Epidemic-Style Information Dissemination in Large-Scale Wireless Networks

Daniela Gavidia
VRIJE UNIVERSITEIT

EPIDEMIC-STYLE INFORMATION DISSEMINATION IN LARGE-SCALE WIRELESS NETWORKS

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op dinsdag 30 juni 2009 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

doort

DANIELA PAOLA GAVIDIA SIMONETTI

geboren te Lima, Peru
promotoren: prof.dr.ir. M.R. van Steen
## CONTENTS

ACKNOWLEDGEMENTS xiii

1 INTRODUCTION 1
  1.1 Drawing inspiration from nature 3
  1.2 Gossiping in Wired Networks 4
  1.3 Gossiping in Wireless Environments 4
  1.4 Research Goals 6
  1.5 Research Methodology 9
  1.6 Outline and Contribution 10

2 THE SHUFFLE PROTOCOL 13
  2.1 Introduction 14
  2.2 System Model 16
    2.2.1 Assumptions 16
    2.2.2 Application Description 17
  2.3 Shuffle Protocol 18
    2.3.1 Protocol Policies 19
    2.3.2 Simulation Setup 19
    2.3.3 Properties 20
  2.4 Performance on the Client Side 27
    2.4.1 Discovery Rate 27
    2.4.2 Probability of seeing an item 28
    2.4.3 Improving Discovery Rate 29
  2.5 Related Work 32
  2.6 Discussion 33
  2.7 Conclusions 34

3 MODELLING THE SHUFFLE PROTOCOL 35
  3.1 Introduction 35
  3.2 A Gossip-based Protocol for Data Dissemination 39
    3.2.1 Protocol assumptions 39
3.2.2 Description ............................................... 40
3.2.3 Properties ............................................... 41
3.2.4 Experimental observations ................................. 42
3.3 Analytical Model of Information Dissemination .......... 44
  3.3.1 Probabilities of state transitions ....................... 44
  3.3.2 Probabilities of selecting and dropping an item ........ 47
  3.3.3 Simplification of $P_{\text{drop}}$ ......................... 49
  3.3.4 Correction factor .................................. 50
  3.3.5 Optimal size for the exchange buffer ................. 51
3.4 Experimental Evaluation .................................. 52
3.5 Modelling with Differential Equations ..................... 54
  3.5.1 Replication ........................................ 55
  3.5.2 Coverage ........................................ 57
3.6 Coverage model (revisited) ................................ 59
3.7 Time Complexity of Experiments ............................ 62
3.8 Conclusions ............................................... 63

4 CANNING SPAM IN WIRELESS GOSSIP NETWORKS ......... 65
  4.1 Introduction ............................................. 66
  4.1.1 The Problem with Spam in Wireless Gossip Networks .. 67
  4.1.2 Motivating example .................................. 68
  4.1.3 Contribution ....................................... 69
  4.2 System Model ........................................... 69
    4.2.1 General Description ............................... 69
    4.2.2 Assumptions ...................................... 70