest copy may be located in some node that has already retransmit the message or in a node that will eventually retransmit it.

The exceptions to this rule are cases where the propagation of the data item dies before the TFS of the successive retransmissions reach *DbC*. We identify two particular cases, both depicted in Figure 4.7, where *DbC*=3 is assumed. Node A is close to the geographical boundaries of the network. There is no node to receive its transmission and store the data item after it. Node B is surrounded by retransmissions with a lower TFS that prevent *mTFS* from growing on its neighbours as it would expect.

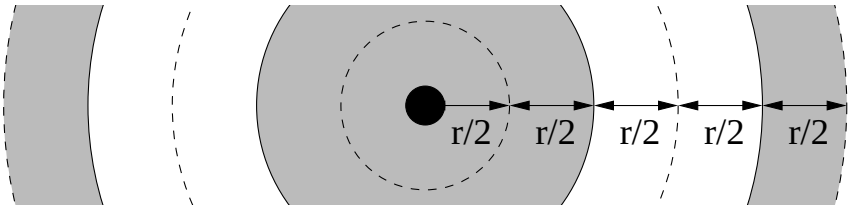


Figure 4.8: Partitioning of the cluster in rings for *DbC*=4

4.3.2.2 Expected Reply Distance

The same rationale used for determining the expected maximum distance can be used to derive the expected average distance from any node to a data item, under two assumptions: $i)$ that the exceptions mentioned are not frequent, and therefore do not significantly contribute for biasing the average, and $ii)$ that nodes are uniformly deployed in the network.

Average is estimated by considering each node storing the data item as a cluster head. The cluster is composed by this node and by all nodes closer to this node than to any other cluster head. The analysis above suggests that the nodes are those located at less than $\left\lceil \frac{\text{DbC}+1}{2} \right\rceil$ hops away from the cluster head. However, as Figure 4.6 shows, when *DbC* is even, some nodes are equidistant in number of hops but not in metric distance of the two cluster heads. Therefore, the estimation must account with the metric distance and not the hops so that each half of these nodes is attributed to one cluster head.

The estimation considers the proportion of the nodes located at each number of hops from the cluster head. We assume that all nodes have the same transmission range r. To encompass the cases for an odd and even *DbC*, the cluster is partitioned in *DbC*+1 rings, centred at the cluster head and with a $r/2$ width, as depicted in Figure 4.8.

Since we assume that nodes are uniformly distributed, the proportion of nodes on each ring to the total number of nodes in the cluster is equal to the proportion of the area of each ring to the area of the cluster. The later is given by Equation 4.1.

$$\frac{\pi\left(\frac{i+1}{2}r\right)^2 - \pi\left(\frac{i}{2}r\right)^2}{\pi\left(\frac{\text{DbC}+1}{2}r\right)^2}, i = 0 \ldots \text{DbC} \tag{4.1}$$

The number of hops required by each node to reach the cluster head depends of the ring where the node is located and is incremented at every two rings. This is given by Equation 4.2.

$$\left\lceil \frac{i+1}{2} \right\rceil, i = 0 \ldots \text{DbC} \tag{4.2}$$

The function $\tau(\text{DbC})$ gives the expected average distance from any node to the cluster head. The function combines Equations 4.1 and 4.2.

$$\begin{aligned} \tau\left(\text{DbC}\right) &= \sum_{i=0}^{\text{DbC}} \left(\left\lceil \frac{i+1}{2} \right\rceil \frac{\pi\left(\frac{i+1}{2}r\right)^2 - \pi\left(\frac{i}{2}r\right)^2}{\pi\left(\frac{\text{DbC}+1}{2}r\right)^2} \right) \\ &= \frac{\sum_{i=0}^{\text{DbC}} \left(\left\lceil \frac{i+1}{2} \right\rceil (2i+1) \right)}{(DbC+1)^2} \end{aligned} \tag{4.3}$$

Figure 4.9 depicts a plot of function τ. The figure shows that the algorithm moderately increases the average distance between any node and the closest copy of an item with the *DbC* constant. This suggests that when we consider the limited storage space made available by the nodes, the algorithm will scale well with the number of items by increasing *DbC*.

4.3.2.3 Cluster Storage Space and Saturation Point

The storage capacity of a subset R of the nodes in the network can be determined by adding, for each node $n \in R$, the storage space it makes available for keeping items produced by other nodes (RS_n) with the storage space occupied by the items the node

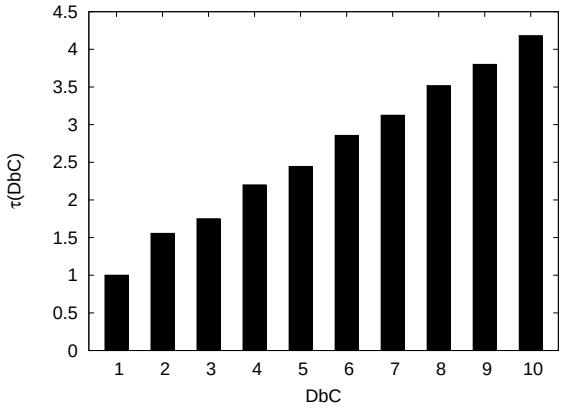


Figure 4.9: Function $\tau(\text{DbC})$

has produced (LS_n). That is, the total storage space for the subset is given by:

$$SS(R) = \sum_{n \in R} (RS_n + LS_n) \tag{4.4}$$

Assuming that all nodes produce an equal number of data items and make available an equal amount of storage space for keeping items advertised by other nodes, Equation 4.4 can be approximated to:

$$SS'_R = |R| \times \left(RS + \frac{|I|}{|N|} \right) \tag{4.5}$$

where $|R|$ is the cardinality of the set R, RS is the storage space made available for data items produced by other nodes, $|I|$ is the total number of items produced by all nodes in the network and $|N|$ the total number of nodes in the network.

Previously, we showed that for each data item, our algorithm divides the network in clusters, each composed of the node storing a copy of the item and those that are closer to this node than to any other node also storing a copy of the item. If we again assume an uniform deployment of the nodes and ignore those at the boundaries, each cluster should have approximately the same number of nodes. Each run of the algorithm defines a possibly different partition of the network in clusters. Because nodes are uniformly distributed and each node generates approximately the same number

of data items, each node will be elected as a cluster head for approximately the same number of data items. Therefore, each cluster will define the total capacity of the network for encompassing new items without degrading the properties of the algorithm.

We define the *saturation point* (SP) of the algorithm as the multiple solutions of equation 4.6

$$SS'_C = |I| \tag{4.6}$$

for a generic cluster C composed by an average number of nodes. That is, the SP gives the multiple combinations of storage space on each node, number of items and number of nodes in the cluster that are theoretically sufficient for storing all data items advertised in a single cluster. The number of nodes in a cluster depends of the *DbC*. Therefore, Equation 4.6 shows that the algorithm can be tailored to networking environments with different number of items and storage space at the nodes. Recall that Section 4.3.2.2 has already shown that a small growth in *DbC* does not represent a significant increase in the average distance to the closest copy of a data item.

For each network, the *saturation point ratio* (SPR) is defined as the ratio between the storage space made available in a cluster and the total number of items.

$$SPR = \frac{SS'_C}{|I|} = \frac{|R| \times \left(RS + \frac{|I|}{|N|}\right)}{|I|} \tag{4.7}$$

In the theoretical model, the algorithm should be able to provide its properties for all values of $SPR \geq 1$. Section 4.7 will use the SPR to evaluate the performance of the algorithm in a simulated environment.

4.3.3 Decreasing the Impact of the Limited Storage

We assume that nodes make available limited storage space for keeping data items advertised by others. This can pose a problem to the performance of the algorithm

given that nodes elected as cluster heads for some data items may not have storage space available for keeping the item. This section shows how the problem is addressed in the algorithm. It first presents a mechanism to improve the selection of the nodes that store the data items, possibly sacrificing the best possible deployment of the replicas. Next, the section briefly discusses storage space managing policies, to be implemented locally at each node.

4.3.3.1 Hold period

In a broadcast, Pampa selects the most adequate nodes for retransmission using the gains in the coverage of each retransmission as the single criteria. Pampa is unfair, because it does not attempt to leverage the number of retransmissions among neighbouring nodes. Depending on the deployment of the nodes and of the location of the producers, some nodes may have their probability of being required to retransmit increased. It was shown before that the probability of being required to store a data item is related with the order of timer expiration among the nodes, as dictated by Pampa. Therefore, some nodes may be more frequently requested to store data. As a result, these nodes may exhaust their local storage space significantly before the other nodes in the neighbourhood.

Figure 4.4 (l. 17-19) show that PADIS compensates this effect by biasing the hold time so that it depends also on the available storage space of the node. The biasing mechanism is triggered locally at the node by the combination of two factors: $i)$ the reception of a REGISTRATION message with TFS=*DbC*, and $ii)$ an occupancy ratio of the storage space above some constant threshold. Notice that the first criteria is justified by knowing that if some message, and in this case, the first, was not received with TFS=*DbC* then the item will not be stored by the node.

As shown in Equation 4.8, the hold period dictated by Pampa (represented as *hPampa*) is used until the node reaches an occupancy ratio threshold (*thresh*). Above this threshold, the hold time is increased proportionally to the occupancy ratio. The

weight of the additional delay in the hold period is dictated by a configuration parameter *bias*.

$$holdPeriod = \begin{cases} \text{hPampa} & , \text{TFS} < \text{DbC} \vee \text{occup} < \text{thresh} \\ \text{hPampa} \times \left(1 + \frac{\text{occup}-\text{thresh}}{1-\text{thresh}} \times \text{bias}\right) & , \text{otherwise} \end{cases} \tag{4.8}$$

4.3.3.2 Storage Space Management

The algorithm tries to distribute data items as evenly as possible by the nodes. However, some nodes may see their storage space filled before its neighbours, either because they are in a region with a low density or because the number of items advertised exceeds the storage capacity of the region. Nodes are required to keep adding items to their storage space until it is completely filled.

When the storage space fills, nodes are required to drop stored items to make room for new items. Different policies could be adopted for the selection of the items to be dropped, for example those surveyed in Section 2.2.

We note that PADIS should avoid deterministic policies. A deterministic criteria applied to different nodes would likely select the same entry for replacement. This would eliminate a large number of replicas of the same item resulting in an uneven distribution of the item. Therefore, we adopted a policy where nodes randomly select one of the existing entries for replacement. As it will be shown later, shuffling policies that aim to mitigate the bias introduced by node movement in the item distribution may also provide a valuable contribution to the leveraging of the number of replicas of the items.

4.3.4 Illustration

PADIS was experimented in a simulated environment. The networking environment is similar to the one used in Section 4.7 for an in-depth evaluation of the algorithm. For illustrative purposes, Figure 4.10 shows the state of a network composed by 100 nodes in a region with $1500m \times 500m$, after the dissemination of 10 data items, identified by numbers between 0 and 9. In this run, the transmission range of each node is of 250m and the configuration parameter *DbC* is set to 2.

The figure is used to illustrate how geographical distribution of the data items is achieved and to develop an intuition on how the removal of the theoretical assumptions that have been presented before will affect the algorithm.

In the figure, rectangles present the items kept at the storage space of each node. Items that have been produced by the node are presented above the dividing line of the rectangle. Items produced by other nodes are presented below the line. Dotted arrows show the propagation path followed for the dissemination of item 0.

PADIS replicated each item between 3 and 6 times, excluding the original, kept at the producers. On average, items were replicated 4.9 times. Item 2 was the one less replicated, possibly due to the central point where the dissemination started. Intuition suggests that the replicas have been deployed in locations adequate to cover the entire network.

The dissemination path for item 0 is a good example of the typical behaviour of the algorithm. Retransmissions are preferably performed by the nodes more distant to the source. This saves device resources (only 25% of the nodes transmitted the message) and allows to define larger clusters, reducing the number of copies required. The shaded area of the figure highlights an undesirable occurrence: the storage of two copies of data item 0 in adjacent nodes. We attribute this behaviour to the small difference of the timers set by both nodes after the reception of the message. As a result, nodes concurrently decided to store the item without being aware of the decision of its neighbours. Unfortunately, this occurrence was not unique. A similar one can be

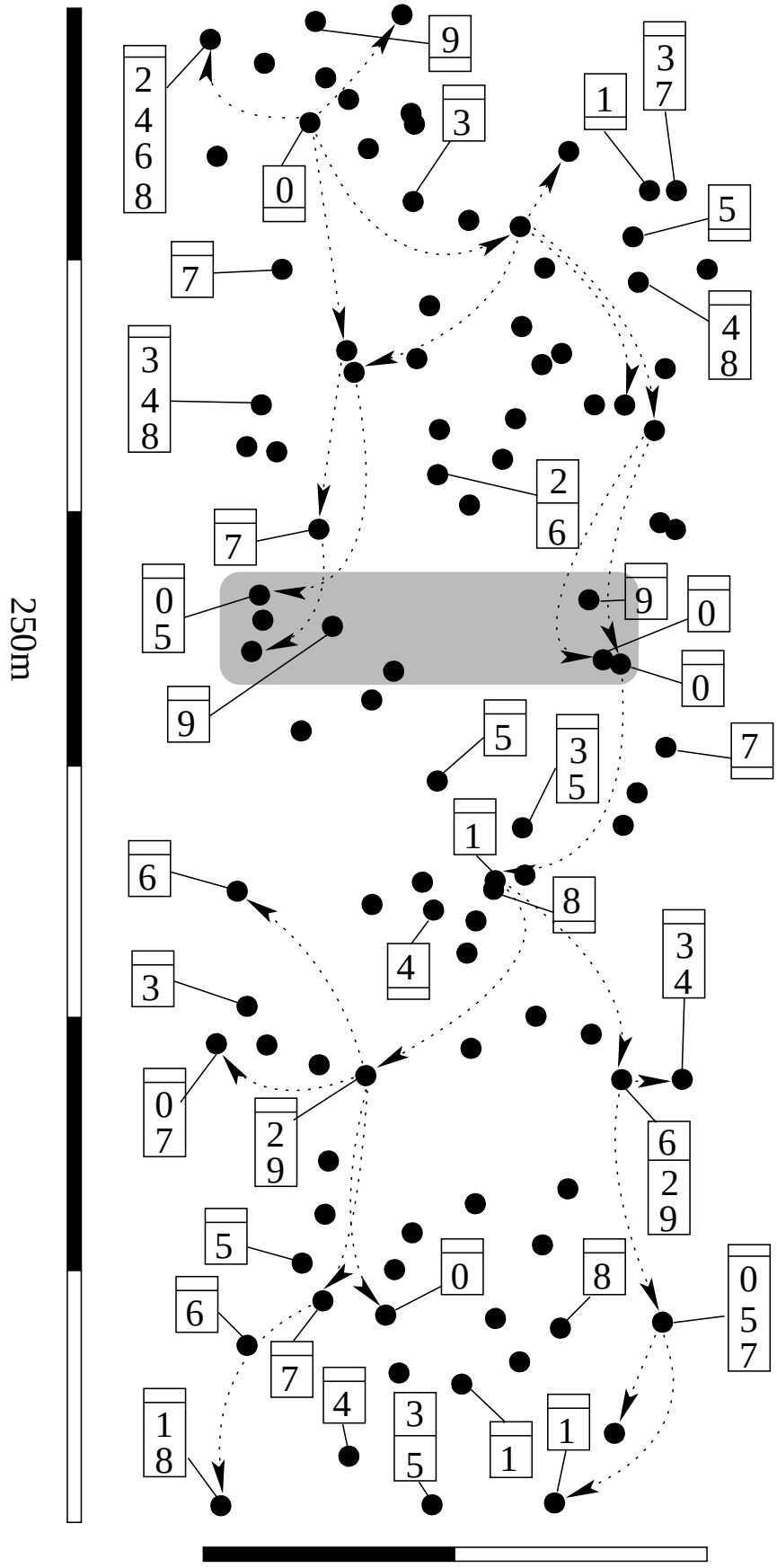


Figure 4.10: A run of the dissemination algorithm

found with data item 1 at the bottom of the figure.

The figure also shows that the replicas are evenly distributed. 35% of the nodes were requested to store at least one item advertised by other nodes. Only one node stored 4 items and two nodes stored 3 items. Even distribution is an important result as it suggests that for uniform deployments, the algorithm tends to leverage storage occupancy, adequately exploiting cluster size.

4.4 Data Retrieval

The data retrieval algorithm assumes that the items have been geographically distributed using PADIS. Therefore, it attempts to save the resources of the devices and bandwidth by forwarding the QUERY message to nodes a limited number of hops away. Only if this initial attempt fails the algorithm broadcasts the message to all nodes in the network. The data retrieval algorithm uses Pampa for the dissemination of the QUERY message. This section presents a detailed description of the data retrieval algorithm.

Figure 4.11 depicts the part of the algorithm performed by the source of the query. Nodes begin by looking for the key in their local storage. If the value is not found, the node uses Pampa to first broadcast a QUERY message within a limited number of hops, dictated by a variable *qTTL* (l. 6-8). The algorithm adapts the value of *qTTL* to the network conditions following an algorithm described below.

Lines 11 to 18 show that if no answer is received, the query will be retransmitted with a TTL large enough to reach all nodes in the network. This query will be retransmitted at growing time intervals until a reply is received or a maximum number of retries is reached.

Figure 4.12 shows how QUERY messages are handled by the nodes. A node receiving a QUERY message and that does not find the value locally (l. 41-45) pushes its address in the *route stack* field of the message, in a route construction process similar to

```
1: procedure QUERY(key)
2:     if STORAGE.HASVAL(key) then
3:         value←STORAGE.GETVAL(key)
4:         DELIVER(key,value)
5:     else
6:         ttl←ROUND(qTTL)
7:         retries_key ← 0
8:         SENDQRY(key,ttl)
9:     end if
10: end procedure

11: upon event TIMER.ALARM(key) do
12:     if retries_key < maxQryRetries then
13:         ttl←NETWORKDIAMETER
14:         SENDQRY(key,ttl)
15:     else
16:         DELIVER(key,NOTFOUND)
17:     end if
18: end upon

19: procedure SENDQRY(key,ttl)
20:     routeStack←{}
21:     PUSH(routeStack,addr)
22:     mid←CREATEMSGID
23:     msg←(QRY,key,routeStack)
24:     PAMPA.RELAY(mid,msg,ttl)
25:     recvdMsgs←recvdMsgs ∪ {mid}
26:     retries_key ←retries_key + 1
27:     qTimeout←firstQTimeout×retries_key
28:     TIMER.SETALARM(qTimeout,key)
29: end procedure
```

Figure 4.11: Retrieval algorithm - sender

```
30: upon event PAMPA.RECEIVED(mid,(QRY,key,routeStack),ttl,holdTime) do
31:     if mid ∉ recvdMsgs then
32:         recvdMsgs←recvdMsgs ∪ {mid}
33:         if STORAGE.HASVAL(key) then
34:             PAMPA.DROP(mid)
35:             mid←CREATEMSGID
36:             value←STORAGE.GETVAL(key)
37:             tfs← 0
38:             nextAddr←POP(routeStack)
39:             msg←(RPLY,key,value,tfs,routeStack)
40:             PAMPA.SEND(nextAddr,mid,msg,netDiameter)
41:         else if ttl> 0 then
42:             PUSH(routeStack,localAddr)
43:             msg_mid ←(QRY,key,routeStack)
44:             ttl_mid ←ttl
45:             TIMER.SETALARM(holdTime,mid)
46:         end if
47:     else
48:         PAMPA.DROP(mid)
49:     end if
50: end upon

51: upon event TIMER.ALARM(mid) do
52:     if PAMPA.DECIDE(mid) then
53:         PAMPA.RELAY(mid,msg_mid)
54:     else
55:         PAMPA.DROP(mid)
56:     end if
57: end upon
```

Figure 4.12: Retrieval algorithm - query handling

the *Simple Search* (SS) algorithm (Lim *et al.*, 2006). The retransmission of the message is dictated by Pampa, as depicted in lines 51 to 57.

If the key is found (l. 33-40), the QUERY message is not retransmitted. Instead, the node sends a point to point REPLY message to the source of the query. The TFS field of the REPLY is set to 0 at the origin and incremented at every intermediate hop to reflect the distance at which the data item was stored. The REPLY message follows the path constructed in the *routeStack* field of the QUERY (Figure 4.13, l. 66-68). When the node that issued the query receives the first reply, it cancels the pending timer and delivers

```
58: upon event PAMPA.RECEIVED(mid,(RPLY,key,value,tfs,routeStack),ttl,holdTime)
    do
59:     if routeStack={} then
60:         if TIMER.EXISTS(key) then
61:             TIMER.CANCELALARM(key)
62:             DELIVER(key,value)
63:             qTTL←prevQWeight×qTTL+(1−prevQWeight)×tfs
64:         end if
65:     else
66:         nextAddr←POP(routeStack)
67:         msg←(RPLY,key,value,tfs+1,routeStack)
68:         PAMPA.SEND(nextAddr,mid,msg,ttl-1)
69:     end if
70: end upon
```

Figure 4.13: Retrieval algorithm - replies handling

the (key,value) pair (l. 59-62).

4.4.1 Adaptation of the *qTTL* value

To save resources, it is desirable that most of the queries get a reply in the first broadcast of the QUERY message. An early reply avoids the second broadcast which is sent to every node in the network and therefore consumes a significantly higher amount of power. The range of the first QUERY message presents an interesting trade-off. It should be large enough to avoid a second broadcast in the majority of the cases but it should be as small as possible to prevent the waste of energy that results from the retransmissions of the QUERY message to nodes more distant than the closest copy of the item. Previously, Section 4.3.2 showed that nodes should be able to retrieve any data item within a bounded distance. In the algorithm, the TTL of the first QUERY message is defined by a variable *qTTL* whose value is adjusted after each query. As depicted in line 63, *qTTL* is tuned by weighting past experiences of the node with the distance at which the reply was found, given by the TFS field of the REPLY. The weight of past experiences and of the latest result is dictated by a constant *prevQWeight* which can be initialised according to the expected conditions of the network, for example by privileging the latest experience if it is expected that network conditions change

rapidly.

4.5 Shuffling

PADIS performs the geographical distribution of the replicas of the items. Unfortunately, node movement will cause distortions to the initial distribution. In addition, the illustration of Section 4.3.4 also showed that the geographical distribution is not perfect and may place replicas of the data items in nodes close to each other. Therefore, a relevant problem is to find algorithms that are able to correct distortions in the replica distribution. We call these "shuffling" algorithms. As their name implies, shuffling algorithms operate by moving replicas from one node to the other, in an attempt to improve the replica distribution. Shuffling algorithms can be used: $i)$ as a corrective measure, to restore an even replica distribution from an unbalanced initial scenario or; $ii)$ pro-actively, to counterbalance the effect of node movement, preventing the system from ever reaching an uneven replica distribution.

Data shuffling is strongly related with the management of the storage space. The shuffling algorithm must decide which items of the storage space it should replace or deliver to other nodes. In our model, each entry of the storage space has a *replace* boolean flag associated. When the *replace* flag is set, this means that the associated item is a good candidate to be moved to another part of the network. Typically, the *replace* flag may be set when a node notices that there are other copies of the same item in its vicinity.

The reader may notice that if one has a perfect "shuffling" algorithm, the initial placement becomes less relevant: no matter the initial distribution of replicas, the system would always converge to an even distribution. However, as we will show, shuffling algorithms may require some time to converge and may benefit from a good initial placement.

4.5.1 Effects of Movement in Data Placement

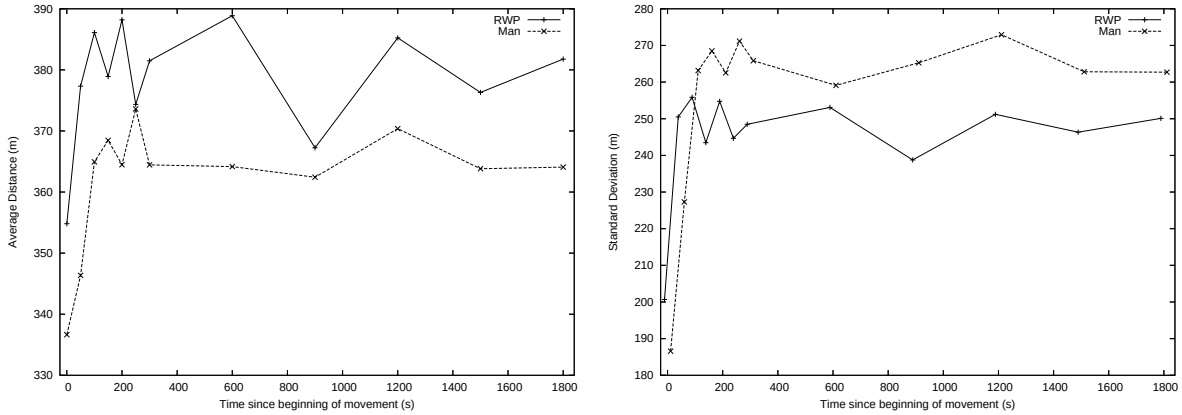
Intuition suggests that the movement of nodes tends to create scenarios where item are unevenly distributed, even if initially replicas were placed using a perfect algorithm. To assess if this intuition proves to be correct, we prepared a simple simulation were 100 nodes are initially deployed according to one of the Random Waypoint (RWP) or Manhattan (Man) movement models in a region with $1500m \times 500m$. Nodes have a transmission range of 250m and store a replica of each data item (*DbC*) every 5 hops.[3]

Initially, nodes do not move and used the dissemination algorithm to distribute 100 items. To remove any bias from this study, no storage constraints were applied to the nodes: nodes can store all the items required by the placement algorithm. At the end of the dissemination period, we measured the average and the standard deviation of the distance (in meters) of each node to the closest copy of each item. These values are depicted at time 0 in Figure 4.14 which presents the average and the standard deviation of each node to each data item averaged from 10 runs with different movements of the nodes. As it can be observed, at time 0, the algorithm places a copy of each data item on average at 336.64m (resp. 354.81m) of each node, in the Manhattan (resp. Random Waypoint) movement model. The ratio of the standard deviation to the average is at this stage of 55.4% (resp. 56.6%).

After the placement, nodes moved according to the predefined movement models for 1800s, without pausing and with a speed of approximately $10m/s$. No messages were exchanged during this period. The distance of each node to the closest node storing a copy of each item was measured periodically, considering the location of each node at that time instant.

The figure confirms that random movement does not provide an homogeneous distribution of the replicas. Because nodes move at random in a delimited region, the average distance can not vary significantly. In the random Waypoint simulations, the average distance could even become below the value found after the placement due

[3]For additional details about the simulation environment see the baseline configuration described in Section 4.7.



(a) Average (b) Standard deviation

Figure 4.14: Effects of movement in the distribution of the replicas

to the well known tendency of the model to concentrate the nodes at the centre of the simulated region (Bettstetter *et al.*, 2003).

The large increase (up to 50%) in the standard deviation, however, shows that there is a much more irregular distribution of the distance of the nodes to the closest copy of each item. Node movement makes the ratio of the standard deviation to the average to be between 65.6% and 73.7% in the Manhattan movement model and between 64.3% and 66.3% in the Random Waypoint. In both cases, 50s after nodes have began to move, the ratio has increased 10%. Relevant for our study is the property of the standard deviation estimator stating that when comparing two standard deviations for the same average, the higher one will include samples with higher distances to the average. Therefore, any increment in the standard deviation is particularly harmful for the data retrieval algorithm as it increases the maximum distance at which an item will be found with consequences on the stabilisation of the *qTTL* value.

4.5.2 Herald Messages

Shuffling algorithms require the exchange of control information among the mobile nodes. Control information is exchanged in HERALD messages. The precise purpose of HERALD messages varies with the shuffling algorithm. For simplicity, we define

data item	hops	lifetime	tfs	store

Figure 4.15: Format of HERALD message tuples

a unique HERALD message format to be used by all the shuffling algorithms to be presented. As it will be clear from the exposition, not all fields are relevant for every algorithm. HERALD messages carry a set of records with the format presented in Figure 4.15.

The field *data item* carries a data item. The field *hops* is initially set to zero and incremented every time the HERALD message is forwarded. The field *lifetime* defines the maximum value for the *hops* field; when a tuple reaches *lifetime* hops the tuple is removed from the message. By setting the *lifetime* one can control how far an item is announced in the network. For instance, by setting a lifetime of one, it is possible to announce a data item to a direct neighbour only. If one or more records have reached their lifetime, a node forwarding the message replaces them by new records, using the same strategy as the source of the message.

The TFS field indicates how many hops the record was forwarded without meeting a replica of the data item. It is incremented every time a node forwards a tuple for a data item it does not own in its local database, or reset to zero otherwise.

The boolean field *store* indicates the purpose of the dissemination of the item. If *store* is set to "false", the item is being disseminated with informational proposes, so that other nodes learn that the item is stored at TFS hops away. A "true" value allows nodes to store the data item in their storage space if some conditions discussed below are met.

4.5.3 Characterisation of Shuffling Algorithms

The shuffling algorithm may cause the number of replicas of each data item to change. For instance, if in order to create a new local replica of a data item a node

needs to discard a replica of another item (due to space constraints) one is increasing the number of replicas of one item and decreasing the number of replicas of another item. Due to mobility, message losses and node disconnections, it is impossible to design a shuffling algorithm that preserves exactly the original number of replicas in the system. In addition, the most adequate number of replicas of each data item may vary to attend to changes in the topology or simply to the location of the replicas. Concerning the number of replicas, we present below two classes of algorithms: those that try to keep a stable number of replicas and those that assume that the number of replicas will be leveraged by random factors. The evaluation will compare them under this perspective.

When a node forwards an HERALD message it may remove, add, or replace the tuples carried in the message. Different algorithms may use different policies to perform these actions as well as to define the route of a HERALD message. For instance, one node may send a HERALD message to a specific node in the system using point-to-point routing. We may also avoid sending control packets explicitly for the operation of the shuffling algorithm by piggybacking HERALD messages in data QUERY or REPLY messages. In this case, the route of the HERALD message is constrained by the route of the control message it is piggybacked with.

4.5.4 Shuffling Algorithms

This section proposes four independent shuffling algorithms that can be integrated with the data retrieval algorithm. One possible classification of the algorithms, named *Default*, *Swap on Query*, *Advertise State* and *Probabilistic*, considers the use of HERALD messages and their effort on preserving the number of replicas of each data item. These aspects are compared in Table 4.1. The algorithms work as follows:

Default A reply found far away from the source of the query signals an uneven distribution of the item, possibly due to the violation of some of the assumptions of the theoretical model or to the movement of the nodes. In the Default algorithm,

Algorithm	Herald messages		Preserve replicas
	Piggyback	On-demand	
Default			
Swap on Query		•	•
Advertise State	•	•	•
Probabilistic	•		

Table 4.1: Comparison of the characteristics of the shuffling algorithms

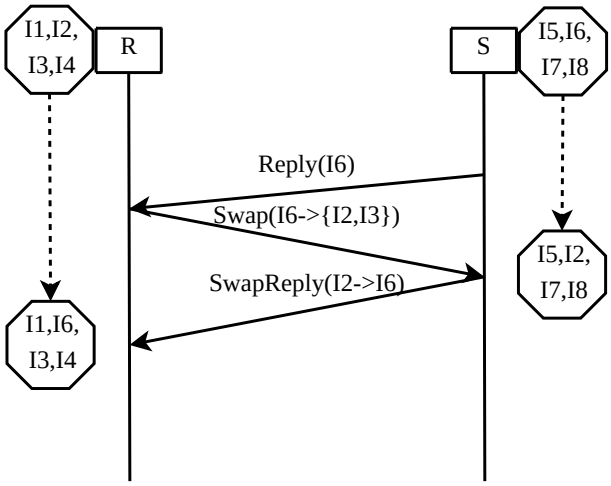


Figure 4.16: Swap of data items between two nodes storage spaces

nodes use the TFS field of REPLY messages to learn the distance at which the item was found. To improve the distribution of the item, nodes update their storage space when they receive a reply to a query from more than *DbC* hops away. The data item received will occupy a random position of the storage space, replacing the existing item. The Default algorithm does not use HERALD messages.

Swap on Query Like for the Default algorithm, a node R decides to store a data item when the first reply is received from a node located at more than *DbC* hops away. The *Swap on Query* algorithm attempts to preserve the number of replicas of each data item by performing a swap with the node S that sent the REPLY. The swap process is summarised in Figure 4.16.

Instead of randomly selecting an item for being erased, node R sends a point-to-point HERALD message to node S. The message follows the reverse path of the REPLY. Each record in the message will contain one of the data items in R's

storage space. The TFS field of the records is set to zero and incremented or reset to zero by the intermediate nodes that forward the HERALD message.

If the item sent in the reply was not produced by S, then node S selects one of the items in the HERALD message and swaps it with the item initially sent to R in the REPLY. The item is randomly selected among those with an highest TFS and which is not present in S's storage space. The details of the swap performed by S are sent in a *SwapReply* HERALD message addressed to R. After receiving it, R replaces the item selected by S with the reply to the query.

If S is the producer of the data item, it cannot remove the item from its storage space. In this case, the *SwapReply* message will contain only the item that was originally replied. When receiving the message, node R randomly selects one of the items in its storage space and replaces it by the item in the REPLY. It should be noted that in this case, the number of copies is not preserved: the queried item gets one additional replica in the system while some other item randomly selected will have one less.

Advertise State In *Swap on Query*, nodes select for swapping any data item satisfying a simple set of constraints. Results should be improved if the items were selected from those that are known to be redundant because another replica is stored in some close neighbour. The *Advertise State* algorithm attempts to mitigate this problem by piggybacking informative HERALD messages in QUERY messages.

HERALD messages are filled with records up to the maximum message size, thus preventing fragmentation of the QUERY message. Data items are selected from those in the storage space having the *replace* flag set to false. The TFS and *hops* fields of the records are set to zero and the *lifetime* field is set to the *DbC* constant. The *store* flag is set to *false*.

Nodes receiving the QUERY message compare the data items contained in the HERALD message with those in their storage space. Data items present in both have their *replace* flag set to true, thus signalling the availability of another replica in the node's proximity. In the *Advertise State* algorithm, the *swap* operation de-

scribed in the *Swap on Query* is changed to preferably select items with the *replace* flag set to true.

This algorithm puts additional effort in the preservation of the number of replicas. REPLY messages include a flag which, if active, informs that the item can not be swapped. In this case, the node that performed the query will not attempt a swap and will not insert the item in its storage space.

Probabilistic The *Probabilistic Shuffling* algorithm continuously updates the storage space of the nodes thanks to the exchange of HERALD messages piggybacked in QUERY messages. Probabilistic shuffling does not require the transmission of dedicated control messages.

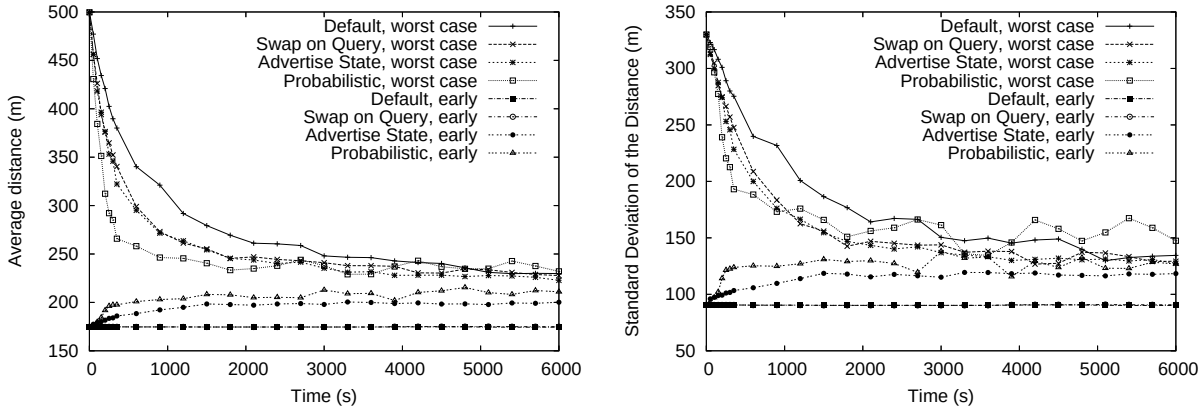
Records of the HERALD message are filled with data items randomly selected from the storage space. Data items in the storage space with the *replace* flag set to *false* are advertised with the same purpose described in the *Advertise State* algorithm. The *lifetime* of these records is set to *DbC*.

Data items with the *replace* flag set to *true* will have the *store* flag of their records also set to *true*. In addition, these records will have the *lifetime* field set to the TTL of the QUERY message. These records are used to create new replicas of the data item. A new replica will be created in a node that receives the message if all the following conditions are met:

1. the HERALD message record has the *store* flag set to *true* and TFS>*DbC*;
2. the node has some storage space entries with the *replace* flag set;
3. a random number generator selects a number with some probability p_{ins}.

The item replaces one of the entries with the *replace* flag set.

A second probability dictates the refreshment rate of the records in the HERALD message. Besides the refreshment imposed by the *lifetime* field, nodes that retransmit the QUERY message may replace each of the records with a data item they own in its storage space with probability p_{rep}.



(a) Average (b) Standard Deviation

Figure 4.17: Distance evaluation of a worst case scenario simulation

4.5.5 Illustration

To develop an intuition on the behaviour of each of the four algorithms, we prepared a worst case scenario simulation and closely monitored the status of the storage space of each node.

In the simulation, 100 nodes were randomly deployed in a region with $1500m \times 500m$. Each node produced one data item. In the worst case scenario, four copies of each data item were deployed: one in the node that produced it and each of the remaining on the three nodes that were geographically closer. This scenario was named "worst case". The "early" scenario was used for comparison. In the "early" scenario, items were previously distributed using PADIS. In both cases, after the deployment of the replicas, 3000 queries were performed, distributed by 6000s.

Figure 4.17 evaluates the metric distance of each node to the closest copy of each data item. The average of these values is presented in Figure 4.17(a). In the "worst case" scenario, all algorithms appear to converge to the same value, although at different speeds. The probabilistic algorithm is the one that converged faster. This is expected, given that this is the algorithm that can shuffle more items on each query.

A comparison between the "worst case" and the "early" scenario suggests some interesting conclusions. Apparently, these shuffling algorithms are unable to bring a

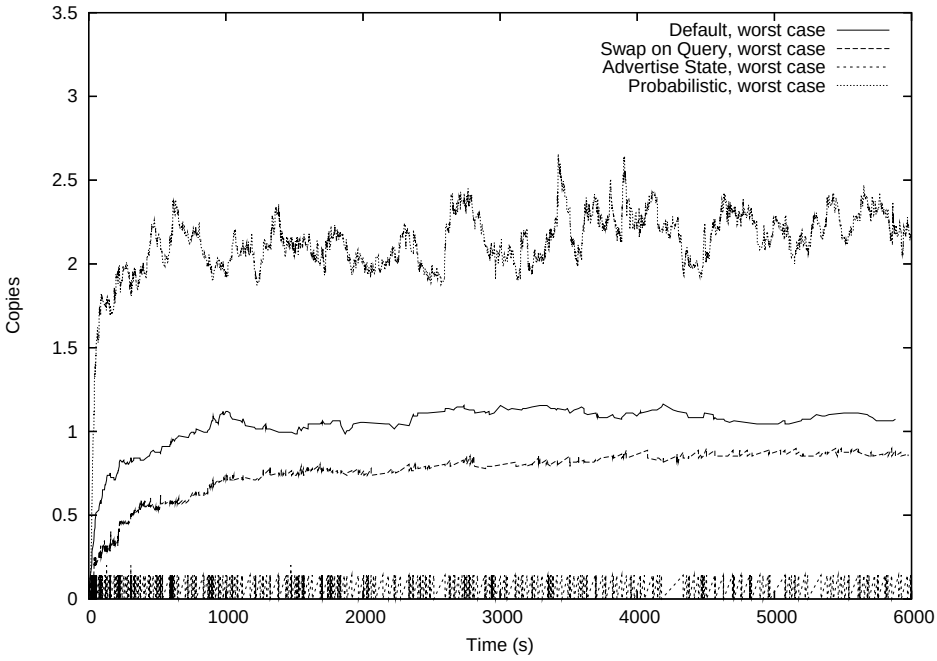
bad distribution to the average distance that can be achieved by the dissemination algorithms. In addition, starting from a good distribution, the shuffling algorithms tend to deteriorate it, although in a limited way. In "early" scenarios, the less aggressive algorithms tend to benefit for a longer time of the good initial distribution. We note that, as Section 4.7 will show, the number of items advertised is very low. Therefore, it is possible that the shuffling algorithms are relocating some of the excessive redundancy left during dissemination.

Figure 4.17(b) shows the progress of the standard deviation metric. As it can be seen, although presenting a lower average, probabilistic algorithms tend to provide a more uneven and unstable distribution of the items. Again, the convergence point is far from the minimal value exhibited at the end of the dissemination in the "early" scenario.

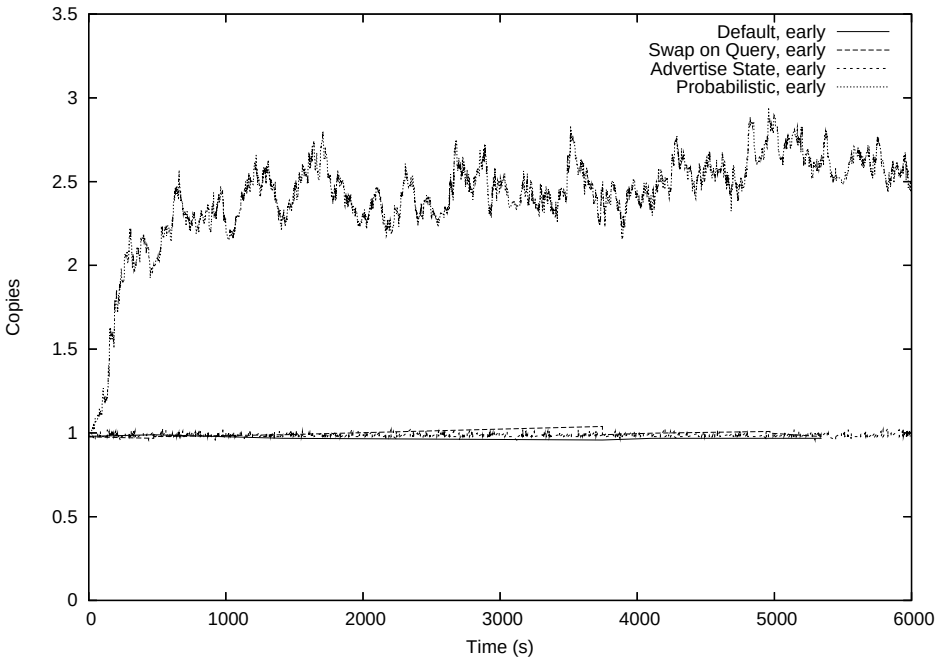
A second aspect considered in this evaluation addressed the impact of the algorithms in the number of copies. The results presented in Figure 4.18 show the standard deviation of the number of copies, calculated after every update to the storage space of any node.

As expected, the probabilistic algorithm performs a more uneven distribution of the number of replicas. However, there are two relevant aspects that deserve some attention: $i)$ like in the distance evaluation, the metric seems to stabilise. This suggests that the algorithm does not continuously benefits some data items by increasing its number of copies and $ii)$ a comparison between Figures 4.17(a) and 4.18(a) shows that, specially in the initial moments of the simulation, although presenting a more uneven distribution of the number of copies, the algorithm presents the lowest average distance.

The *Advertise State* algorithm is the only that effectively enforces the preservation of the number of replicas. The pattern it exhibits in Figure 4.18(a) is explained by the moment at which nodes replace items in their storage space during the exchange of *Swap* and *Swap Reply* messages. Figure 4.17(a) shows that the *Advertise State* and *Swap on Query* algorithms present a similar performance in the worst case but not in the



(a) Worst case test



(b) Early dissemination test

Figure 4.18: Variation of the standard deviation of the number of copies

early tests. This is inverted when we compare the stability of the number of replicas in Figure 4.18. An important conclusion that can be withdrawn from these results is that the number of copies should not be the unique factor for evaluating the quality of the distribution.

4.6 Comparison with Related Work

Table 4.2 extends to PADIS and the shuffling algorithms the comparison of the related work of Table 2.1 (see page 34). As it can be observed, PADIS provides a unique combination of features and requirements.

PADIS characteristics suggest that the algorithm fits in the *Improved Location Unaware* category given that the source of the data is unknown although it performs geographical distribution of the replicas. In the same class, the DAFN and DCG also provide geographical distribution without degrading the quality of the information, in contrast with the non-uniform algorithm. However, they rely on a predictable access pattern to the data.

Replica geographical distribution can also be found in some of the algorithms in the *Location Aware*, *Server Oriented* and *Trial and Error* classes. However, all these algorithms require devices to be aware of their location.

The shuffling algorithms associated with PADIS provide two alternatives for the re-allocation of the replicas. The *Shuffle On Query* and *Advertise State* algorithms use dedicated messages. Therefore, we consider that like DAFN and DCG, these algorithms explicitly address the problem using a coordination mechanism. On the other hand, the decision to shuffle is local to each node in the *Default* and *Probabilistic* algorithms and does not require the exchange of explicit messages. This characteristic is only shared by the CacheData algorithm, which assumes a small number of data producers and by the non-uniform dissemination algorithm.

PADIS is the only algorithm using broadcast messages to disseminate data and

Class	Protocol	Node Movement	Location Awareness	Access Prediction	Producers	Replica Refresh/ Leveraging	Geograph. Distr.	Message type	
								Proactive Dissem.	Query
Location Unaware	Simple Search	•			$<< n$			–	B
	Rumour Routing				n			rw	rw
	*-SAF	•		•	n	○		–	
	Aut. Gossipping	•		•	n	○		opp	–
Improved	Non-Unif				n		local/degrad.	B	–
	*-DAFN	•		•	n	○	coordination	–	
	*-DCG	•		•	n	○	coordination	–	
	PADIS + Shuffle	•			n	**○/•**	**local/coord**	**B**	**B**
Trial & Error	7DS	•			$<< n$			–	p2p/B
	Sailhan et al.	•			$<< n$			–	p2p
	Double rulings		•		n		deterministic	p2p	p2p
	GLS	•	•		n	○	deterministic	p2p	p2p
Owner	CacheData	•			$<< n$		local	–	p2p
	DCS	•	•		n	•		p2p	p2p
Loc. Aware	CachePath	•			n			–	p2p
	R-DCS	•	•		n	•	deterministic	p2p	p2p

•: feature of the algorithm ○: implicitly provided
n: approx. all nodes in the network, $<< n$: small number of nodes
rw: random-walk, B: broadcast, p2p: point-to-point message, opp: opportunistic, –: not applicable

Table 4.2: Comparison of the data dissemination algorithms surveyed with PADIS

queries. Pampa is expected to contribute to the limited impact of this option on the power consumption of the devices. However, we note that data is disseminated using point-to-point messages only by location aware algorithms. Contrary to PADIS, many of the remaining algorithms do not pro-actively disseminate replicas, thus degrading their resilience to node or localised failures in the network.

The description of the SAF, DAFN and DCG algorithms omits the algorithm used for locating the replicas. Therefore, it is not possible to perform a fair comparison between PADIS and the remaining. However, we note that the replica location unawareness that characterises *Location Unaware* and *Improved Location Unaware* algorithms suggests that the broadcast based, data retrieval algorithm presented in this thesis is a good candidate for providing this service.

4.7 Evaluation

We have implemented a prototype of our algorithms in the *ns-2* network simulator v. 2.28. The simulated network is composed of 100 nodes using IEEE 802.11 network interfaces at 2Mb/s. The nodes are uniformly deployed over a region with $1500m \times 500m$.

In all tests, data items have 300 bytes. Measurements have been taken in number of messages and number of data items stored at the nodes. Therefore, the size of the data item is only relevant for estimating the traffic generated at the network.

Simulations are composed by an initial dissemination of the data items followed by queries. The nodes performing the queries and the queried items are selected using an uniform distribution. No warm-up period is used. All values presented below average 100 independent runs, combining different dissemination and query operations and deployments of the nodes.

The evaluation is divided in two parts. Section 4.7.1 evaluates the data distribution capabilities of PADIS. The capability of the shuffling algorithms to correct a biased

distribution is the focus of Section 4.7.2.

4.7.1 Dissemination Algorithm

PADIS was evaluated by comparing its performance in different conditions. The evaluation uses two metrics. The "average distance of the replies" measures the distance (in number of hops) from the querying node to the source of the first reply received. The distance of a reply is 0 if the value is stored in the querying node. The "average number of transmissions" measures the resource consumption of the devices. The metric is defined for registrations and queries. Both sum the total number of messages of each type and divide it by the number of operations. For queries, both QUERY and REPLY messages (initial transmissions and forwarding) performed by all nodes in a simulation are accounted.

Runs are executed for 900s of simulated time. Each run consisted of 400 queries over a variable number of disseminated data items to be presented below. Data items are disseminated in time instants selected uniformly between 0 and 400s. Queries start at 200s and are uniformly distributed until the 890s of simulated time. The simulation ensures that only advertised records can be queried so that the evaluation does not become obfuscated by bogus queries. The *Default* shuffling algorithm was used in these tests. Recall that this algorithm does not require additional messages or increases the original message sizes.

4.7.1.1 Sensitivity to Different Network Configurations

The performance of the algorithm is affected by the number of nodes in the neighbourhood of each node, the storage size at every node and the number of items advertised in the network. To evaluate the effect of the variation of each of these parameters individually, we fixed a value for each in a baseline configuration. Each parameter was then individually varied keeping the remaining consistent with the baseline configuration.

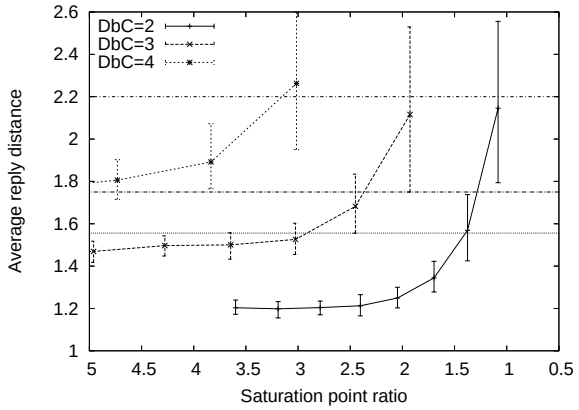
Network Density The number of neighbours was varied by configuring the nodes with different transmission powers while keeping the size of the simulated space constant. The transmission power was set such that transmission ranges varied between 150 and 325 meters, using the Free Space propagation model defined in the *ns-2* network simulator. A transmission range of 250m was settled for the baseline configuration.

Storage Size To evaluate the behaviour of the algorithm for nodes with different resources, we varied the number of items that can be kept by each node between 2 and 16. In the baseline configuration, each node makes available room for 10 records advertised by other nodes.

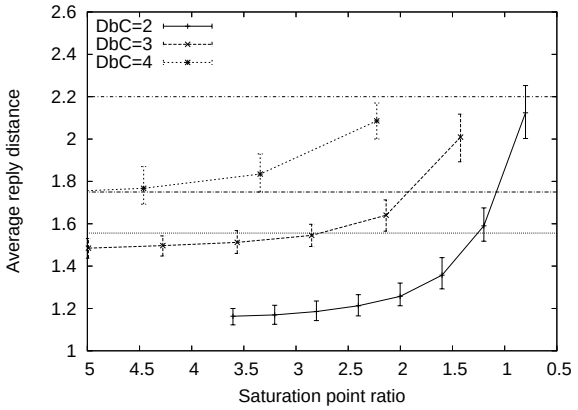
Number of Items The number of items advertised was varied between 50 and 800, at intervals of 50. Advertisements were uniformly distributed by the nodes. In the baseline configuration, 200 data items are advertised.

The different tests are harmonised by the *Saturation Point Ratio*. Recall from Section 4.3.2.3 that the saturation point ratio considers the storage space made available by the nodes on each cluster, what depends of the storage space made available by each node and of the number of nodes in the cluster. While the storage space of each node is defined by the *storage size* and *number of items*, which are well known on each simulation, the number of nodes in the cluster must be estimated.

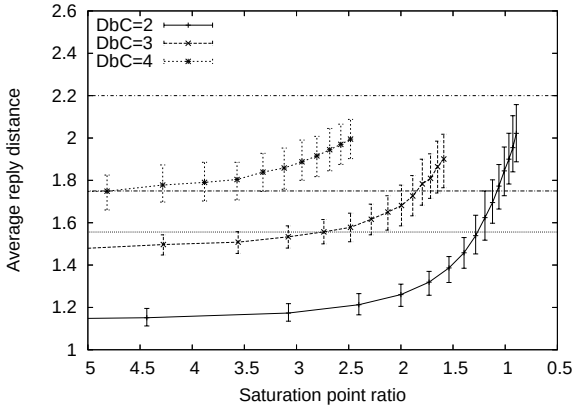
Because the network is limited in both the number of nodes and dimensions, the size of the cluster varies significantly between the nodes at the centre of the simulated space and those close to its borders. Knowing in advance that the estimation would be inaccurate, the number of nodes in each cluster was derived from the average number of 1 hop neighbours of each node and by the transmission radius r. The number of 1 hop neighbours was counted by averaging the number of nodes that receive each broadcast of a message on every simulation with the same transmission range. The nodes in a cluster were derived from the number of 1 hop neighbours by proportion to the increment of the area defined by a transmission radius of r to the radius required



(a) Variation of number of neighbours



(b) Variation of storage size



(c) Variation of number of items

Figure 4.19: Average distance of the replies

by $\frac{\text{DbC}+1}{2}r$, whose circle defines the cluster area.

Note that the baseline configuration has a saturation point ratio above one for all values of *DbC*, meaning that the storage capacity of the network has not been exceeded. Figure 4.19 shows the average distance of the replies in the simulations. Error bars show the highest and lowest average reply distance of a subset of 80% of the simulations for the scenario that excluded the 10% with higher and lower values. To facilitate the comparison with the theoretical model, the figures show the values of the theoretical expected average reply for every *DbC* tested given by function τ, previously presented in Section 4.3.2.2. The values of function τ for the *DbC* values tested are respectively $\tau(2) = 1.55(5)$, $\tau(3) = 1.75$ and $\tau(4) = 2.2$.

Independently of the *DbC* or of the parameter that was varied, all tests show that for large values of the saturation point ratio the average reply distance remains approximately constant. Also, to higher *DbC*, always correspond increasing average reply distances. These two observations confirm that the algorithm stores a limited number of copies of each data item. A desirable feature because it shows that the algorithm is capable to adapt to different loads by reserving space for other items and by providing replies close to the average expected distance. However, when the system is below the Saturation Point (SP), our algorithm exhibits a smaller average reply distance than the computed for the idealised model. This confirms the intuition developed in Section 4.3.4: our approach presents slightly more redundancy than expected. A side effect, confirmed by the plots is that the average distance of the replies becomes above the theoretical expected limited for saturation point ratios above one.

There are different reasons for the differences between the simulated and the theoretical results. Recall that function τ assumes an idealised networking environment. In particular, the following assumptions are not satisfied by the simulations:

Collisions and Partitions. In some wireless medium access protocols, like the IEEE 802.11 used in these tests, broadcasts are not acknowledged by the receivers. At some point, two transmissions may collide, preventing some nodes from receiving both of them. Although collisions may not affect the delivery of the message, they may influence the storage or propagation decisions of the nodes.

Because the number of nodes and network size have been kept constant in all simulations, partitions are more likely to occur when the transmission range decreases. The performance of the algorithm in a partitioned environment becomes harder to predict, as the probability of having a query replied will depend of the probability of the producer and querier nodes to be on the same partition. It should be noted that the error bars in Figure 4.19(a) grow with a decreasing number of neighbours. In addition, the bars are significantly smaller when the number of items or storage size varies. Therefore, the large variation of that set of results is attributed to partitions.

Concurrency. Pampa may not provide a sufficiently distinct hold time for two nodes to prevent them from taking a similar decision because they are not aware of the decision of the other. Examples of this event that resulted in redundant storage of some data items have been observed in the illustrative run presented in Section 4.3.4.

Network boundaries. As discussed in Section 4.3.2.1, some nodes may not have the closest copy of some data items at the expected theoretical limit of $\left\lceil \frac{DbC+1}{2} \right\rceil$ assumed by function τ. The most evident case are nodes close to the network boundaries which lack the continuation of the propagation chain to get the item eventually stored. An interesting aspect, that possibly justifies the increasing degradation of performance for higher values of *DbC* is that, for the same network, an higher *DbC* implies a small number of copies stored and, therefore, an increase in the number of nodes affected by this condition.

Besides influencing the average distance, the estimation of the number of nodes on each cluster is also more inaccurate at the network boundaries. Note that the smaller number of neighbours results in significantly smaller clusters to what correspond higher saturation point ratios.

Unlimited storage space. As discussed before, when the storage space of the nodes is fully occupied, nodes may need to discard some of the old items to make room for new ones. As we have seen, the algorithm does not provide a perfect replica distribution and introduces some excessive redundancy. Therefore, it should be expected that this effect appears when the saturation point ratio is above 1. When the replica selected for replacement is not a redundant one, the random replacing of replicas bias the item distribution because a cluster looses its replica of the item significantly increasing the distance of the nodes in the cluster to the closest copy.

Node deployment. A fundamental assumption of the saturation point estimator is that at every forwarding, there exists one node located precisely at the transmission radius from the previous source. Although Pampa favours the forwarding of the messages by the nodes more distant to the source of the previous transmis-

sion, it is only possible to assume that the node will be located at some distance lower or equal to the transmission radius. These smaller distances in propagation result in additional hops travelled by the REGISTRATION and QUERY messages and reduces the area covered by each cluster.

Independently of the factors above, Figure 4.19 shows that the algorithm exhibits the expected behaviour when the saturation point ratio approaches 1. As it becomes theoretically impossible to store all the items in the target *DbC*, the average reply distance increases. Still, the system continues to provide acceptable results, suggesting that in the majority of the cases copies are found within only a few hops in excess of the theoretical limit.

Figure 4.20 present the same results than Figure 4.19(c) but using the number of advertised items in the x axis. It confirms that the selection of an adequate *DbC* is crucial for a good performance of the algorithm. It can be seen that the average distance of the replies for the different values of *DbC* tend to approximate as the occupancy of the storage space increases. In particular, the lines for *DbC*=2 and *DbC*=3 intersect around the Saturation Point for *DbC*=2. This is an expected behaviour of the algorithm since that above the Saturation Point it is not theoretically possible to store all information in the nodes in the expected range. Therefore, it is preferable to extend the size of the cluster, increasing its storage capacity, so that the algorithm can continue to apply its fine dissemination capabilities.

4.7.1.2 Message Overhead

A second aspect that is relevant to assert on the performance of the algorithm is the total number of messages required for disseminating and retrieving a data item. The following subsections analyse individually each of these operations using the number of transmissions per operation as the metric. The goal is to show that geographical distribution provides a good tradeoff, compensating the cost of the initial dissemination of the data items with the gains given by the reduced number of messages required

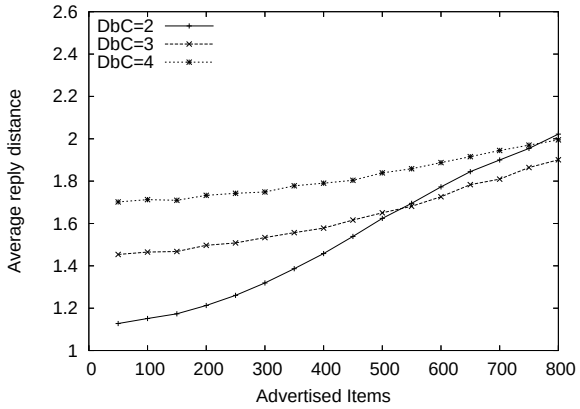


Figure 4.20: Average distance of the replies with variation of number of advertised items

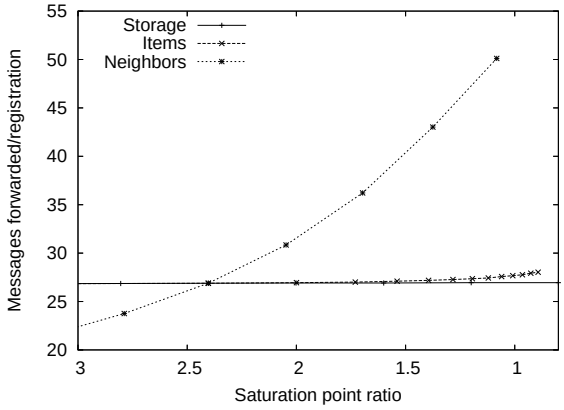


Figure 4.21: Transmissions per registration

for data retrieval. As expected, Pampa plays an important role in the reduction of the number of messages required for data dissemination and retrieval.

4.7.1.2.1 Dissemination PADIS broadcasts a message to disseminate each data item. In consistency with results for Pampa, the number of transmissions generated by dissemination is strongly dependent of the network density, what can be confirmed in Figure 4.21.

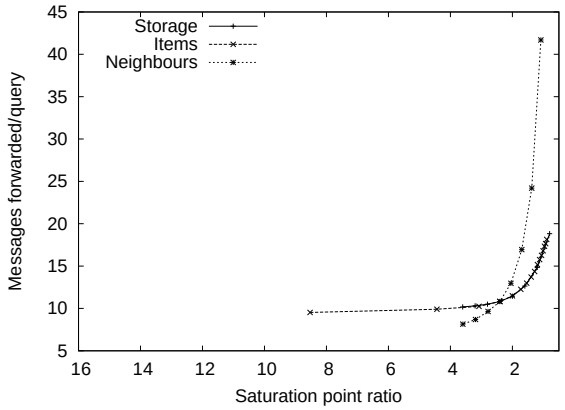
The figure shows that for variations of the storage space or number of items, the number of messages per operation is approximately the same until close to the Saturation Point. Interestingly, the number of messages required for the dissemination increases for lower values of the saturation point ratio in the tests that varied the num-

ber of items. We attribute this variation to the bias applied to the hold time when the occupancy ratio of the nodes is above some threshold (see Section 4.3.3.1). Although it promotes an even distribution of the items, the additional hold time also prevents Pampa from using the most adequate (distant) nodes for message propagation. Therefore, additional retransmissions will be required on each broadcast. This variation is not visible when the storage space is varied due to the large granularity of the bias to the hold time: when the storage space is small, all nodes rapidly commute from the regular Pampa's hold time to the biased version, attenuating any benefit of this delay.

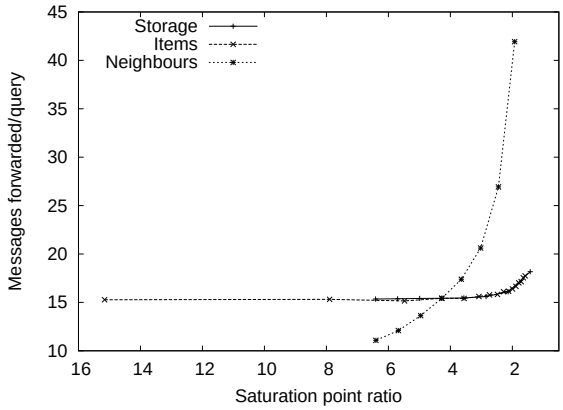
4.7.1.2.2 Queries The average number of messages per query is presented in Figure 4.22. It is interesting to notice that, for each value of *DbC*, the lines that capture the behaviour of the system with the size of the storage space and with the number of items practically overlap. This confirms that, as expected, when the system is below the Saturation Point, none of these factors influences the number of messages transmitted per query. A feature that can be clearly attributed to Pampa.

Additionally, we compared the growing ratios of the curves for the average distance of the replies and for the number of messages forwarded/query in both scenarios. The difference between these ratios is less than 2% when the storage space is changed and less than 7% when the number of items changes. These small values show that the growing of the average distance implies an almost linear grow of the number of messages. In practise, this confirms the efficiency of our mechanism for adapting the TTL of the first QUERY message, which prevents the frequent retransmission of the queries using a full broadcast.

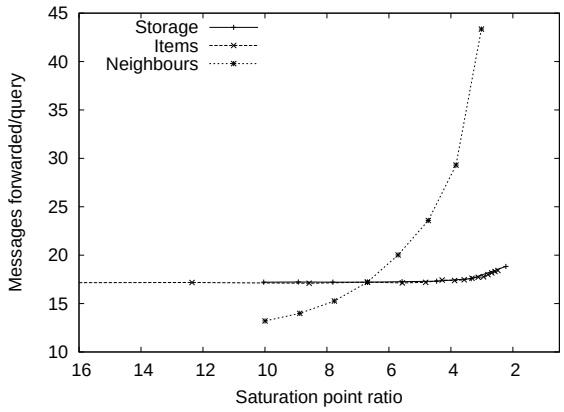
On the other hand, we expect the number of messages to drop significantly when the node density increases because we benefit from the properties of Pampa, which adapts the proportion of nodes retransmitting a message to the network density. Comparing results depicted in Figures 4.19(a) and 4.22(c), it can be seen that although the distance of the replies tends to stabilise with the grow of the network density, the number of messages continues to diminish. Here, the difference between the ratios is higher



(a) *DbC*=2



(b) *DbC*=3



(c) *DbC*=4

Figure 4.22: Transmissions per query

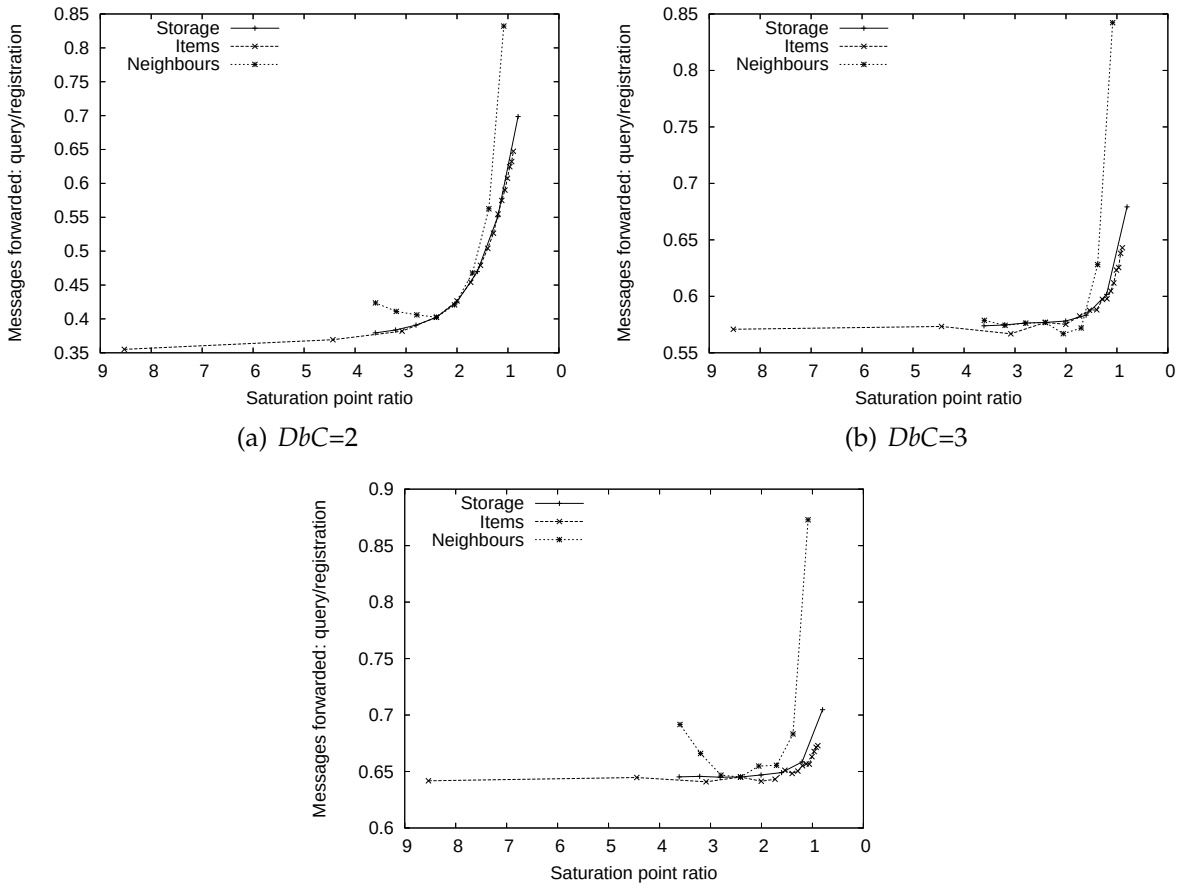
than 36%.

These results show that network density is the dominating factor of the number of messages per query what confirms previous results for Pampa.

4.7.1.3 Attenuation of the Dissemination Cost

The transmissions presented in Figure 4.21 are those required by Pampa to flood the entire network. This would be the number of messages required in the majority of the cases by queries if no early dissemination of the items was performed. A comparison with the number of messages required for data retrieval is depicted in Figure 4.23.

Results show that queries always require a non-negligible lower number of mes-



(a) *DbC*=2

(b) *DbC*=3

(c) *DbC*=4

Figure 4.23: Ratio of transmissions per query/transmissions per registration

sages than dissemination. This suggests that the cost of dissemination can be absorbed after a small number of queries. Not surprisingly, the gains of our algorithm are smaller when the *DbC* is higher or with a lower saturation point ratio. This is explained by noticing that the number of messages required for a registration is constant and independent of the *DbC*. On the other hand, an higher *DbC* implies an expected higher average reply distance, thus requiring an higher TTL of the first QUERY message.

We note that these results are strongly dependent of the simulated scenario. Absolute results determining the number of queries after which the use of this algorithm becomes advantageous in comparison with a strict pull model, where data items would not be disseminated in advance will always depend of the simulated environment. However, our algorithm provides the additional advantage of replicating the data

items, making the network more robust against failures and departures of nodes.

4.7.2 Shuffling Algorithms

This section compares the performance of the four shuffling algorithms. Two classes of tests were defined. *Convergence* tests evaluate the capability of the algorithms to improve an initially biased distribution. *Mitigation* tests measured the capability of the algorithms to attenuate the effects of the continuous node movement.

The test-bed for shuffling algorithms is similar, in number of nodes and network dimension, to the one used for the dissemination algorithm. All tests begin with a dissemination phase, where a variable number of items is advertised. Nodes do not move during the dissemination phase.

In *Convergence* tests, the initial dissemination is biased by having the nodes to move during 300s with an average speed of $10m/s$. Nodes then stop and perform 1000 queries, distributed by 2000s and randomly distributed by the data items using an uniform distribution.

In *Mitigation* tests, nodes do not stop after the dissemination phase. Tests were performed using average speeds of $2m/s$ and $5m/s$. The number of queries and the duration of the query phase is similar to the one used in convergence tests.

The maximum storage space of each node was set to 10 items, excluding the items it produces. New storage space entries are created only during the dissemination phase. That is, the shuffling algorithms may only replace existing entries on the node's storage space.

We experimented two movement models frequently referred in the literature. Both are briefly described here for self-containment of this text. In the Random Waypoint Movement Model (Johnson & Maltz, 1996), nodes select a random location in the simulated region and move in a straight line to it at a speed randomly defined between a minimum and a maximum value. When the location is reached, nodes stop for a pre-

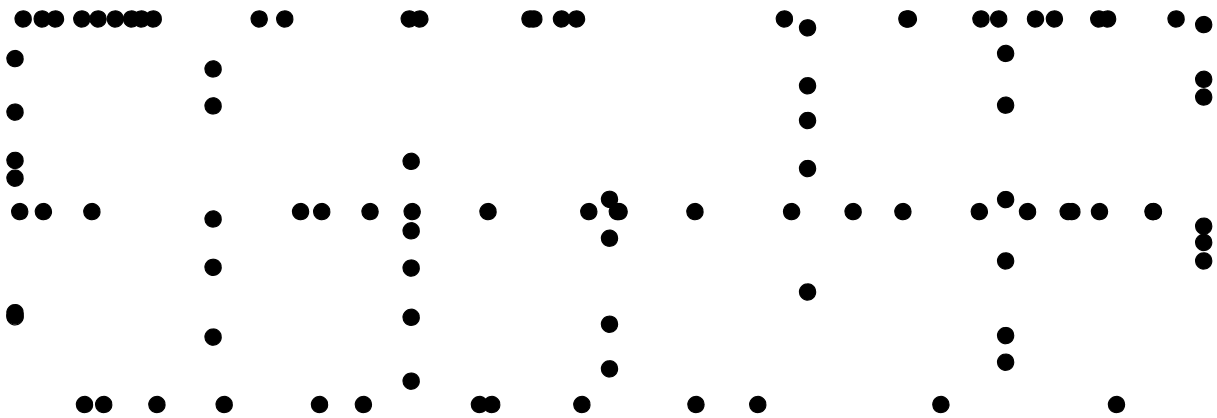


Figure 4.24: Snapshot of the Manhattan Grid movement model with 7 by 3 streets

defined amount of time and initiate a new cycle by selecting another random location.

In the Manhattan Grid movement model the region is traversed by vertical and horizontal streets. Nodes move over the streets with a speed that can be varied within some interval. When reaching an intersection, nodes decide to turn with some probability.

We note that, although not moving during the dissemination phase, nodes are initially deployed according to the movement model being tested. Both movement models consider pause times for the nodes. However, in our simulations, the nodes were configured for not stopping. The Random Waypoint movement model was used in convergence tests. Mitigation tests were run with both movement models. To prevent anomalies on the average speed such as those identified in the Random Waypoint (Bettstetter *et al.*, 2003), minimum and maximum speeds were always set $1m/s$ respectively below and above the average. In the Manhattan Grid movement model, the grid is composed of 7×3 streets spaced 250m. A snapshot of a node deployment in a simulation using the Manhattan movement model with the configuration above is presented in Figure 4.24.

To simulate different network conditions, we have varied the transmission range of the nodes, the number of data items advertised and the *DbC* constant. Three transmission ranges were tested: 200m, 250m and 300m permitting to evaluate algorithm's performance in sparser and denser networks. Nodes advertise 100, 400 or 700 items,

DbC	Transmission Range	Items	Saturation Point Ratio
2	200	100	4.30
		400	1.37
		700	0.95
	250	100	6.03
		400	1.92
		700	1.33
	300	100	7.87
		400	2.51
		700	1.74
4	200	100	11.93
		400	3.80
		700	2.63
	250	100	16.75
		400	5.33
		700	3.70
	300	100	21.87
		400	6.96
		700	4.83

Table 4.3: Saturation Point Ratios for Convergence tests

thus producing different storage space occupancy ratios. Finally, we tested *DbC* values of 2 and 4. Like in the evaluation of the dissemination algorithm, all results average 100 different runs with independently generated sets of dissemination and queries and node movements. To facilitate comparisons with the previous tests, Table 4.3 shows the corresponding saturation point ratio estimations for each combination tested in convergence tests.

For mitigation tests we opted by configurations with a smaller transmission range to represent sparser networks were the need to move the items is higher. Table 4.4 shows that the node distribution of the movement models affects the saturation point ratio. This is easily observed by noticing that, for the same configuration, the saturation point ratio for the Manhattan Grid movement model is approximately 60% of the value for the Random Waypoint movement model. Recall that the ratio is estimated by counting the number of nodes that receive each broadcast message, what indicates that each node has a significantly lower average number of neighbours in the Manhat-

DbC	Transmission Range	Items	Saturation Point Ratio			
			RWP		Manhattan	
			$2m/s$	$5m/s$	$2m/s$	$5m/s$
2	200	400	1.36	1.37	0.78	0.78
		700	0.94	0.94	0.54	0.54
	250	700	1.31	1.32	0.81	0.81
4	200	700	2.60	2.61	1.50	1.50

Table 4.4: Saturation Point Ratios for Mitigation tests

tan Grid movement model. The table suggests that node movement in the Random Waypoint model did not affect the node density, given that the saturation point ratio is similar.

The performance of the algorithms is compared using two metrics. The "average distance" considers the geographical distribution of the replicas of the data items. It is calculated by measuring, for each node, the distance to the closest copy of each item. The metric uses the average of all the distances. By itself, the average may not be a good estimator given that the algorithms may privilege some of the items and provide them with a very low average distance. Therefore, the standard deviation is presented to confirm the quality of the geographical distribution. Together, lower averages and standard deviations indicates a better performing algorithm.

We note that given the characteristics of the dissemination algorithm, values too small are not achievable or desirable. Theoretically, the average distance and the standard deviation should be correlated, for each *DbC*, with function τ (defined in Section 4.3.2.2) and with the transmission range. However, the differences to the theoretical model noted in the evaluation of the distribution algorithm are also applicable to the case where nodes move. For example, the irregular distribution of the nodes does not guarantee each next hop to be found exactly in the limit of the transmission range of the node. In addition, we must also consider the bias introduced by the node movement model.

A second aspect considered in the evaluation of the algorithms was the number of replicas of each data item. We note that in all tests the average is constant given that

the storage space and the number of items is also constant. The metric used was the standard deviation of the number of copies of each data item. The average is plotted to allow a visual estimation of its proportion to the standard deviation. Again, the ideal value of the standard deviation may not be zero. This is because the most adequate number of replicas will depend of their location. For example, data items with replicas stored at the nodes closer to the centre of the simulated space may require a smaller number of replicas.

The evaluation compares the performance of the algorithms with each other and with the state that was initially found after the dissemination phase. To observe the evolution of the algorithms with time, snapshots of the state and location of the nodes are taken at periodic intervals of each test. The first snapshot occurs at the end of the dissemination phase. Plots represent this moment as time 0. The second snapshot is taken 300s after, thus being coincident with the moment at which nodes stop in convergence tests and queries start. A snapshot is taken again every 400s, with the last one being coincident with the end of the simulation.

4.7.2.1 Probability of Insertion

The *Probabilistic* algorithm use two constants: p_{rep} dictates the probability of a transmitting node to replace a record in the HERALD message; p_{ins} the probability of an item being inserted when the node has some storage space entries marked with the *replace* flag set to true. In this evaluation, we assume that p_{rep} should be necessarily small: if records are frequently replaced, the TFS field is always kept at a low value. This would prevent nodes more distant from receiving records with an high TFS and therefore, from inserting the data items. In the simulations, we used a constant probability of 0.2 for p_{rep} as we considered that it represents a good tradeoff between the refreshment of the records in the message and their continuation for being used in latter updates.

A definitive value for p_{ins} however is harder to establish. Intuitively, an higher value of p_{ins} should provide a more aggressive update policy, that converges more rapidly for the leveraging of the items distribution. However, if p_{ins} is excessively

high, different neighbour nodes, receiving the same message will store the same item, creating an undesirable level of redundancy for the item in some locations. Figure 4.25 compare the performance of two versions of the *Probabilistic* algorithm using $p_{ins} = 0.5$ and $p_{ins} = 0.8$ in the convergence tests.

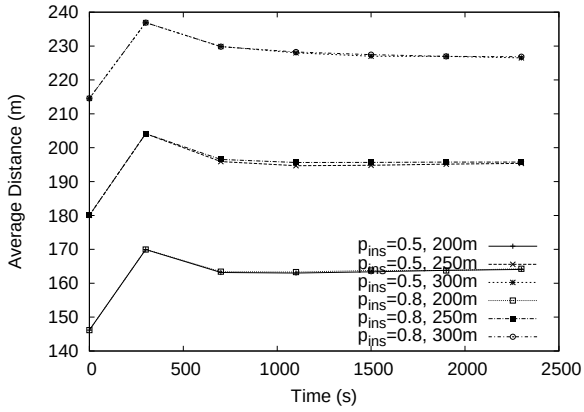
Contrary to our expectations, the performance of the algorithms is very similar. In some cases, $p_{ins} = 0.8$ presents an average smaller by a few meters in the earlier stages of the tests. The difference tends to be attenuated later. In the Probabilistic algorithm, a 30% difference in the probability may provide an almost negligible contribution to the speed at which the algorithm improves the item distribution. Such a smaller difference suggests that a comparison of both variants with the remaining algorithms would not provide a valuable contribution to the evaluation. Evaluation of convergence and mitigation tests omit the results for $p_{ins} = 0.5$.

The following analysis will also omit results for 250 and 300m when *DbC*$= 4$. This is justified by observing that in Figure 4.25 there is a small increase of the average distance between the end of the dissemination phase ($t = 0s$) and the end of the movement period ($t = 300s$). With *DbC*$= 4$, the distance between the replicas at the end of the dissemination phase is close to what is achieved after the random movement of the nodes in a closed region with the predefined dimension.

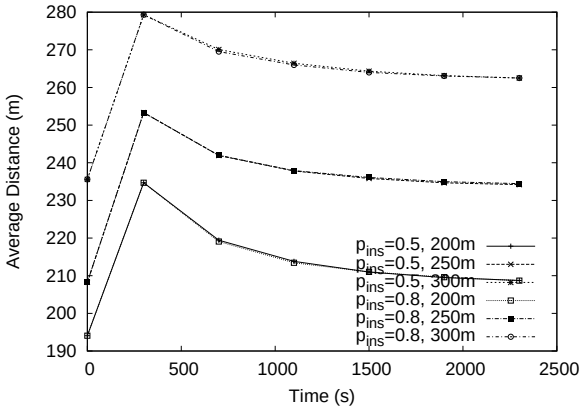
4.7.2.2 Convergence Tests

The simulation results for the convergence tests are presented from Figure 4.26 to Figure 4.29. The figures show that, in general, the four algorithms reduce the "average distance" metric, in a tendency to correct the bad distribution that can be found after nodes have moved. The standard deviation and the average curves exhibit a similar pattern, suggesting that the leveraging is being equally distributed by the data items.

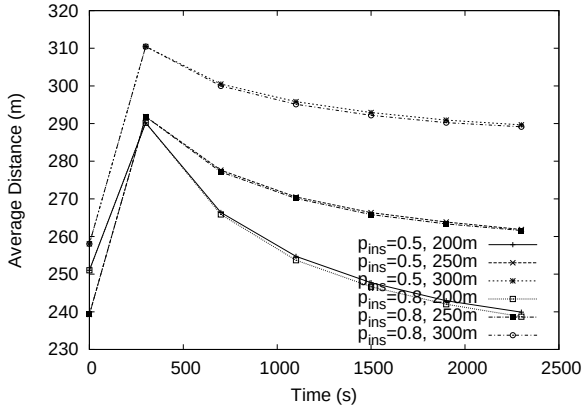
Not surprisingly, the *Probabilistic* algorithm is the algorithm that reacts faster to the biased distribution. This is due to the number of data items that can be shuffled on each query. Algorithms that keep the number of replicas constant will shuffle at most



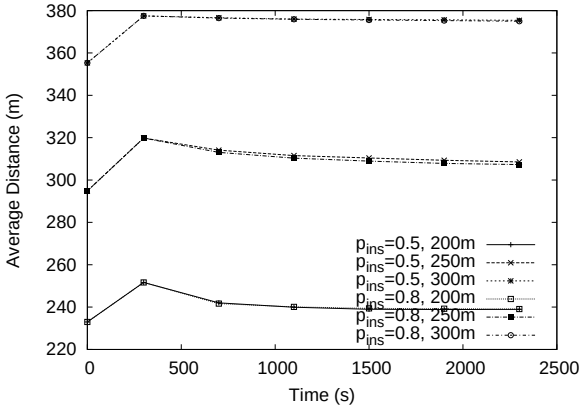
(a) *DbC*=2, 100 items



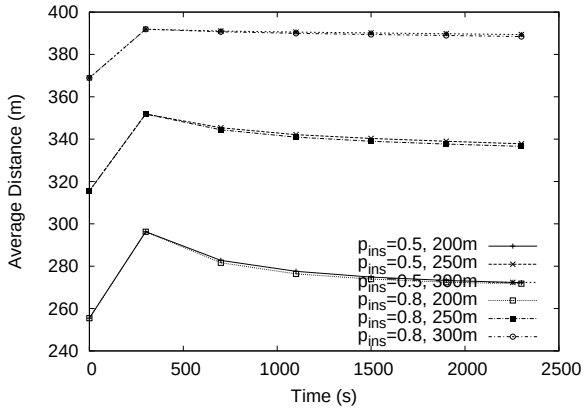
(b) *DbC*=2, 400 items



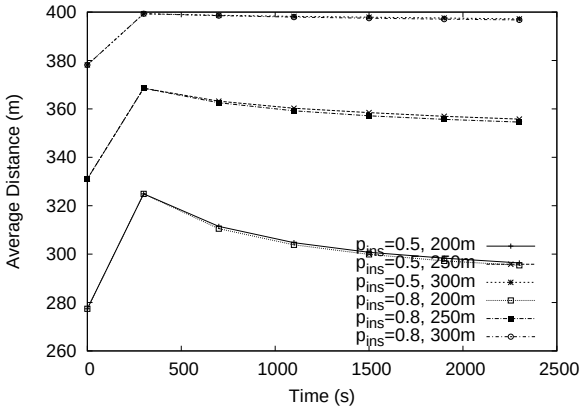
(c) *DbC*=2, 700 items



(d) *DbC*=4, 100 items

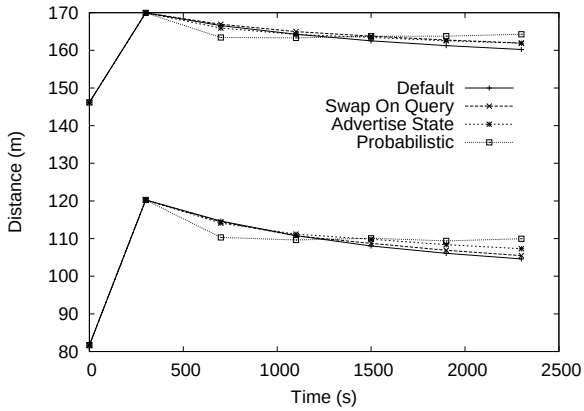


(e) *DbC*=4, 400 items

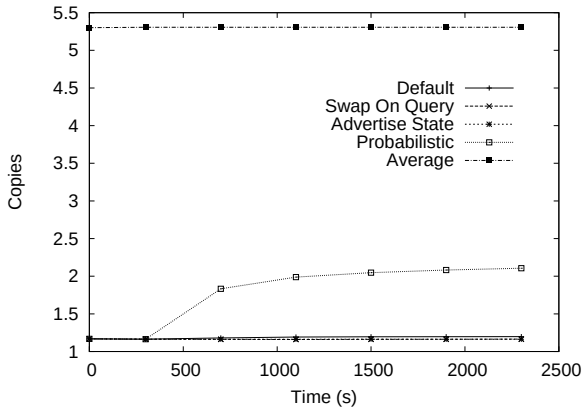


(f) *DbC*=4, 700 items

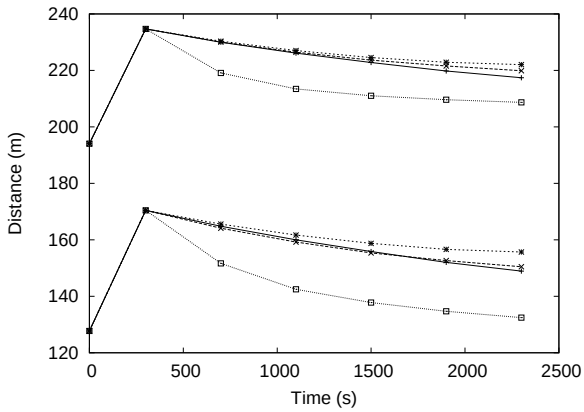
Figure 4.25: Average distance of the nodes to the closest replica for two configurations of the Probabilistic algorithm



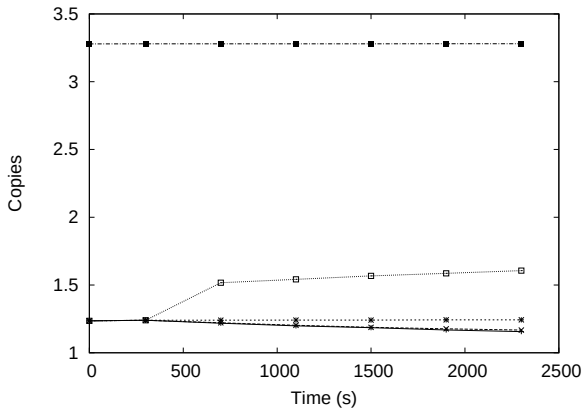
(a) Distance, 100 items



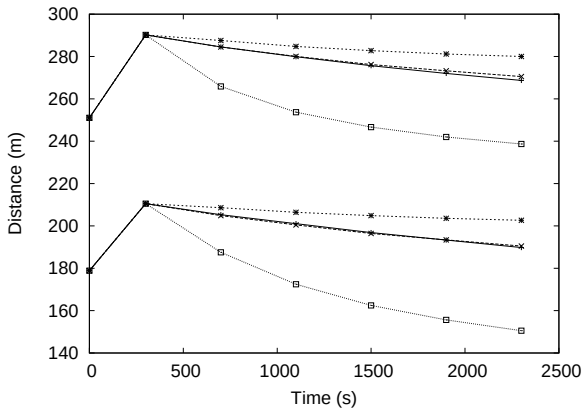
(b) Copies, 100 items



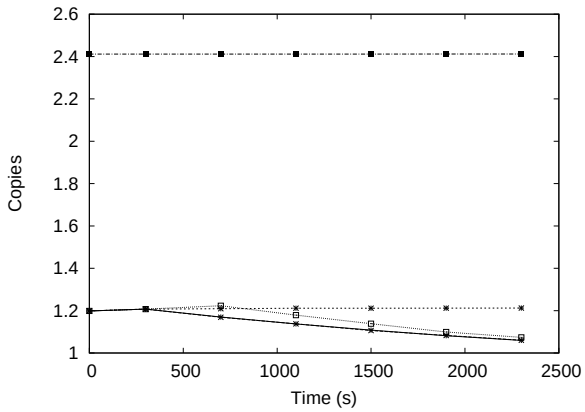
(c) Distance, 400 items



(d) Copies, 400 items

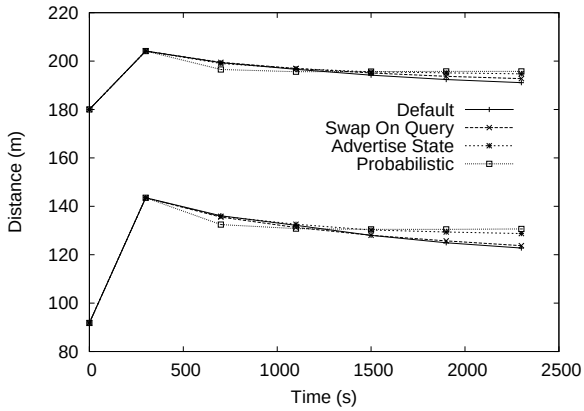


(e) Distance, 700 items

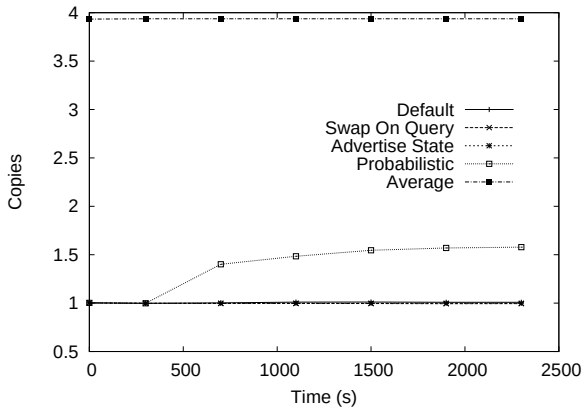


(f) Copies, 700 items

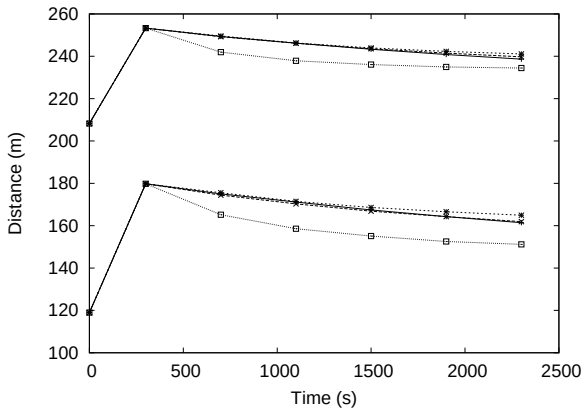
Figure 4.26: Convergence tests when *DbC*=2 and transmission range=200m



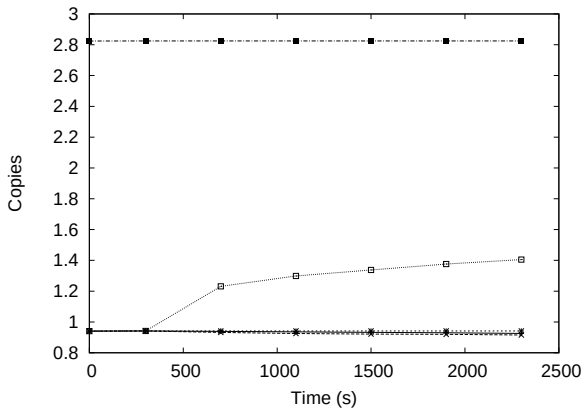
(a) Distance, 100 items



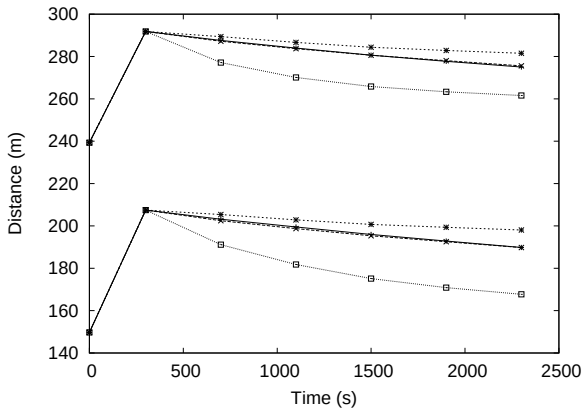
(b) Copies, 100 items



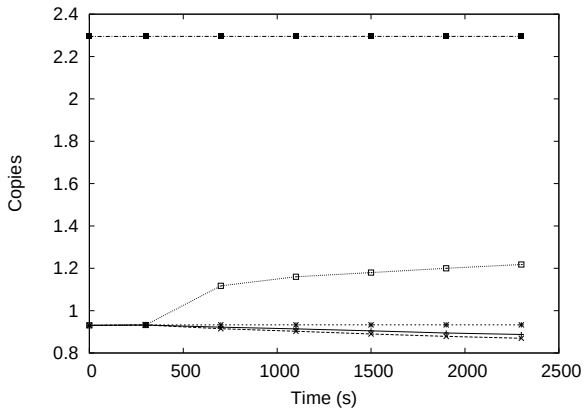
(c) Distance, 400 items



(d) Copies, 400 items

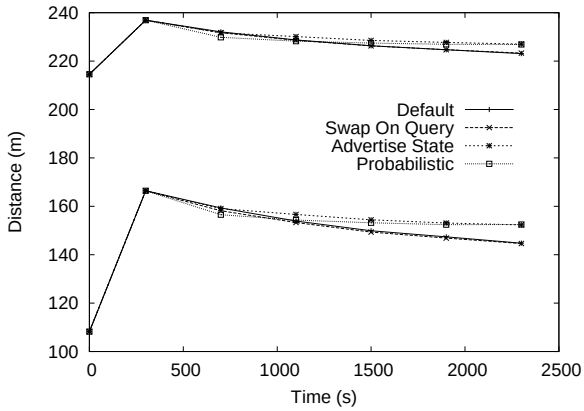


(e) Distance, 700 items

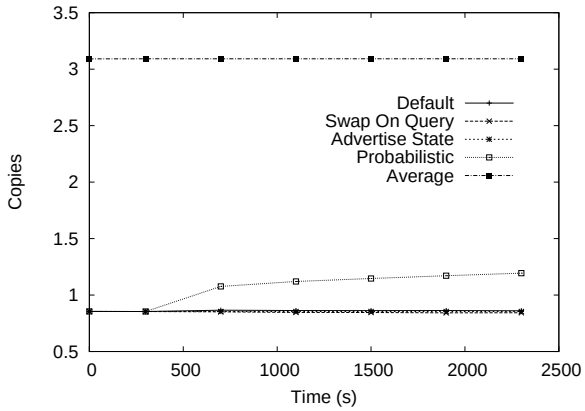


(f) Copies, 700 items

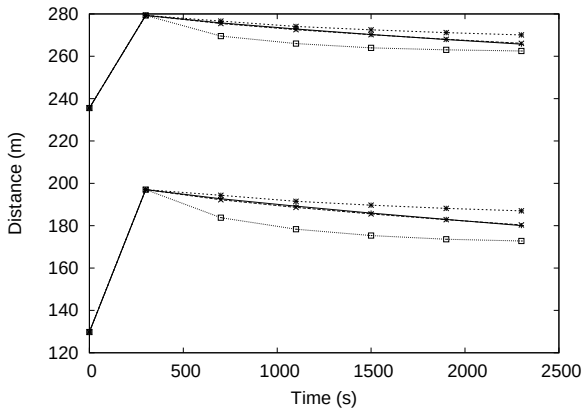
Figure 4.27: Convergence tests when *DbC*=2 and transmission range=250m



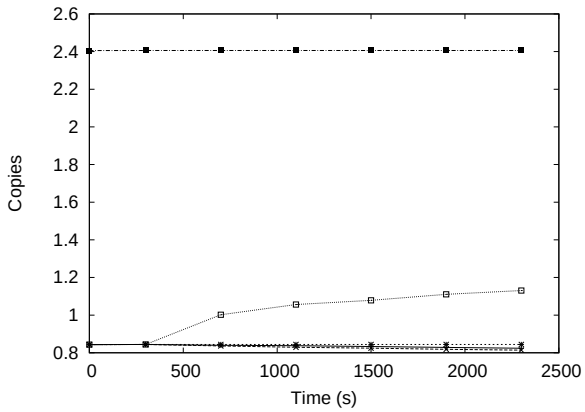
(a) Distance, 100 items



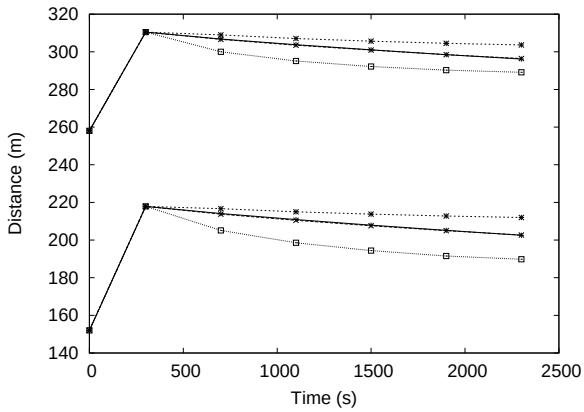
(b) Copies, 100 items



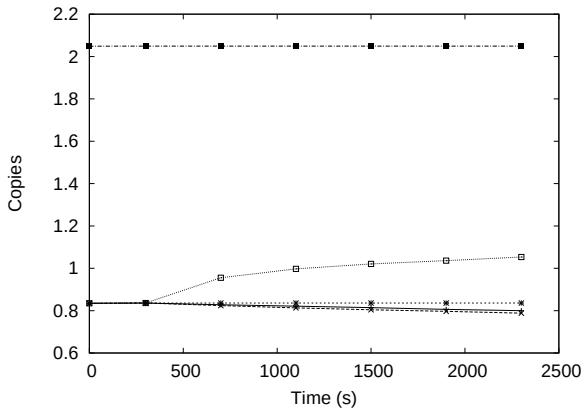
(c) Distance, 400 items



(d) Copies, 400 items

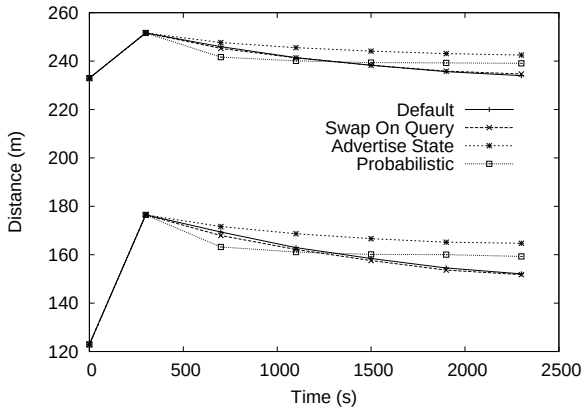


(e) Distance, 700 items

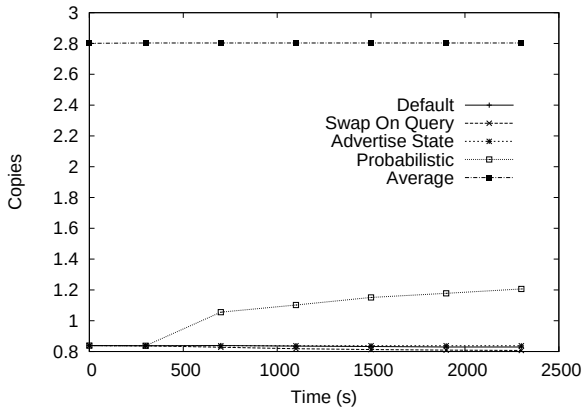


(f) Copies, 700 items

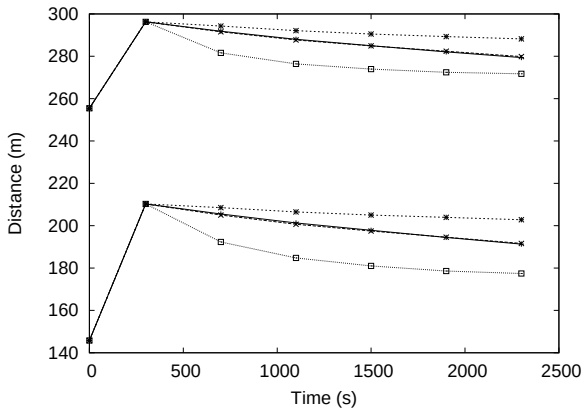
Figure 4.28: Convergence tests when *DbC*=2 and transmission range=300m



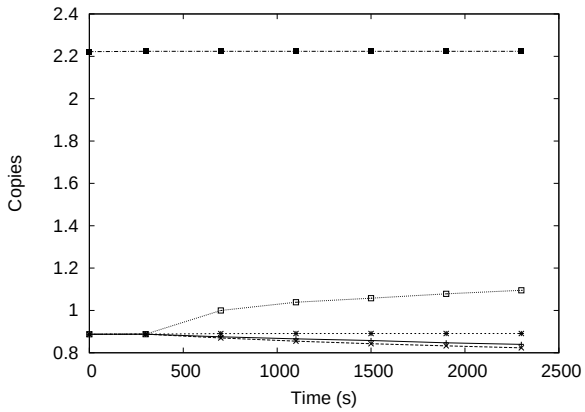
(a) Distance, 100 items



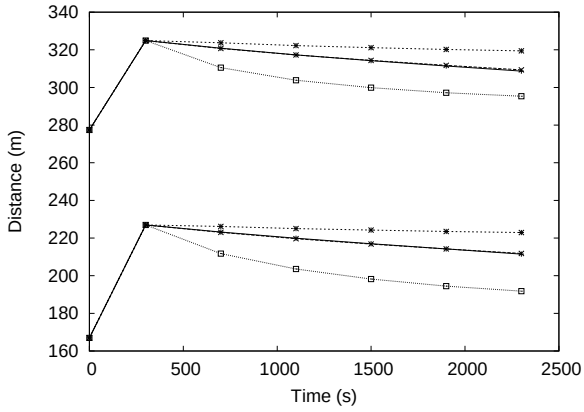
(b) Copies, 100 items



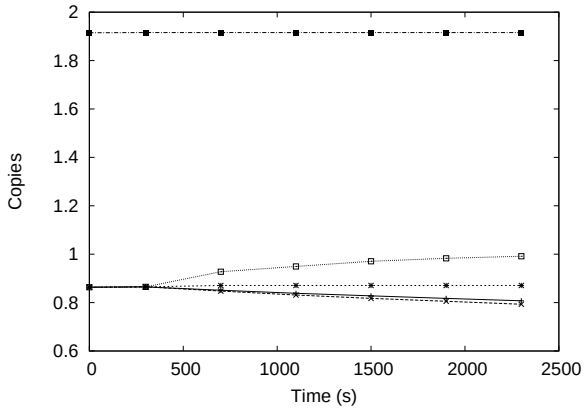
(c) Distance, 400 items



(d) Copies, 400 items



(e) Distance, 700 items



(f) Copies, 700 items

Figure 4.29: Convergence tests when *DbC*=4 and transmission range=200m

two items. The default algorithm at most one while an undefined number of data items can be shuffled when the *Probabilistic* algorithm is used. The *Advertise State* algorithm is penalised by the frequent impossibility of swapping data items. As it can be observed, the algorithm is particularly slower than the remaining when the average number of copies is smaller. These are the cases where the probability of receiving a reply from the producer of the item is higher.

In general, algorithms converge to some value between the average distance at the end of the dissemination phase and at the end of the movement phase. That is, although attenuating the impact of the node movement, the algorithms do not bring the average distance to the low value that is achieved by the dissemination algorithm. We attribute this behaviour to two interrelated factors.

1. The number of queries that may result in a shuffle of some data item decreases with time. As a result of their operation, all algorithms progressively increase the probability of finding a data item within the expected distance. Therefore, the number of queries that will reach the conditions required by each algorithm for shuffling is reduced. Not surprisingly, the "curve" that results from this effect is more visible in the *Probabilistic* algorithm, the only one that can shuffle more than one data item unrelated with the query.

2. Algorithms may be achieving a satisfactory distribution with a distance higher than the one provided by the dissemination algorithm. This is expected given that it was already shown that the dissemination algorithm induces excessive redundancy. This claim is supported by noticing that the point to where the items appear to converge is higher when the number of items is lower, that is, when there is more space at the nodes for storing the redundant copies.

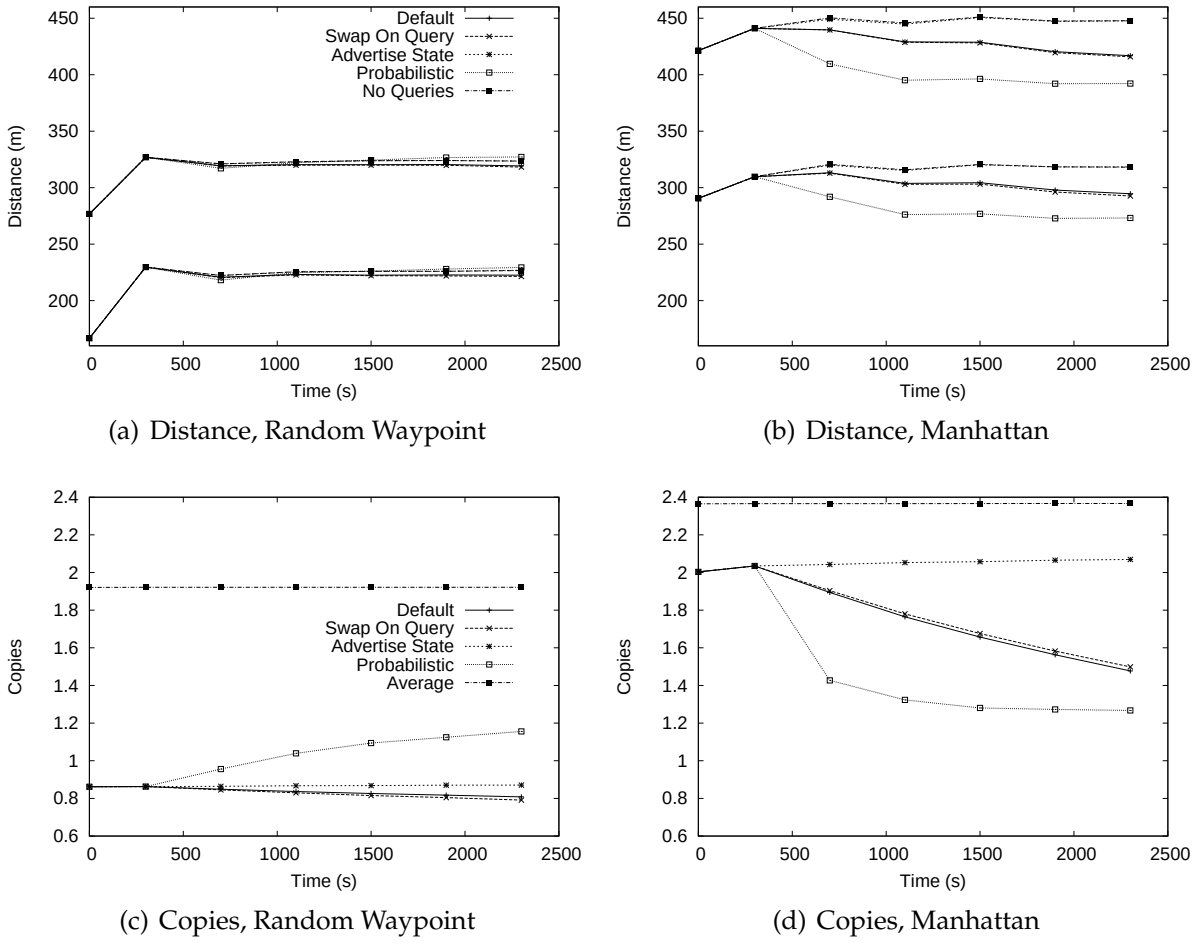
Results for the different algorithms are closer when there are less items advertised. In particular, with 100 items, all algorithms present marginal differences between them, with advantage for the algorithms that do not piggyback HERALD messages on

queries. These algorithms tend to converge to an average distance lower than the remaining.

An observation of the graphics for the number of copies show that the algorithms use different methods for achieving their goals. In particular, the *Probabilistic* algorithm rapidly deteriorates the standard deviation of the number of copies while decreasing the average distance. This indicates that the number of replicas of the items is less homogeneous in the *Probabilistic* algorithm. This can be attributed to $i)$ a deviation of the algorithm, if the preferred items are selected due to some possible random criteria; or $ii)$ an improvement of the geographical distribution that takes into account the location of each replica. Results suggest that heterogeneity can be attributed to improvements in the geographical distribution.

In general, the random algorithm presents the lowest average distance, even when the standard deviation of the number of copies is significantly higher, what suggests that the algorithm is improving the distribution. That is, the algorithm achieves a better performance with a bigger deviation in the number of copies than the remaining, which keep the number of copies close to what was determined during the dissemination. We interpret this result as a confirmation that the algorithm helps to remove the additional redundancy that was already shown to be created by PADIS.

A particular case is observed in Figures 4.26(e) and 4.26(f), which is the only test performed with a Saturation Point Ratio below 1. Recall that in this case, the algorithm is unable to provide its properties given that the storage space made available by the nodes is not sufficient for keeping the desired number of replicas. In this test, the *Probabilistic* algorithm reduces the standard deviation of the number of copies, showing that the grow that is visible in the remaining scenarios can be inverted in some cases. As a result, the algorithm is able to improve the average distance to a value that is clearly below the remaining and even below the one that was achieved by the dissemination algorithm.



(a) Distance, Random Waypoint

(b) Distance, Manhattan

(c) Copies, Random Waypoint

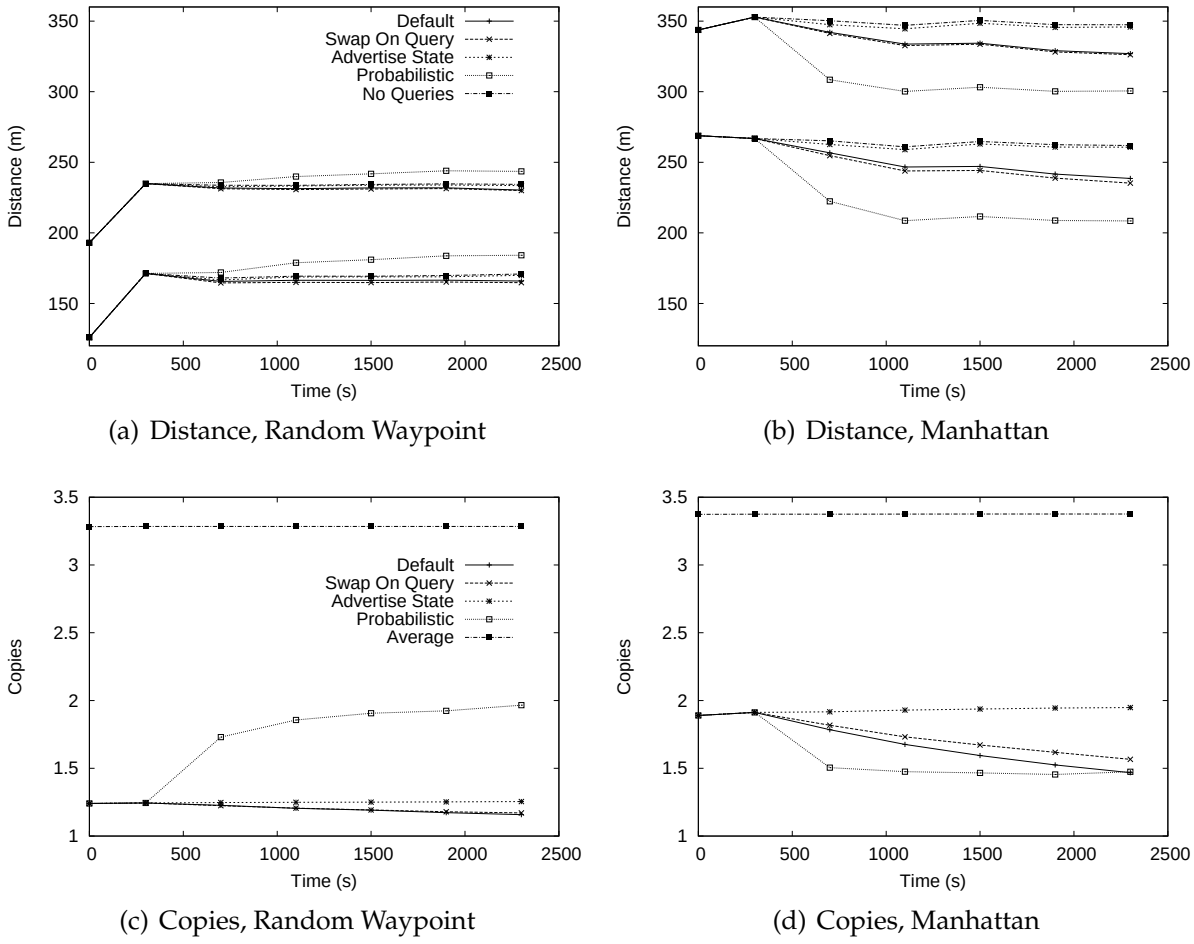
(d) Copies, Manhattan

Figure 4.30: Mitigation tests with 200m transmission range, $2m/s$ average speed, 700 items and *DbC*=4

4.7.2.3 Mitigation Tests

In the mitigation tests, the shuffling algorithms were compared using the Random Waypoint and the Manhattan Grid movement models. The simulation results are depicted from Figure 4.30 to Figure 4.37. The figures have been arranged by average speed and from the highest to the lowest saturation point ratio. The plots showing the metric distance of the nodes to the closest copy of each item also present the results for control tests. In these tests nodes move but no queries are made. Given that in the remaining, shuffling always results from queries, the tests evidence what would happen if no shuffling was used.

Recall from Table 4.4 that using the same configurations, the Manhattan Grid



(a) Distance, Random Waypoint

(b) Distance, Manhattan

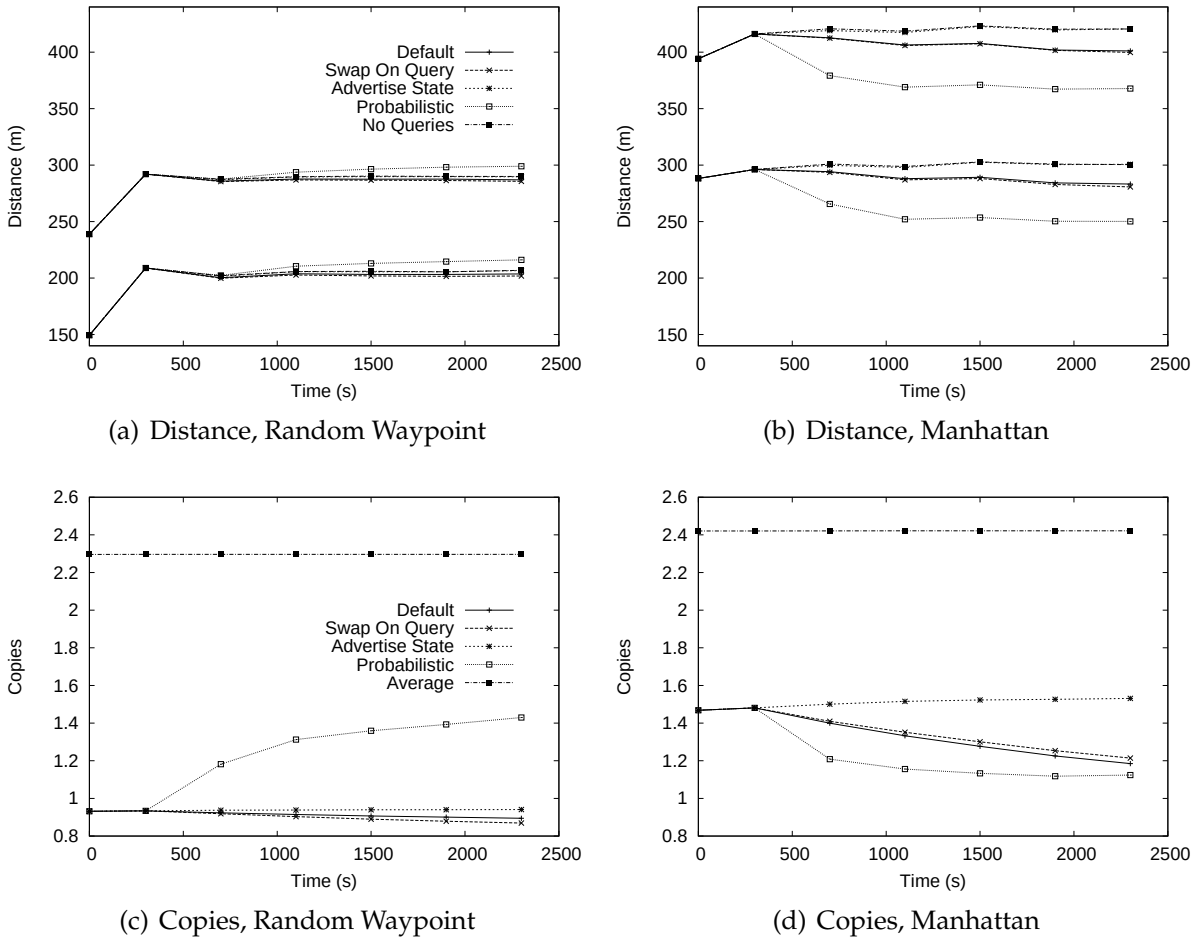
(c) Copies, Random Waypoint

(d) Copies, Manhattan

Figure 4.31: Mitigation tests with 200m transmission range, $2m/s$ average speed, 400 items and *DbC*=2

movement model tests present a saturation point ratio that is approximately 60% of the value for the Random Waypoint tests. The impact of the saturation point ratio is clearly visible in side-by-side comparisons of both movement models.

The average distance of both movement models at the end of the dissemination phase (in the plots corresponding to time 0s) and at the beginning of the query phase (time 300s) present a similar pattern. At the end of the dissemination phase, and with *DbC*=2, the Random Waypoint movement model presents an average distance that is $55\% \approx 60\%$ of the average distance exhibited by the Manhattan Grid movement model. At the beginning of the query phase, the distributions become closer and are in the interval $65\% \approx 69\%$. As expected, larger *DbC*'s also contribute to attenuate the



(a) Distance, Random Waypoint

(b) Distance, Manhattan

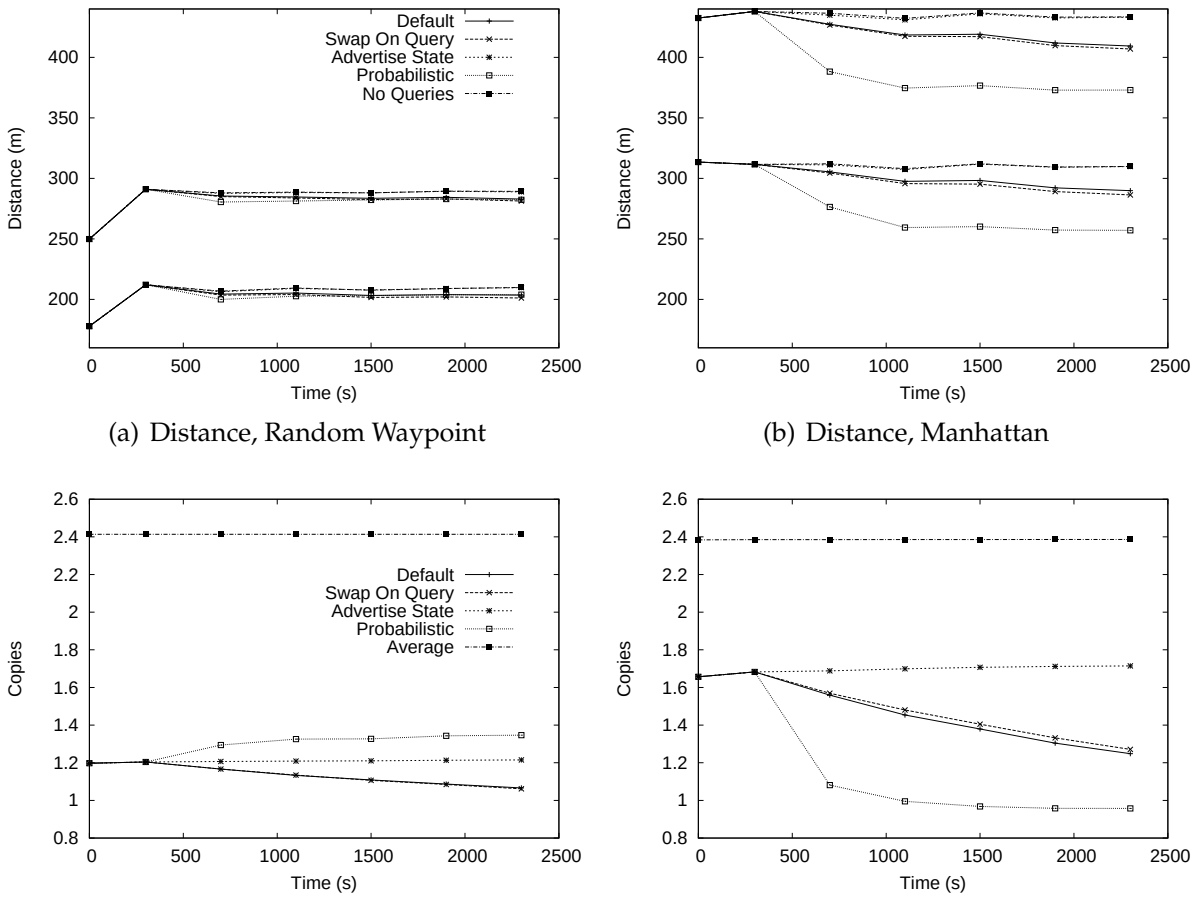
(c) Copies, Random Waypoint

(d) Copies, Manhattan

Figure 4.32: Mitigation tests with 250m transmission range, $2m/s$ average speed, 700 items and *DbC*=2

differences between the movement models and the ratios become around 65% and 70% respectively at times 0s and 300s.

A common trend that can be found in the Random Waypoint model tests is the stabilisation of the average distance, aligned with the control tests. This suggests that the shuffling algorithms are unable to keep the pace with the changes in the topology. These difficulties are more evident when we compare the graphics for the different speeds with the graphics presented for the same configuration in convergence tests. It can be observed that the tests for nodes moving at $5m/s$ are those presenting the lowest gains and that convergence tests always present the highest gains. However, we note that the requirement that nodes never stop is particularly demanding. As convergence



(a) Distance, Random Waypoint

(b) Distance, Manhattan

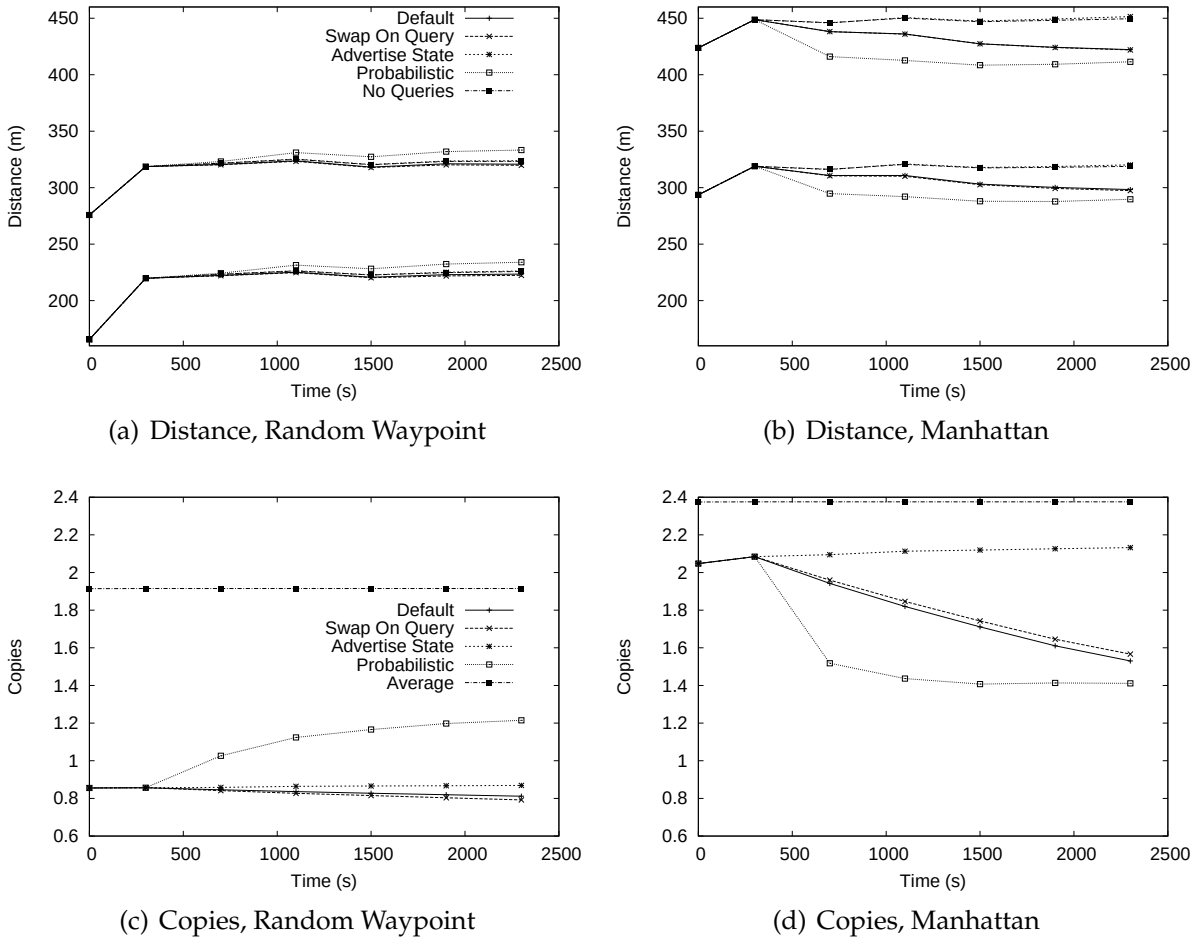
(c) Copies, Random Waypoint

(d) Copies, Manhattan

Figure 4.33: Mitigation tests with 200m transmission range, $2m/s$ average speed, 700 items and *DbC*=2

tests have shown, soon after the topology has stabilised, the algorithms are able to improve the average distance. In a real scenario, we expect the algorithms to present a behaviour that would be between the two extreme situations. We note also that the pace of queries in these tests is constant. If the number of queries was increased, the number of opportunities for the algorithms to improve the shuffling would also increase and we should expect better results.

The *Probabilistic* algorithm is the one whose results tend to differentiate more from the remaining. In some cases, it is even presenting results that are worst than those observed in the control tests, suggesting that the algorithm is making more harm than good to the distribution. This is consistent with the behaviour observed in the conver-



(a) Distance, Random Waypoint

(b) Distance, Manhattan

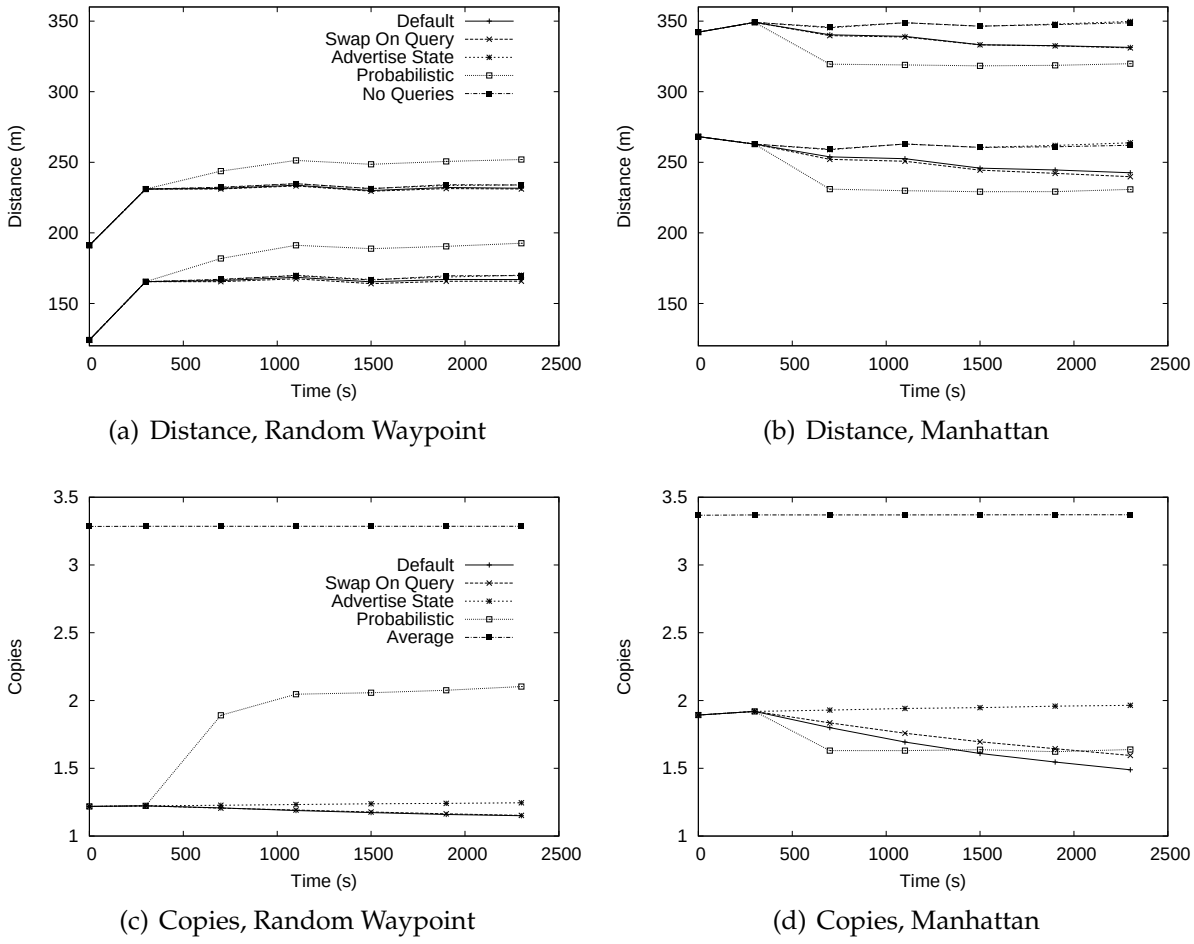
(c) Copies, Random Waypoint

(d) Copies, Manhattan

Figure 4.34: Mitigation tests with 200m transmission range, $5m/s$ average speed, 700 items and *DbC*=4

gence tests where it was noticed that the algorithm tends to stabilise at some average distance above the minimum, specially when the saturation point ratio is high. We note that like before, Figure 4.33 shows that in the lowest saturation point ratio we experimented, the *Probabilistic* algorithm is again providing the lowest average distance, although not so clearly distinguishable from the remaining algorithms.

The results for the Manhattan Grid movement model on the other hand contradict the previous analysis by showing that even when nodes move it is still possible to improve the average distance metric. The behaviour of each algorithm concerning the average distance is comparable with the graphics presented for convergence tests. We emphasise two aspects that are likely to justify these differences: i) the substan-



(a) Distance, Random Waypoint

(b) Distance, Manhattan

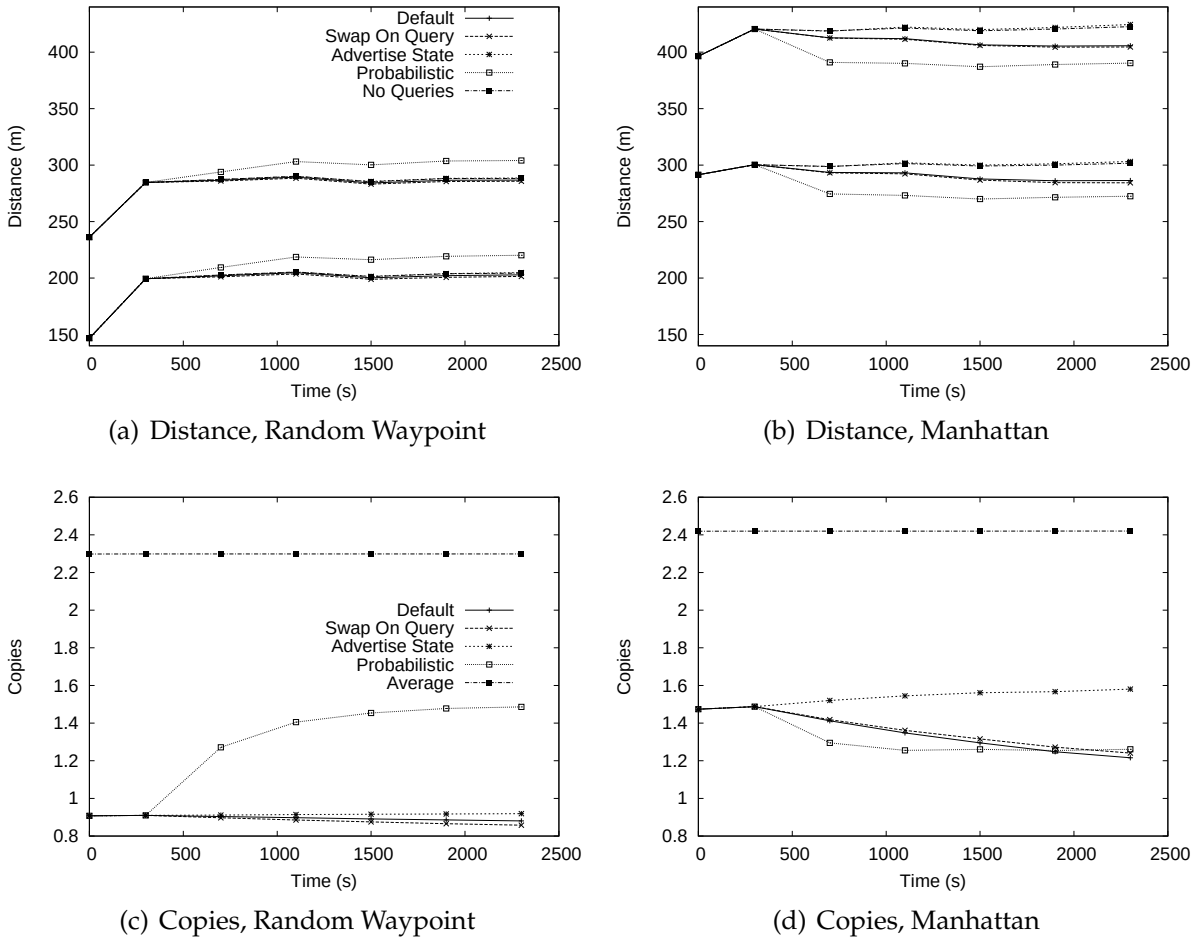
(c) Copies, Random Waypoint

(d) Copies, Manhattan

Figure 4.35: Mitigation tests with 200m transmission range, $5m/s$ average speed, 400 items and *DbC*=2

tially smaller number of neighbours and corresponding saturation point ratio; and ii) the highest average distance at the end of the dissemination phase, even with saturation point ratios above one (see for example, the results for this movement model in Figures 4.30 and 4.34 which have a saturation point ratio of 1.5).

In these tests, the *Probabilistic* algorithm clearly outperforms the remaining. The algorithm rapidly makes the average distance converge to a value that is, in general, below the achieved with the initial dissemination. It should also be noted that, contradicting the rule observed in the majority of the convergence tests, the standard deviation of the number of copies also tends to decrease. This confirms that none of the algorithms applies a particular bias to the number of copies of some item that could,



(a) Distance, Random Waypoint

(b) Distance, Manhattan

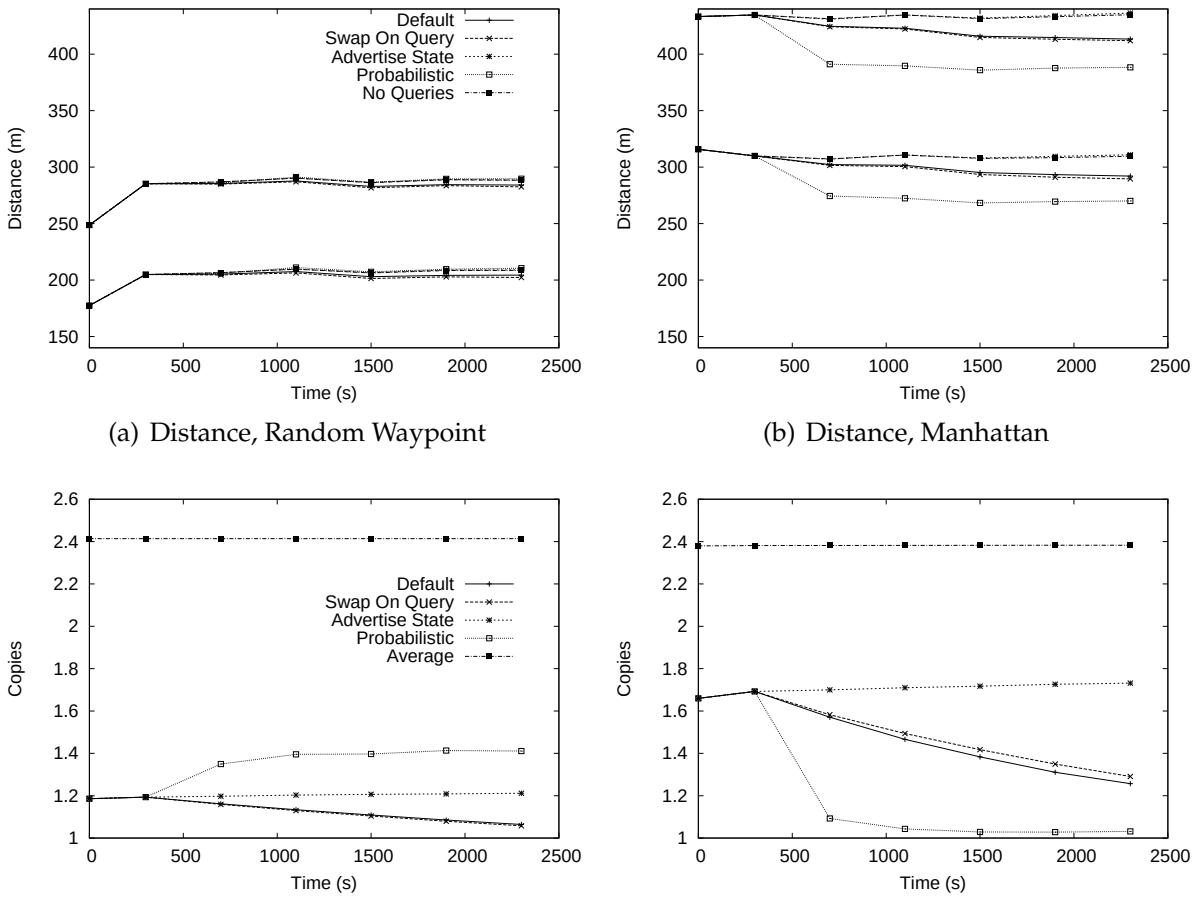
(c) Copies, Random Waypoint

(d) Copies, Manhattan

Figure 4.36: Mitigation tests with 250m transmission range, $5m/s$ average speed, 700 items and *DbC*=2

in the long term, affect the distribution. In fact, the standard deviation adapts to the networking environment and can either increase or decrease to improve the average distance.

The *Advertise State* algorithm is the one that exhibits the worst performance. In the Manhattan Grid tests it is in general coincident with the control tests. We attribute this behaviour to two particular features of the algorithm: $i)$ its strong commitment on the preservation of the number of replicas and $ii)$ an incorrect application of the *replace* flag that results from node movement. Recall from Section 4.5.3 that this algorithm only advertises items stored with the *replace* flag set to false, meaning that no other replica is known to be in the neighbourhood. However, in the presence of node



(a) Distance, Random Waypoint

(b) Distance, Manhattan

(c) Copies, Random Waypoint

(d) Copies, Manhattan

Figure 4.37: Mitigation tests with 200m transmission range, $5m/s$ average speed, 700 items and *DbC*=2

movement, nodes establish temporarily contact with others which also store replicas of the same items. Due to the lack of refreshment of the flag, after some time, nodes will cease to advertise their items, rendering the algorithm useless. The definition of an adequate refreshment criteria for the flag will always depend of specific conditions of the environment where the algorithm is going to be applied, removing generality to the algorithm.

These results confirm the trend that emerged from the analysis of the convergence tests. Shuffling algorithms can contribute to improve the distribution performed by the dissemination algorithm, specially in the cases where the saturation point ratio is very low. However, we note that in the general case, PADIS gives a valuable contribution to

the overall performance of the framework. In particular, it provides an estimate of the number of copies that are required and places them in adequate locations. Shuffling algorithms help by refining the dissemination. However, they depend of the queries and may suffer performance problems in some movement patterns.

4.8 Summary

This chapter presented PADIS, an algorithm for disseminating information in ad-hoc networks. The algorithm is fully distributed as it does not assume the presence of only a few data sources in the ad-hoc network; each node, instead, is expected to advertise its own data items. The main goal of the algorithm is to ensure an even geographical distribution of the disseminated data items, so that requests for a given data item are satisfied by some nodes close to the source of the query. Data items are disseminated with a counter to provide a minimal distance between the copies. The use of Pampa reduces the number of messages required for propagation and increases the geographical distance between the hops.

The chapter also presented a complement to PADIS, that consists on different algorithms for leveraging item distribution in the presence of node movement. The four algorithms described use different techniques and put different efforts in the preservation of the number of replicas. The algorithms require a limited amount of power, given that they either use point-to-point messages or piggyback their information in messages used for data retrieval.

Simulation results show that PADIS achieves a fair dissemination of items throughout the network although the results are slightly worst than what could be theoretically expected. The results are justified by the impossibility of providing an ideal networking environment.

Comparative simulations of the shuffling algorithms have confirmed their capability to improve data distribution, particularly in the less favourable conditions for

PADIS. The *Probabilistic* shuffling algorithm clearly stands, by lowering the average distance from each node to each data item faster than the remaining.

5

Application

It is easy to find in the literature different applications for data distribution algorithms in mobile ad hoc networks. Good examples are the support of cooperative work performed by geological teams during their field studies (Hara, 2001) or the distributed caching of web pages (Sailhan & Issarny, 2003). In general, applications can be arranged in two classes: those that provide innovative services that emerge from the possibilities offered by the mobility of the devices and those that aim to extend the different services typically provided by the Internet to the ad hoc networking environment. This chapter focus on the later.

Many of the services provided in the Internet are implemented using a client-server model requiring well known and reliable servers. In MANETs, a comparable model can only be achieved by replicating the service in multiple nodes. Therefore, a challenge that is posed in extending Internet applications to MANETs is to devise alternative implementations that are able to cope with the constraints of the networking environment. Ideally, the alternatives should be able to mask the differences of the underlying environment so that applications developed for the wired setting could run transparently in MANETs.

This chapter shows how the algorithms described in this thesis have been applied to support the Session Initiation Protocol (SIP) (Rosenberg *et al.*, 2002) in ad hoc networks. The SIP architecture is network-centric (Schulzrinne & Rosenberg, 2000), as SIP relies on network entities to assist end users in their operations. SIP deployment in server-less environments, such as MANETs, is thus not straightforward. For self-containment, the chapter begins by providing in Section 5.1 a short overview of SIP and

decentralised SIP (dSIP) (Leggio *et al.*, 2005), a solution for deploying SIP in MANETs.

dSIP was initially proposed in the context of 1 hop MANETs: those where all nodes are in the transmission range of each other. In addition, dSIP exhibited scalability problems. Section 5.2 describes SIPCache, a combination of PADIS with dSIP and with a data gathering module, developed to mitigate dSIP's previous limitations.

Section 5.3 evaluates the data gathering module of SIPCache. The chapter is concluded with a summary of the results in Section 5.4.

5.1 Overview of SIP

The Session Initiation Protocol (SIP) (Rosenberg *et al.*, 2002) is a signalling protocol designed by the IETF to initiate and manage multimedia communication sessions in the Internet. A signalling protocol negotiates all the parameters necessary for a communication session, e.g., its type (VoIP call, video call, chat) and characteristics (e.g, audio codecs) used. A most relevant characteristic of SIP is its flexibility: SIP can transparently signal any type of media session and is easily extendable to support new features.

A SIP network is composed of multiple domains, each independently administered by some organisation. Each domain provides a *Registrar Server* and a *Proxy Server*. In order to use SIP services, it is necessary to register a user's own contact information in a SIP domain. Usually, the information consists in the SIP address, that is, the SIP unique user name, also referred to as Address of Records (AOR), and the current IP address. An association between a user's AOR and his contact IP address is called a "binding" and is stored in the *Registrar Server* for the domain. Queries are addressed to *Proxy Servers*, who forwards them to the *Registrar Server* if the query is addressed to some AOR of the local domain. Queries for AORs of other domains are forwarded to the corresponding *Proxy Server*.

Authorised users communicate their bindings to the registrar server using a *SIP*

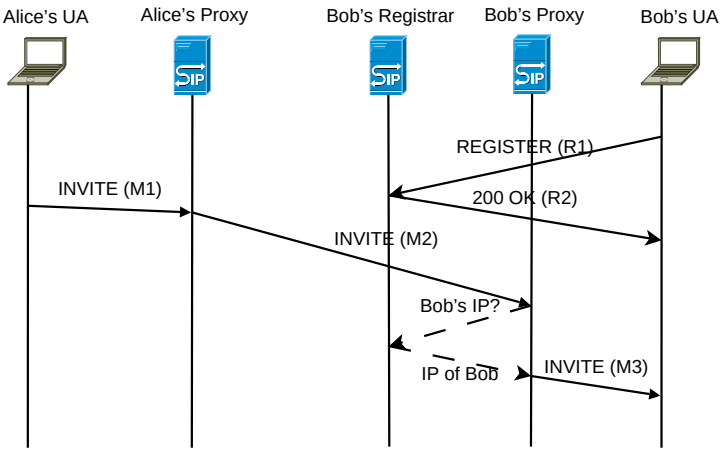


Figure 5.1: Simplified logical message flow in SIP

REGISTER message. Figure 5.1 shows that Bob has registered to his registrar server (message R1) and that the server has acknowledged the registration (message R2).

In general, when contacting a SIP user, a SIP client only knows the AOR of the person to contact. Similarly to what happens in e-mail systems, the contact IP address is typically unknown. In the example of Figure 5.1, user Alice, who wants to begin a SIP session with Bob, sends an INVITE message first to the *SIP Proxy Server* of her domain (message M1). The role of the proxy server is to locate, on behalf of the authorised users, the final recipient of a SIP request. Alice's proxy, thus, contacts the proxy serving Bob's domain (message M2).

Bob's proxy recognises that the INVITE request is addressed to a user belonging to its domain and exploits the location service, querying the domain registrar about the contact address of the target user, as indicated by the dashed line messages in Figure 5.1. The registrar returns to the proxy the contact information, and the message is forwarded to destination (message M3). The location service related messages are depicted using dashed lines to signal that the distinction between proxy and registrar is merely logical: it is not uncommon that operators co-locate their functionalities into the same host. The IETF has not defined a standard protocol for exploiting the location service.

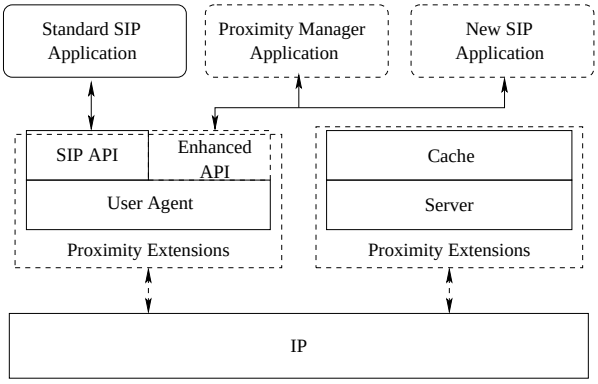


Figure 5.2: Software architecture for decentralised SIP

5.1.1 Decentralised SIP

Mobile Ad hoc Networks are by definition self-managed networks without the support of an underlying infrastructure composed for example of SIP proxies or registrar servers, fundamental to the correct behaviour of the SIP protocol. Decentralised SIP (dSIP) (Leggio *et al.*, 2005) is an architecture that overcomes this limitation, allowing the use of SIP also in server-less environments, such as MANETs. In addition, dSIP extends standard SIP features with proximity functionalities for discovering which users are present in the MANET.

The key idea of dSIP is to embed in each enabled device a basic subset of SIP proxy and registrar server functionalities, so that dSIP devices are self-capable to maintain and exploit the location service. Decentralised SIP is particularly suited for small MANETs, with few dozens of nodes at most.

The software architecture of dSIP is shown in Figure 5.2: the modules bordered with solid lines are standard SIP modules in a device. The dashed modules are the extensions defined by dSIP. In a standard SIP client, only the user agent (UA) side of the stack would be present. In dSIP, the server module is added, and the server standard capabilities are enhanced with the proximity functionalities. The interested reader is referred to (Leggio *et al.*, 2005) for a complete description of the role of each module.

dSIP does not require the modification of the existing software modules of SIP

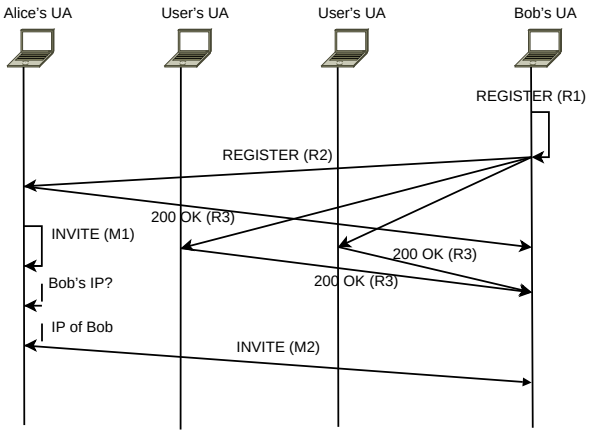


Figure 5.3: Simplified logical message flow in dSIP

clients. Extensions are enabled by adding new submodules. This choice allows interoperability of dSIP UAs with standard SIP clients: a standard SIP application can be deployed on top of dSIP equally well as an application that exploits dSIP proximity functionalities. A native SIP application is unaware of the presence of a modified SIP stack in the device, since it only utilises the standard SIP features. Moreover, a native application can be utilised in MANETs, since the underlying middleware is able to handle all the SIP messages sent by the application in the proper way.

The working principle of dSIP is that the user agent registers with the co-located registrar server, according to standard SIP procedures, by sending internally a REGISTER message. The server then registers the SIP user to the network spreading a SIP REGISTER message. The server modules in the network receive the REGISTER message, update their cache entry with the binding, and may reply to the registering node by sending a "200 OK" confirmation message. The registering node server module updates its cache with the bindings received from the other nodes. With this procedure, also referred to as general registration, the SIP location service is maintained in a distributed way among all the nodes. Figure 5.3 illustrates dSIP's handling of SIP REGISTER and INVITE messages.

A native SIP application sends a REGISTER message to its predefined registrar server. The dSIP modules transparently intercepts the message and routes it to the local server, ensuring interoperability with standard SIP applications.

Inviting a peer to a SIP session is similar: the INVITE message is routed to the co-located server, which checks in its cache for a binding for the queried user. The INVITE message is forwarded to the correct address if a match is found. dSIP proximity functionalities extend SIP's standard services by supporting queries for the list of users in the network. This allows the user to begin sessions also with previously unknown users.

5.2 SIPCache

dSIP works well in small networks. The design of dSIP assumes that all the server module instances know the bindings of all users, what is not scalable. Moreover, dSIP makes no assumption about the mechanism used for the spreading of the bindings.

dSIP was combined with PADIS to address these issues. The key idea of the resulting framework, named SIPCache, is to perform a partial replication of the AORs. The architecture of SIPCache is depicted in Figure 5.4. SIPCache benefits of the geographical distribution of the replicas provided by PADIS to keep a copy of the AORs in the proximity of any node. PADIS uses the original dSIP's cache as its storage space. To fully encompass the original dSIP facilities, SIPCache includes a data gathering module supporting the proximity functionalities.

5.2.1 Dissemination and Retrieval Using PADIS

In SIPCache, the dissemination and retrieval algorithms described in Chapter 4 are used to perform respectively the dissemination and query of bindings. The integration of PADIS with dSIP required some minimal changes. For example, data items were complemented with an expire field and a version number to allow nodes to purge old records and keep track of the most up-to-date binding.

Figure 5.5 shows the logical flow of the dissemination in SIPCache. In comparison with dSIP, it can be observed that the message flow reflects the PADIS data distribution

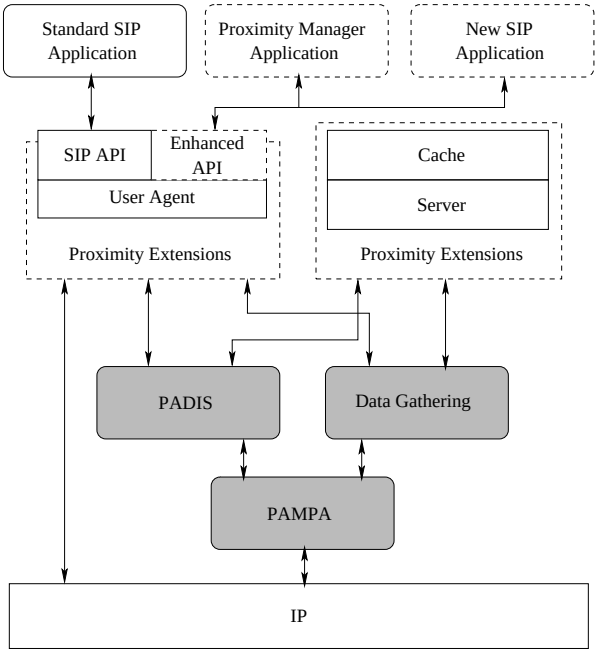


Figure 5.4: Software architecture for SIPCache

model. Registrations are conceptually delivered to a subset of the participants, with the remaining contributing only for the message dissemination. Given that bindings are stored on a subset of the participants, queries must be broadcast. However, the fine geographical distribution capabilities of PADIS permit that, in the majority of the cases, a copy of the binding is found in a limited number of hops from any node.

Crucial for the integration of PADIS in SIPCache are functions mapping SIP messages on the corresponding fields of PADIS. In particular, these functions are able to map AORs, locations and user interests on the key and value fields of the data items of PADIS. Figure 5.6 shows how the dissemination functions of PADIS and dSIP have been integrated. In constrast with dSIP, SIPCache does not use acknowledgement messages. These are used in dSIP to synchronise the cache of the incoming users what could affect the scalability of the algorithm in large scale networks, due to the large number of acknowledgements that would be received. In SIPCache, nodes are not expected to store every AOR and therefore, acknowledgements are not required. If the probabilistic shuffling policy presented in Chapter 4 is used, the data items piggybacked on the messages for data shuffling can also be used to populate the empty

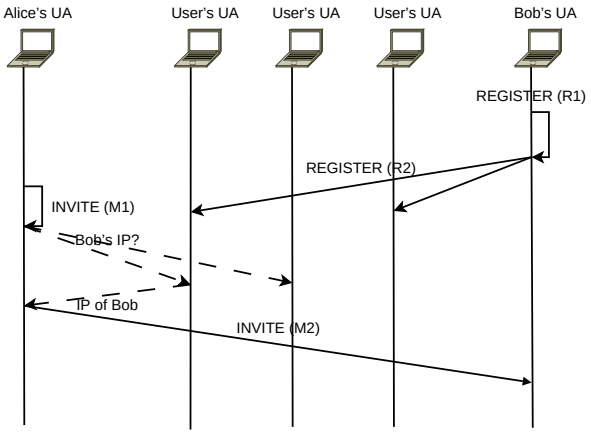


Figure 5.5: Simplified logical message flow in SIPCache

```
1: procedure BINDING(SIPMessage)
2:     binding ← EXTRACTBINDING(SIPMessage)
3:     PADIS.REGISTER(binding)
4: end procedure
```

Figure 5.6: Binding dissemination in SIPCache

storage spaces of the new nodes.

Depending of the type of query performed, SIPCache may use either the data retrieval algorithm described in Chapter 4 if the query is aimed to a specific AOR (targeted queries) or the data gathering module, described below, if the goal is to retrieve a set of AORs matching some criteria. The pseudo-code for query handling is outlined in Figure 5.7. The figure uses an abstract monitor for representing the suspension of the thread of execution until a reply is found. In the case of targeted queries, the key is passed to PADIS's query method who will search the local cache and trigger one run of the query algorithm if the reply is not locally available.

5.2.2 Data Gathering Module

One of the facilities provided by dSIP was to allow an user to retrieve a list of the participants whose AOR satisfied some criteria. For example, users may find interesting to learn about all available printers or to find other users in the ad hoc network matching a specified profile. A query for all the people interested, e.g., in sports, can

```
1: reply←” “
2: monitor

3: function QUERY(SIPMessage)
4:     if GETQRYTYPE(SIPMessage)=TGTQRY then
5:         AOR ← EXTRACTAOR(SIPMessage)
6:         PADIS.QUERY(AOR)
7:     else
8:         cond ← EXTRACTCOND(SIPMessage)
9:         DG.GATHER(cond)
10:    end if
11:    MONITOR.WAIT
12:    return reply
13: end function
```

Figure 5.7: Query handling in SIPCache

be issued, and the application should return to the querying node a list of users and the necessary contact information. In practice, instead of a key, queries will be performed specifying a condition that must be satisfied by the key of the data item. Conditional queries operations are referred to as “data gathering” operations.

Given that in dSIP each node kept a full list of the AORs advertised by other nodes, data gathering was an operation local to each node. SIPCache performs only partial replication of the AORs. Therefore, to perform a data gathering operation, a node can not rely exclusively in the content of its storage space. Instead, it must also query some other nodes to learn about the AORs it does not store locally.

It should be noted that when data items are distributed by several nodes, the data gathering problem has some similarities with the problem of data aggregation, extensively investigated in the scope of sensor networks (see for example (He *et al.*, 2004; Madden *et al.*, 2002; Przydatek *et al.*, 2003; Sharaf *et al.*, 2004; Shrivastava *et al.*, 2004)). However, data aggregation algorithms typically assumes a different data distribution model and could not fully exploit the fine geographical distribution capabilities of PADIS.

Our data gathering module is well suited to the cases where replicas of the data have been geographically dispersed over the network. The goal is to reduce the cost

of each operation by limiting the search to the nodes located a few hops away from the source, benefiting of the Pampa message dissemination capabilities. Data gathering operations run in two phases: query dissemination and data delivery. In the first phase, queries are disseminated within a bounded number of hops using Pampa. In the data delivery phase, each node replies to the source of the first transmission of the query it has received. Nodes that receive multiple replies are responsible for removing redundancy of the data items they collect and to forward the aggregate to the source of the query. A detailed description of the module is presented below.

5.2.2.1 Detailed Description

In a process similar to the one used by PADIS, each data gathering operation partitions the nodes in the network in clusters. The partitioning is triggered by the broadcast of a data GATHERING message using Pampa. Nodes selected for retransmitting the message become cluster heads. A node becomes a member of at most two clusters: of the one it leads (if any) and of the cluster headed by the node that transmitted the first copy of the message it has received.

The main characteristic of our clustering approach is that it is stateless and message-driven, that is, clusters are formed based only on the propagation of received messages, as dictated by Pampa. There is no need to implement a dedicated membership protocol, or for the cluster heads to keep state information on the nodes belonging to their cluster. Each cluster head is responsible for preventing the delivery of redundant data items to the source node by locally aggregating replies from the remaining cluster members. An example of a cluster partitioning of a network is depicted in Figure 5.8.

The use of Pampa gives a significant contribution to the performance and adaptability of the data gathering module. As emphasised in Chapter 3, in the absence of abnormal propagation effects, Pampa selects for retransmission the nodes that are more distant from the previous source. Therefore, and using the same assumptions, the module selects for cluster heads the nodes that are more distant from the sources

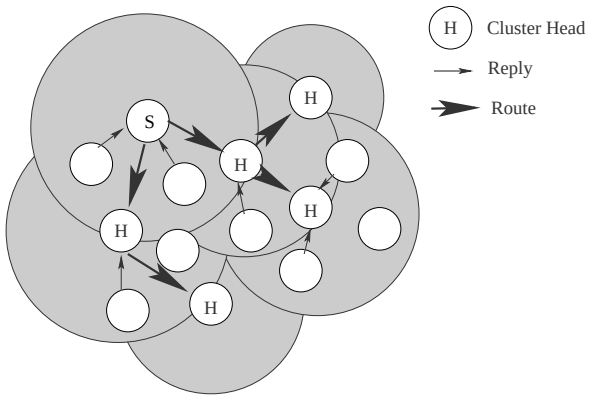


Figure 5.8: Propagation of gathering messages and replies

of the previous transmissions. The advantages of this approach are:

- a geographical extension of the coverage of each retransmission, allowing to reduce the diameter (in hops) of the query;
- an increase in the number of nodes on each cluster, allowing to collect more information on each cluster head. Because more nodes are reached, the probabilities of collecting all data items satisfying the condition also increases;
- a simple adaptation to different network conditions, as it was shown in the evaluation of Pampa.

As shown in Figure 5.9, a data gathering operation is triggered by the broadcast of a GATHERING message using Pampa and containing the condition to be satisfied by the data items. The message accumulates the path to be followed by the replies in a *routeStack* field.

Each data gathering operation is performed using a ring search. The radius of the ring is defined at the source of the query in a field commonly identified as Time-To-Live (TTL), decremented by each node that forwards the message.

To limit the number of redundant replies, the message is filled up with data items up to a maximum message size. These data items are selected from those that are known to satisfy the condition and have been received in a previous transmission of

```
1: procedure GATHER(cond)
2:     mid←CREATEMSGID
3:     knownSet_mid ←STORAGE.SEARCH(cond) ▷ Return items in local storage space
4:     for (key,val) ∈ knownSet_mid do
5:         DELIVER(key,val)
6:     end for
7:     routeStack← {}                                          ▷ Prepare message
8:     PUSH(routeStack,addr)
9:     ttl←GETGATHERINGTTL
10:    advSet←CREATEADVSET(cond,knownSet_mid,knownSet_mid)
11:    msg←(GATHER,cond,routeStack,advSet)
12:    PAMPA.RELAY(mid,msg,ttl)
13:    recvdMsgs← recvdMsgs∪{mid}
14: end procedure
```

Figure 5.9: Bootstrap of the data gathering operation

the data GATHERING message or are stored in the node performing the retransmission. These items are used to let the receivers learn about some of the items that are already known by the cluster head and therefore do not need to be sent.

We have defined three different policies for selecting these items, and compare them in the evaluation section. Figure 5.10 presents one implementation of function CREATEADVSET for each policy.

The *not filled* policy (Figure 5.10(a)) is the control policy that will allow us to assert the benefits of our approach. It disseminates the GATHERING messages without any data items. We expect operations using this policy to present the highest number of reply messages although the lowest size of the GATHERING message.

Of the *Ordered* and *Random* policies, the later is the one expected to occupy the available space in a smaller number of hops. In the *Ordered* policy (Figure 5.10(b)), each node forwarding the GATHERING message pushes the data items available in its storage space and that satisfy the query until the message becomes filled. The data items remain unchanged if a node retransmits a message already filled. Although simpler, an expected side effect of this policy is the redundant advertisement of the same data items by multiple nodes what may reduce its efficiency. Nodes do not reply with items present in the message.

```
1: procedure CREATEADVSET(cond,firstSet,recvdSet)
2:     return {}
3: end procedure
```

(a) Not filled

```
1: procedure CREATEADVSET(cond,firstSet,recvdSet)
2:     set ← firstSet
3:     localSet ← STORAGE.SEARCH(cond)
4:     while ¬FULL(set) do
5:         set ← set ⋃ {x : x ∈ localSet ∧ x ∉ set}
6:     end while
7:     return set
8: end procedure
```

(b) Ordered

```
1: procedure CREATEADVSET(cond,firstSet,recvdSet)
2:     set ← {}
3:     localSet ← STORAGE.SEARCH(cond)
4:     bigSet ← firstSet ⋃ recvdSet ⋃ localSet
5:     while ¬FULL(set) do
6:         set ← set ⋃ {x : x ∈ bigSet ∧ x ∉ set}
7:     end while
8:     return set
9: end procedure
```

(c) Random

Figure 5.10: Policies for filling data gathering messages with known data items

In the *Random* policy depicted in Figure 5.10(c), nodes snoop the network and collect in a *recvdSet* set all the data items advertised in every retransmission they listen of the GATHERING message. The items advertised are randomly selected from this set and from those locally known by the node. The reply will not include any item that is present in the *recvdSet* set. We expect this to be the most efficient redundancy removal mechanism as it prevents the retransmission of data items known by several cluster heads.

It should be noted that the selection of a policy does not imply a change in the number of retransmissions of the GATHERING message, although it should affect the number of bytes transmitted. The gains are expected to occur in the reduction of the

number of reply messages. We notice that the transmission of a frame implies both a variable energy cost that depends of the number of bytes transmitted but also a fixed cost (Feeney & Nilsson, 2001). In the fixed cost, one must account for example with the collision avoidance protocol (which requires 3 link layer frames in IEEE802.11) as well as the fixed sized headers of the lower level protocols. Therefore, policies that require the transmission of more bytes may still provide non negligible energy savings resulting from the transmission of a smaller number of messages.

Figure 5.11 shows the handling of a GATHERING message by the nodes. The propagation strictly follows Pampa, which decides both the hold period and to forward or not the message. The message will be replied if at the end of the hold period, Pampa decides not to retransmit. A special case occurs when the dissemination phase is concluded (signalled by the *TTL* field), in which the reply is immediately forwarded to the cluster head.

If a node decides to retransmit, it decrements the TTL field of the message, updates the data items to be forwarded according to the predefined policy and push its address in the route stack, electing himself as cluster head. After retransmitting the message, the node enters a second hold period defined by multiplying the value of the *TTL* field by a constant. The value should be enough to permit the dissemination phase to be concluded so that all replies can be successively collected by the cluster heads and delivered to it before his timer expires.

Independently of being or not a cluster head, all nodes send a reply message only when it is not empty. Figure 5.12 shows the pseudo-code of the reply phase of the data gathering operation.

If a node receives a reply then it is either a cluster head or the source of the gathering operation. Cluster heads collect all data items received in replies. The aggregate of the replied data items, without duplicates, is forwarded to the previous cluster head.

A message may be delivered to a cluster head after its timer has expired. Nodes do not keep any state about past operations. Therefore, when a node receives a reply

```
15: upon event PAMPA.RECEIVED(mid,(GATHER,cond,routeStack,advSet),ttl,holdTime)
    do
16:     if mid ∉ recvdMsgs then
17:         recvdMsgs← recvdMsgs ∪{mid}
18:         knownSet_mid ← advSet
19:         replySet_mid ← {}
20:         ttl_mid ← ttl − 1
21:         if ttl_mid > 0 then
22:             TIMER.SETALARM(holdTime,(FWD,mid))
23:         else
24:             PAMPA.DROP(mid)
25:             SENDREPLY(mid)
26:         end if
27:     else
28:         knownSet_mid ← knownSet_mid ∪ advSet
29:     end if
30: end upon

31: upon event TIMER.ALARM((FWD,mid)) do
32:     if PAMPA.DECIDE(mid) then
33:         FWDDG(mid)
34:         TIMER.SETALARM(kTime × ttl_mid,(RPLY,mid))
35:     else
36:         PAMPA.DROP(mid)
37:         SENDREPLY(mid)
38:     end if
39: end upon
```

Figure 5.11: Data Gathering message dissemination

for an unknown data gathering operation, it uses the route stack field to immediately forward the message. Note that since the node does not keep any state, it can not remove redundant replies from the message. Redundancy could possibly be removed at the next cluster head (if any) if it is delivered before its timer expires. In the worst case, it is delivered to the source of the query, who is the ultimate responsible for purging redundant replies before delivering the data to the application.

```
40: upon event PAMPA.RECEIVED(mid,(RPLY,replySet,routeStack),ttl,holdTime) do
41:     if routeStack={} then                          ▷ The node is the destination
42:         for (key,val) ∈ replySet ∧ (key,val) ∉ knownSet_mid do
43:             DELIVER((key,val))
44:         end for
45:         knownSet_mid ← knownSet_mid ⋃ replySet
46:     else                                                    ▷ Intermediate hop
47:         if TIMER.PENDING(RPLY,mid) then
48:             replySet_mid ← replySet_mid ⋃ replySet
49:         else
50:             nextAddr←POP(routeStack)
51:             ttl←ttl-1
52:             msg←(RPLY,replySet,routeStack)
53:             PAMPA.SEND(nextAddr,mid,msg,ttl)
54:         end if
55:     end if
56: end upon

57: upon event TIMER.ALARM((RPLY,mid)) do
58:     SENDREPLY(mid)
59: end upon
```

Figure 5.12: Reply handling

5.3 Evaluation

To evaluate the performance of the data gathering module, we have integrated it with the prototype of PADIS, implemented in the *ns-2* network simulator v 2.28. Experiments were developed to confirm that, given a fine initial distribution of the data, the module is able to retrieve a significant proportion of the data items requested using a small number of messages. In addition, we wanted to compare the three policies for filling GATHERING messages with known items to confirm if our intuition concerning the performance of each was correct.

5.3.1 Test-bed Description

The simulated environment is similar to the one used in the preceding chapters. Experiments used a simulated network composed of 100 nodes uniformly disposed

```
60: procedure SENDREPLY(mid)
61:     reply←STORAGE.SEARCH(cond_mid)
62:     replySet_mid ← replySet_mid ∪ {x : x ∈ reply ∧ x ∉ knownSet_mid}
63:     if replySet_mid ≠ {} then
64:         mid←CREATEMSGID
65:         nextAddr ← POP(routeStack_mid)
66:         msg ← (RPLY,replySet_mid,routeStack)
67:         PAMPA.SEND(nextAddr,mid,msg,netDiameter)
68:     end if
69: end procedure

70: procedure FWDDG(mid)
71:     PUSH(routeStack,addr)
72:     advSet ← CREATEADVSET(cond_mid, advSet_mid , knownSet_mid)
73:     msg ← (GATHER,cond_mid, routeStack, advSet)
74:     PAMPA.RELAY(mid,msg,ttl_mid)
75: end procedure
```

Figure 5.13: Auxiliary functions

over a region with $1500m \times 500m$. 100 different random deployments of the nodes were defined. Nodes do not move during the simulations. The simulated network is an IEEE 802.11 at 2Mb/s. Network interfaces have a range of 250m using the Free Space propagation model.

Runs are executed for 2400s of simulated time. Evaluation of data gathering is preceded by the dissemination of the data items, that takes place between 1s and 1600s. The tests experimented different loads of the network. The number of items was varied between 100 and 800 at intervals of 100. All nodes perform an equal number of advertisements. Nodes make available storage space for accepting at most 10 items, excluding the items they advertise. Each data item has a size of 300 bytes (50 for the key and 250 for the value). The maximum message size was set to 1300 bytes. Due to the size estimated for the condition and headers, a GATHERING message can contain up to 3 data items and a reply up to 4 data items.

Each run consists of 400 random data gathering operations starting after the distribution of the data items and uniformly distributed at a pace of approximately one operation every 2s. In the following, each data point averages 100 runs in similar

conditions but with a different combination of node deployment, moment of data advertising, sources of the gathering operations and items queried. The efficiency of the data gathering operation is evaluated using two metrics.

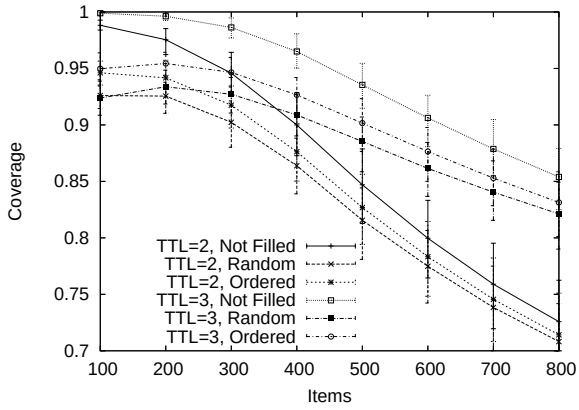
Coverage Gives the proportion of data items satisfying each gathering operation that are effectively delivered to the querying nodes. Only data items present in the node's storage space or delivered to it in the first 10 seconds after the query has been issued are accounted. In the tests presented, each operation tries to collect 10 data items.

Traffic A second metric accounts with the number of gathering query and reply packets and bytes forwarded by the nodes. It should be noted that each forwarding of a query or reply is accounted individually. Bytes are counted at the link layer level, thus including the IEEE 802.11 MAC header.

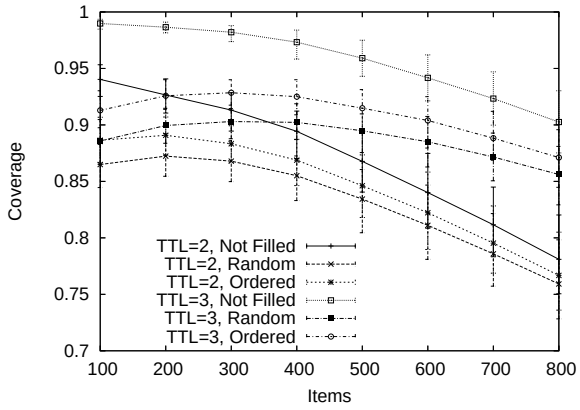
PADIS was configured with Distance Between Copies (DbC) values between 2 and 4. Results presented in Chapter 4 suggest that if data was adequately distributed and the saturation point ratio is above 1, in the majority of the cases data items should be found in one of any node's neighbours at most $\left\lceil \frac{\text{DbC}+1}{2} \right\rceil$ hops away. This assumption was used to experiment the data gathering module. For each value of DbC, two initial values were tested for the TTL field of the GATHERING message: $\left\lceil \frac{\text{DbC}+1}{2} \right\rceil$ and $\left\lceil \frac{\text{DbC}+1}{2} \right\rceil + 1$. Therefore, for DbC values of 2, 3 and 4, the initial values of the TTL field have been set respectively as 2 and 3, 2 and 3, 3 and 4.

5.3.2 Coverage

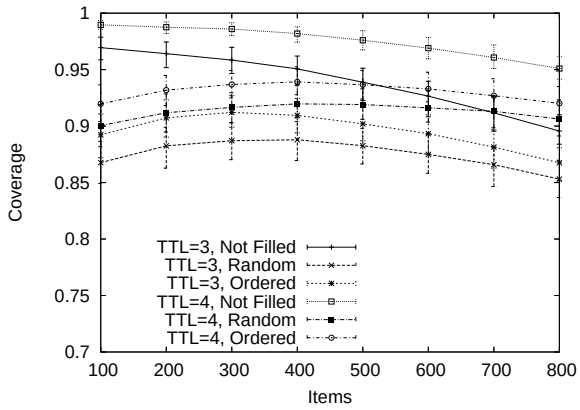
The proportion of items retrieved on each gathering operation is depicted in Figure 5.14. All the plots exhibit a common pattern. The best performing combination is the one that extends the search one hop away of the expected distance for the predefined value of DbC. Also, for each DbC and TTL, the "Not Filled" is the policy that gives an higher coverage, followed by the "Ordered" policy. The differences between



(a) DbC=2



(b) DbC=3



(c) DbC=4

Figure 5.14: Coverage of the data gathering module

policies are more significant for a lower number of items advertised but never exceed 10%, corresponding to one queried item.

The plots confirm the efficiency of PADIS. Within a bounded number of hops, not far from the expected average distance, nodes are able to find a large majority of the items that have been advertised.

Chapter 4 referred that the number of nodes that would suffer from an ill distribution of the items in its neighbourhood and therefore would be likely to fail the expected maximum distance should increase with the DbC. This explains the lowest performance when DbC is 3 or 4 (in comparison with DbC=2) for a small numbers of advertised items. Not surprisingly, the coverage decays more significantly for DbC=2. This is due to the approximation to the saturation point, which occurs approximately

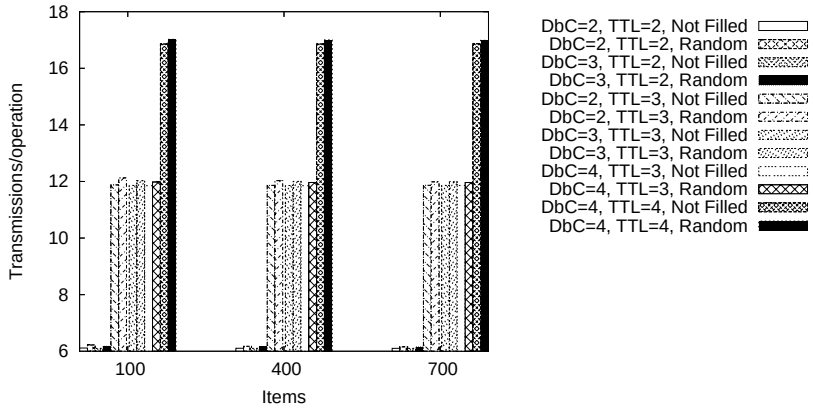


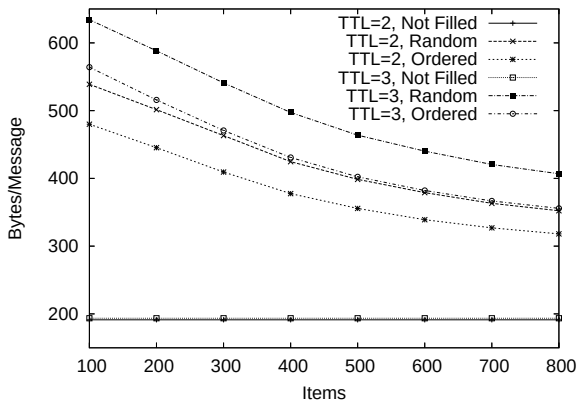
Figure 5.15: Retransmissions of the gathering message

at 650 items. Even in conditions beyond those that would permit to PADIS to maintain its properties, above 70% of the items are found within 2 hops and more than 80% have one replica located at most 3 hops away. For the remaining values of DbC tested, results do not decay so rapidly. This is justified by the lowest saturation point ratio attained for these values of DbC. Notice that specially for DbC=4, there is an almost constant performance, showing that the storage capacity of the nodes is far from being exhausted. Because less copies of the items are stored, there is also less redundancy and therefore the distribution of the items is improved.

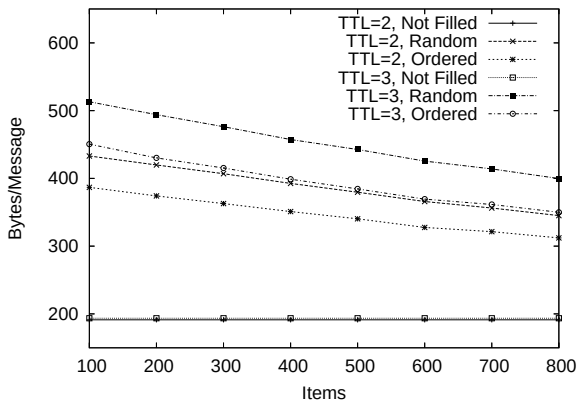
5.3.3 Traffic

Figure 5.15 compares the number of retransmissions of each GATHERING message for some of the scenarios presented above. The figure confirms that the number of retransmissions is dictated by the TTL and is independent of the DbC or the policy. A close observation of the figure shows that different policies exhibit minor differences on the ratio, which we attribute to negligible effects of the different message sizes on Pampa. Depending of the value of the TTL, the dissemination of a GATHERING message required on average between 6 and 17 transmissions.

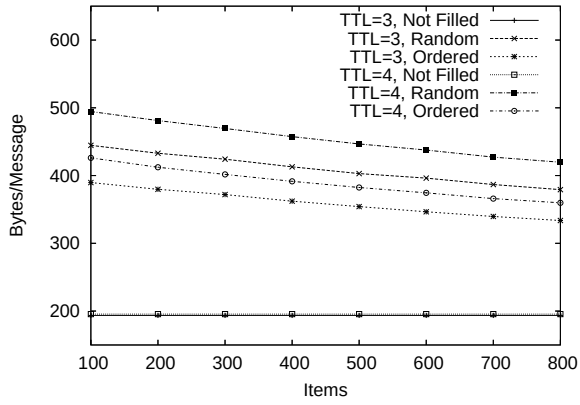
While the number of messages is kept constant, Figure 5.16 confirms that their sizes are affected by the policy used. Recall that in the "Not Filled" policy, the GATH-



(a) DbC=2



(b) DbC=3



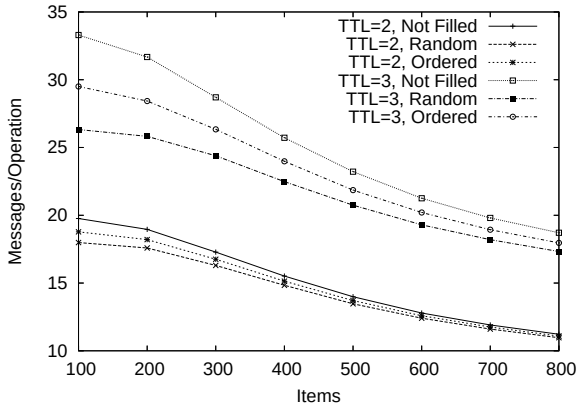
(c) DbC=4

Figure 5.16: Average data gathering message size

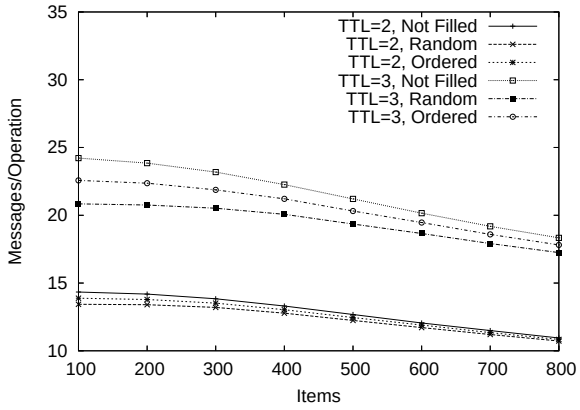
ERING message only contains the query. Therefore, as expected, the "Not Filled" policy presents a constant message size that is independent of the DbC or number of items advertised.

The "Ordered" and "Random" policies are both affected by the number of items. As the number of items increases, the probability of the nodes forwarding the message to store one of the items satisfying the query decreases. Because in the "Random" policy nodes can also append to the GATHERING message items listened on different retransmissions, there is an higher probability of having items to include in the message.

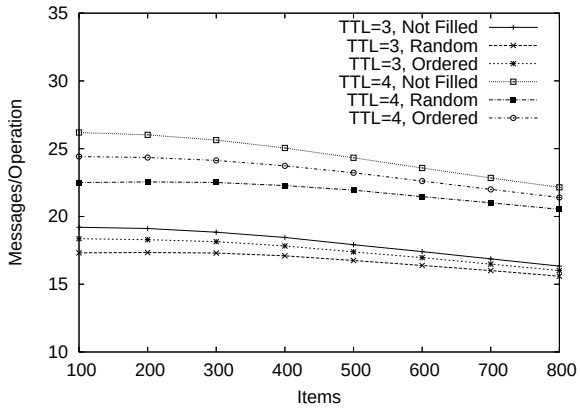
The number of reply messages also varies significantly, as depicted in Figure 5.17. In the "Not Filled" policy, nodes are not aware of the items that are already known



(a) DbC=2



(b) DbC=3



(c) DbC=4

Figure 5.17: Average number of reply messages

by previous cluster heads. Therefore, all nodes storing at least one item satisfying the query will send a reply to the cluster head. The difference is more visible in situations where the redundancy is higher, what occurs for low values of DbC and items.

It should be noted that since the number of GATHERING messages is independent of the policy, these plots represent the effective gains in the number of messages of each policy. Table 5.1 presents for 100 and 800 items the proportion of the replies transmitted by the "Not Filled" policy for those used by the "Random" policy. The table shows that ratios vary significantly, providing more substantial gains when the redundancy is higher, i.e., when there are less items advertised and when the TTL of the message is higher.

We emphasise that these results vary according to the tested scenario. The size

DbC	TTL	100		800	
		Reply Msgs.	Total Bytes	Reply Msgs.	Total Bytes
2	2	91.02%	106.21%	97.74%	112.40%
	3	79.07%	98.83%	92.61%	113.16%
3	2	93.71%	109.43%	97.91%	112.37%
	3	86.06%	107.43%	94.07%	114.48%
4	3	90.17%	112.31%	95.41%	116.10%
	4	85.92%	111.54%	92.68%	117.21%

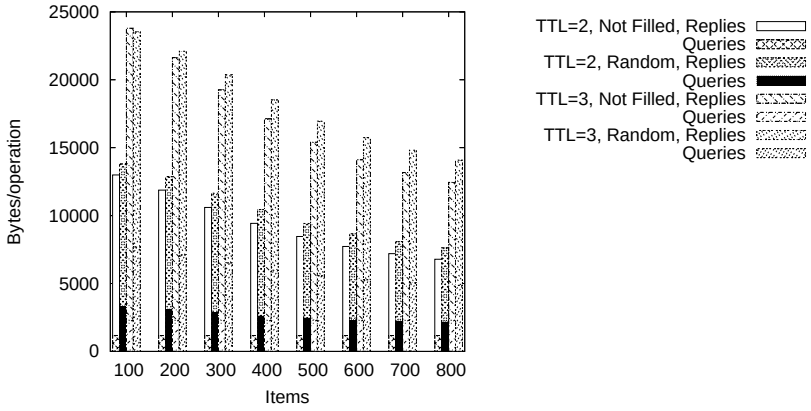
Table 5.1: Proportion of the reply messages and total bytes per operation transmitted by the "Not Filled" policy sent by the "Random" policy

of the data items, for example, is one factor affecting the performance of the different policies as it influences the number of data items that can be carried on each GATHERING message. An increasing number of items on the GATHERING message should provide additional gains in the number of messages because it would let nodes learn about more items that do not need to be replied. On the contrary, an increased number of items satisfying the condition should reduce the relevance of the "Random" policy because the probability of some node to listen to a GATHERING message carrying the item(s) it can provide will be smaller.

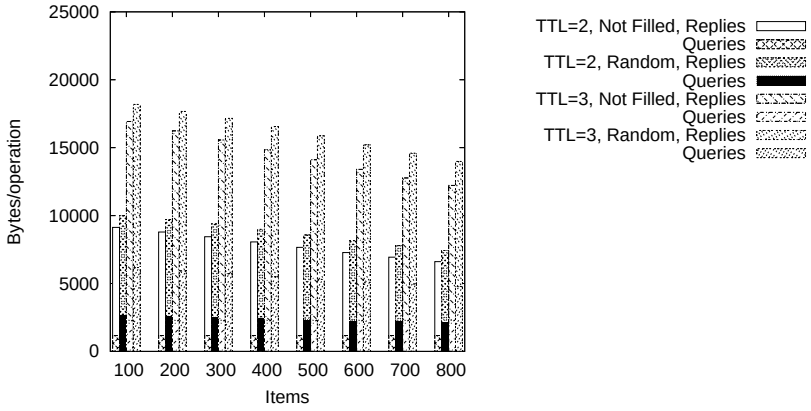
Figure 5.18 compares the average number of bytes transmitted per operation for "Not Filled" and "Random Policies". The figure clearly shows that the "Not Filled" policy consumes less bytes per operation. It also shows that there is a different distribution of the weight of each message type on the contribution for the total number of bytes transmitted.

As it was shown before, the "Random" policy uses more bytes on GATHERING messages. The difference is attenuated by the amount of bytes transmitted in *reply* messages, which is significantly higher in the "Not Filled" policy due to its inability to prevent redundant retransmissions. Table 5.1 supports these conclusions by presenting side-by-side the ratio of reply messages and total bytes per operation between the "Not Filled" and the "Random" policies.

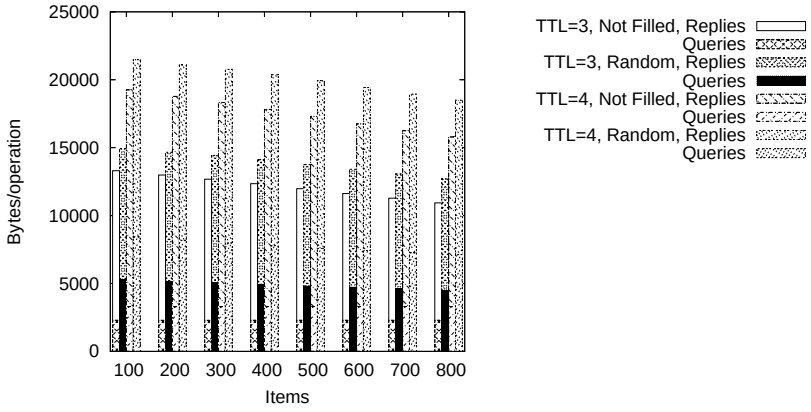
The table shows that in the cases where there is more redundancy, which are those where less items are advertised, the "Random" policy can save up to 20% of the reply



(a) DbC=2



(b) DbC=3



(c) DbC=4

Figure 5.18: Average bytes per operation

messages. However, the total number of bytes transmitted per data gathering operation increases up to 17%. As discussed before, the "Random" policy can still present some gains which can not be accounted in the simulation: a smaller number of messages decreases contention and reduces control messages, for example those required by the IEEE 802.11 MAC protocol for point-to-point message passing.

An undesirable side effect of the reduction of the number of reply messages is the increasing relevance of message losses to the coverage. Because there are less redundant replies, each reply is more likely to contain the unique copy of a data item satisfying the query that will be delivered to the source. The lowest coverage exhibited by the "Ordered" and "Random" policies (presented in Figure 5.14) is attributed to message losses. To support this conclusion, we notice that the difference (which in rare cases is of approximately 10%, or 1 item per gathering operation) is more significant when the number of advertised items is smaller, meaning that there is more redundancy. As redundancy decreases, the coverage results tend to approximate, showing that the "Not Filled" policy tends to become equally vulnerable to message losses. The $ns-2$ simulator was configured to present a reliable media, without message loss due to interference. Message drops continue to occur, for example due to an excessive number of collisions or the arrival of a second packet to the ARP temporary buffer, addressed to the same node while the ARP module is still trying to retrieve the MAC address of the node.

5.4 Summary

This chapter presented an application of the data distribution algorithms named SIPCache. SIPCache is a distributed location service for the SIP protocol, aimed to multi-hop ad hoc networks. SIPCache emerges directly from the combination of our data distribution algorithms with dSIP, an extension to the Session Initiation Protocol (SIP) for MANETs developed in an independent project, outside the scope of this thesis.

SIPCache confirmed that the algorithms are relevant for realistic environments. In

particular, it was shown that the algorithms can contribute to the adaptation required by some popular services provided in the wired Internet which rely on centralised servers.

SIPCache uses a data gathering module to collect an unknown number of data items satisfying some constraint. The module relies on the geographical distribution of the replicas and therefore, limits the scope of the search to a few hops away from the source. Simulations have showed that the module requires a limited number of messages per operation and presented acceptable completeness even in adverse conditions. It uses Pampa to disseminate the query within a limited number of hops and to define a cluster structure. Cluster heads are responsible for collecting information on behalf of the querying node, purging the replies from multiple copies of the same item.

Evaluation compared different approaches aiming to further reduce the traffic by preventing nodes from sending replies without a valid contribution. It was shown that it is possible to reduce the number of replies at expenses of increasing message sizes. In addition, a tradeoff between most efficient approaches to remove redundancy and the completeness of the information received was identified.

6

Conclusions and Future Work

The absence of a supporting infrastructure makes application development for Mobile Ad hoc Networks (MANETs) particularly challenging. During application development, the programmer is faced with different problems that can not be solved using traditional approaches. For example, due to the high unreliability of mobile devices and wireless networks it is inadequate to centralise data for distributed applications on a single node. Decentralised solutions that replicate data by different nodes should be preferred.

Message passing as been identified as one of the most significant sources of power consumption in mobile devices. The multi-hop nature of MANETs, together with the limited computational resources and power available to the devices, suggest that replicas should be geographically distributed. Geographical distribution of the replicas reduces power consumption because data can be retrieved by any node in the network using a small number of messages.

The geographical distribution of the replicas in MANETs is not a new subject. However, in a survey to the related work, we showed that previous algorithms required additional information like the location of the nodes or an anticipated information about the popularity of the items.

This thesis showed that it is possible to perform geographical distribution based only on the Received Signal Strength Information (RSSI), which can be made available by the wireless network interface of the devices. The algorithm presented in the thesis, named PADIS, performs a plain geographical distribution in the sense that it does not

bias the distribution of any item. Therefore, it is adequate for scenarios where it is not possible to estimate the data items that will be more requested.

Devices participating in MANETs are expected to move. A fine geographical distribution may become inadequate as a result of the movement of the nodes. The thesis presented, evaluated and compared different algorithms capable of leveraging the geographical distribution of the items. These algorithms complement PADIS as all share the same set of requirements. To preserve the resources of the devices, shuffling is triggered by information collected during data retrieval operations. The algorithms experiment different approximations, with and without preservation of the number of replicas and dedicated messages.

PADIS rely on a novel broadcast algorithm for MANETs named Pampa and equally presented in the thesis. The goal of Pampa is to reduce the number of nodes required to retransmit a broadcast message. Pampa is fully distributed and does not require the use of dedicated control messages. Instead, it ranks nodes according to their distance to the previous source and selects for retransmission those that are more distant. The distance is estimated from the Received Signal Strength Indicator (RSSI). Evaluation results show that this innovative application of the RSSI makes Pampa self-adaptable to different node densities and improves the delivery ratio when compared with the related work.

Finally, the thesis presented an illustrative application of these results. PADIS and Pampa were successfully integrated in a middleware library aiming to decentralise the Session Initiation Protocol (SIP) architecture so that it can be applied to ad hoc networks. The resulting framework, named SIPCache contributed to increase the scalability of the original work. SIPCache successfully reduces the resource consumption of the devices and extends the number of participants that can be simultaneously using the service.

6.1 Future Work

Our algorithms present a limited set of requirements what makes them applicable in a broad range of services. In the near future, we plan to continue our research in two directions: $i)$ to devise and evaluate complementary features for our algorithms; and $ii)$ to evaluate its applicability in other services, in particular as a complement to other research trends that have been pursued by the authors.

PADIS guarantees a maximum distance from any node to any data item although requiring a small number of messages. The algorithm does not put a bound on the number of replicas or on the minimal distance, in hops, between them. In the future, we plan to investigate different tradeoffs between the number of transmissions and the memory resources required by the distribution. For example, we plan to devise algorithms that, although requiring additional messages, ensure a minimal distance between the replicas.

The algorithm relies on the Distance between Copies (DbC) constant to define the maximum distance between a node and a replica of each data item. The self-adaptation of the algorithm would be improved if the DbC did not have to be estimated in advance by the system developer. Ideally, the algorithm should be able to change, in run-time, the value of the DbC to attend to variations in the network conditions. Addressing this issue is a complex task. The definition of an adequate DbC depends of multiple variables like the network density, the storage space made available by each node and the number of data items, which can vary with time but also with different regions of the network.

To conclude our prospective future work concerning the improvement of the algorithms, we plan to develop and evaluate other shuffling algorithms, possibly combining properties from some of those already described in this thesis. An aspect that we plan to address is the tradeoff between the size of the data items and the number of items that can be included in an HERALD message. In particular, a challenging scenario considers the case where only keys are advertised in HERALD message due to the size

of the complete items. In this scenario, the decision to shuffle should carefully weight the benefits of the operation with the power consumed for transferring the data item.

Routing protocols for MANETs typically use flooding to broadcast route request messages (Perkins & Royer, 1999; Johnson *et al.*, 2001). Pampa can contribute to attenuate the power consumed on route discovery by reducing the number of nodes required to retransmit the route request messages. However, Pampa may introduce some bias to the route discovery algorithm which can also decrease the performance of the routing protocol and should therefore be carefully weighted. Pampa is expected to increase unfairness as nodes are selected strictly by their geographical coordinates. In the past, we experimented to apply biased delays to the propagation of route request messages of routing protocols. The goal was to improve fairness and load balancing by having unfairly overused nodes to delay the route request message retransmission (Miranda & Rodrigues, 2003; Miranda & Rodrigues, 2005; Miranda & Rodrigues, 2006b). As a future work, the authors will investigate how Pampa can be adapted to simultaneously improve fairness and reduce the number of retransmissions in a broadcast.

The problems of privacy and reputation have been widely addressed in the scope of MANETs. Previously, we proposed an anonymity mechanism for MANETs that binded the users to a virtual pseudonym, thus creating a permanent link between an anonymous user and his reputation (Miranda & Rodrigues, 2006a). One of the limitations of our proposal was the lack of a reliable mechanism for storing data and reputation information in the MANET. To address this problem, our framework required nodes to periodically connect to a trusted reliable server in the wired network. In future work, we expect to benefit of PADIS to create a new anonymity framework that alleviates these constraints.

References

ACHARYA, SWARUP, ALONSO, RAFAEL, FRANKLIN, MICHAEL, & ZDONIK, STANLEY. 1995. Broadcast disks: data management for asymmetric communication environments. *Pages 199–210 of: Proceedings of the 1995 ACM SIGMOD international conference on management of data (SIGMOD '95)*. New York, NY, USA: ACM Press.

BETTSTETTER, C., RESTA, G., & SANTI, P. 2003. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE transactions on mobile computing*, **2**(3), 257–269.

BRAGINSKY, DAVID, & ESTRIN, DEBORAH. 2002. Rumor routing algorthim for sensor networks. *Pages 22–31 of: Proceedings of the 1st ACM international workshop on wireless sensor networks and applications (WSNA '02)*. New York, NY, USA: ACM Press.

CHANG, NICHOLAS, & LIU, MINGYAN. 2004. Revisiting the ttl-based controlled flooding search: optimality and randomization. *Pages 85–99 of: Proceedings of the 10th annual international conference on mobile computing and networking (MobiCom '04)*. New York, NY, USA: ACM Press.

DATTA, ANWITAMAN, QUARTERONI, SILVIA, & ABERER, KARL. 2004. Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in mobile ad-hoc networks. *In:* BOUZEGHOUB, MOKRANE, GOBLE, CAROLE, KASHYAP, VIPUL, *et al.* (eds), *Proceedings of the international conference on semantics of a networked world (IC-SNW'04)*. Lecture Notes in Computer Science, vol. 3226 / 2004. Paris, France: Springer-Verlag Heidelberg.

DRABKIN, VADIM, FRIEDMAN, ROY, KLIOT, GABRIEL, & SEGAL, MARC. 2006. *RAPID: Reliable probabilistic dissemination in wireless ad-hoc networks*. Tech. rept. CS-2006-19. Computer Science Department, Technion - Israel Institute of Technology.

FEENEY, LAURA MARIE, & NILSSON, MARTIN. 2001. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. *Pages 1548–1557 of: Proceedings of the 20th annual joint conference of the IEEE computer and communications societies (INFOCOM 2001)*, vol. 3. Anchorage, AK USA: IEEE.

GHOSE, ABHISHEK, GROSSKLAGS, JENS, & CHUANG, JOHN. 2003. Resilient data-centric storage in wireless sensor networks. *IEEE distributed systems online*, Nov.

GILBERT, SETH, & LYNCH, NANCY. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT news*, **33**(2), 51–59.

HAAS, ZYGMUNT J. 1997 (Oct. 12–16). A new routing protocol for the reconfigurable wireless networks. *Pages 562–566 of: Proceedings of the IEEE 6th international conference on universal personal communications*, vol. 2.

HAAS, ZYGMUNT J., HALPERN, JOSEPH Y., & LI, LI. 2002. Gossip-based ad hoc routing. *Pages 1707–1716 of: Proceedings of the 21st annual joint conference of the IEEE computer and communications societies (INFOCOM 2002)*, vol. 3. IEEE.

HARA, TAKAHIRO. 2001. Effective replica allocation in ad hoc networks for improving data accessibility. *Pages 1568–1576 of: Proceedings of the 20th annual joint conference of the IEEE computer and communications societies (INFOCOM 2001)*, vol. 3. Anchorage, AK: IEEE.

HARA, TAKAHIRO. 2002 (Jan. 8–11). Replica allocation in ad hoc networks with periodic data update. *Pages 79–86 of: Proceedings of the 3rd international conference on mobile data management, (MDM 2002)*.

HARA, TAKAHIRO, MURAKAMI, NORISHIGE, & NISHIO, SHOJIRO. 2004. Replica allocation for correlated data items in ad hoc sensor networks. *SIGMOD Record*, **33**(1), 38–43.

HE, TIAN, BLUM, BRIAN M., STANKOVIC, JOHN A., & ABDELZAHER, TAREK. 2004. AIDA: Adaptive application-independent data aggregation in wireless sensor networks. *Transactions on embedded computing systems*, **3**(2), 426–457.

HUANG, QING, BAI, YONG, & CHEN, LAN. 2006. Efficient lightweight broadcasting protocols for multi-hop ad hoc networks. *In: Proceedings of the 17th annual IEEE international symposium on personal, indoor and mobile radio communications (PIMRC'06)*. Helsinki, Finland: IEEE, for University of Oulu.

IEEE 802.11. 1999. *Wireless LAN medium access control (mac) and physical layer (phy) specifications*. ANSI/IEEE Std 802.11, 1999 Edition. IEEE Computer Society LAN MAN Standards Committee, 3 Park Avenue, New York, NY 10016-5997, USA.

JOHNSON, DAVID B., & MALTZ, DAVID A. 1996. *Mobile computing*. Kluwer Academic Publishers. Chap. Dynamic Source Routing in Ad Hoc Wireless Networks, pages 153–181.

JOHNSON, DAVID B., MALTZ, DAVID A., & BROCH, JOSH. 2001. *Ad hoc networking*. Addison-Wesley. Chap. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks, pages 139–172.

KARP, BRAD, & KUNG, H. T. 2000. GPSR: greedy perimeter stateless routing for wireless networks. *Pages 243–254 of: Proceedings of the 6th annual international conference on mobile computing and networking (MobiCom '00)*. New York, NY, USA: ACM Press.

KRISHNAMACHARI, BHASKAR, & AHN, JOON. 2006 (Apr. 3–6). Optimizing data replication for expanding ring-based queries in wireless sensor networks. *Pages 1–10 of: Proceedings of the 4th international symposium on modeling and optimization in mobile, ad hoc and wireless networks (WiOpt'06)*.

LEGGIO, SIMONE, MANNER, JUKKA, HULKKONEN, ANTTI, & RAATIKAINEN, KIMMO. 2005 (May 23–26). Session initiation protocol deployment in ad-hoc networks: a

decentralized approach. *In: Proceedings of the 2nd international workshop on wireless ad-hoc networks (IWWAN)*. Centre for Telecommunications Research, King's College London, London, UK.

LEVIS, PHILIP, PATEL, NEIL, CULLER, DAVID, & SHENKER, SCOTT. 2004. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. *In: Proceedings of the 1st USENIX/ACM symposium on networked systems design and implementation (NSDI 2004)*.

LI, JINYANG, JANNOTTI, JOHN, COUTO, DOUGLAS S. J. DE, KARGER, DAVID R., & MORRIS, ROBERT. 2000. A scalable location service for geographic ad hoc routing. *Pages 120–130 of: Proceedings of the 6th annual international conference on mobile computing and networking (MobiCom '00)*. New York, NY, USA: ACM Press.

LIM, SUNHO, LEE, WANG-CHIEN, CAO, GUOHONG, & DAS, CHITA R. 2006. A novel caching scheme for improving internet-based mobile ad hoc networks performance. *Elsevier journal on ad-hoc networks*, **4**(2), 225–239.

LIU, CHANGLING, & KAISER, JÖRG. 2003 (Oct.). *A survey of mobile ad hoc network routing protocols*. Tech. rept. 2003-08. Department of Computer Structures - University of Ulm, Germany.

MADDEN, SAMUEL, FRANKLIN, MICHAEL J., HELLERSTEIN, JOSEPH M., & HONG, WEI. 2002. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS operating systems review*, **36**(SI), 131–146.

MIRANDA, HUGO, & RODRIGUES, LUÍS. 2003. Friends and foes: Preventing selfishness in open mobile ad hoc networks. *Pages 440–445 of: Proceedings of the international workshop on mobile distributed computing (MDC'03), in conjunction with the 23rd IEEE international conference on distributed computing systems (ICDCS)*. Providence, Rhode Island: IEEE.

MIRANDA, HUGO, & RODRIGUES, LUÍS. 2005 (June 6–10). Using a fairness monitoring service to improve load-balancing in DSR. *Pages 314–320 of: Proceedings of the*

1st international workshop on services and infrastructure for the ubiquitous and mobile internet (SIUMI'05), in conjunction with ICDCS'2005.

MIRANDA, HUGO, & RODRIGUES, LUÍS. 2006a (July 17–21). A framework to provide anonymity in reputation systems. *In: Proceedings of the 3rd annual international conference on mobile and ubiquitous systems: Networks and services (MOBIQUITOUS 2006)*.

MIRANDA, HUGO, & RODRIGUES, LUÍS. 2006b (June 29–30). A two-side perspective on cooperation in mobile ad hoc networks. *Pages 109–118 of:* JOSÉ, RUI, & BAQUERO, CARLOS (eds), *Proceedings of the conference on mobile and ubiquitous systems*. Escola de Engenharia - Universidade do Minho, Guimarães, Portugal.

PAPADOPOULI, MARIA, & SCHULZRINNE, HENNING. 2001. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. *Pages 117–127 of: Proceedings of the 2nd ACM international symposium on mobile ad hoc networking & computing*. ACM Press.

PERICH, F., UNDERCOFFER, J., KAGAL, L., JOSHI, A., FININ, T., & YESHA, Y. 2004 (Aug. 22–26). In reputation we believe: query processing in mobile ad-hoc networks. *Pages 326–334 of: Proceedings of the 1st annual international conference on mobile and ubiquitous systems: Networking and services (MOBIQUITOUS 2004)*.

PERKINS, CHARLES E., & ROYER, ELIZABETH M. 1999 (Feb.). Ad-hoc on-demand distance vector routing. *Pages 90–100 of: Proceedings of the 2nd IEEE workshop on mobile computing systems and applications*.

PRZYDATEK, BARTOSZ, SONG, DAWN, & PERRIG, ADRIAN. 2003. SIA: secure information aggregation in sensor networks. *Pages 255–265 of: Proceedings of the 1st international conference on embedded networked sensor systems (SenSys '03)*. New York, NY, USA: ACM Press.

RATNASAMY, S., KARP, B., SHENKER, S., ESTRIN, D., GOVINDAN, R., YIN, L., & YU, F. 2003. Data-centric storage in sensornets with GHT, a geographic hash table. *Mobile networks and applications*, **8**(4), 427–442.

RATNASAMY, SYLVIA, KARP, BRAD, YIN, LI, YU, FANG, ESTRIN, DEBORAH, GOVINDAN, RAMESH, & SHENKER, SCOTT. 2002. GHT: a geographic hash table for data-centric storage. *Pages 78–87 of: Proceedings of the 1st ACM international workshop on wireless sensor networks and applications (WSNA '02)*. New York, NY, USA: ACM Press.

ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., & SCHOOLER, E. 2002 (June). *SIP: Session initiation protocol*. Request For Comments 3261. IETF.

SAILHAN, FRANÇOISE, & ISSARNY, VALÉRIE. 2003. Cooperative caching in ad hoc networks. *Pages 13–28 of: Proceedings of the 4th international conference on mobile data management (MDM '03)*. London, UK: Springer-Verlag.

SARKAR, RIK, ZHU, XIANJIN, & GAO, JIE. 2006. Double rulings for information brokerage in sensor networks. *Pages 286–297 of: Proceedings of the 12th annual international conference on mobile computing and networking (MobiCom '06)*. New York, NY, USA: ACM Press.

SCHULZRINNE, HENNING, & ROSENBERG, JONATHAN. 2000. The session initiation protocol: Internet-centric signaling. *IEEE communications magazine*, **38**(10), 134–141.

SHARAF, MOHAMED A., BEAVER, JONATHAN, LABRINIDIS, ALEXANDROS, & CHRYSANTHIS, PANOS K. 2004. Balancing energy efficiency and quality of aggregate data in sensor networks. *The VLDB journal*, **13**(4), 384–403.

SHRIVASTAVA, NISHEETH, BURAGOHAIN, CHIRANJEEB, AGRAWAL, DIVYAKANT, & SURI, SUBHASH. 2004. Medians and beyond: new aggregation techniques for sen-

sor networks. *Pages 239–249 of: Proceedings of the 2nd international conference on embedded networked sensor systems (SenSys '04)*. New York, NY, USA: ACM Press.

TILAK, SAMEER, MURPHY, AMY, & HEINZELMAN, WENDI. 2003 (Nov. 4–7). Non-uniform information dissemination for sensor networks. *Pages 295–304 of: Proceedings of the 11th IEEE international conference on network protocols (ICNP'03)*.

TSENG, YU-CHEE, NI, SZE-YAO, CHEN, YUH-SHYAN, & SHEU, JANG-PING. 2002. The broadcast storm problem in a mobile ad hoc network. *Wireless networks*, **8**(2/3), 153–167.

WU, WEI, & TAN, KIAN-LEE. 2006 (May 10–12). Global cache management in nonuniform mobile broadcast. *In: Proceedings of the 7th international conference on mobile data management (MDM 2006)*.

YIN, LIANGZHONG, & CAO, GUOHONG. 2006. Supporting cooperative caching in ad hoc networks. *IEEE transactions on mobile computing*, **5**(1), 77–89.

Index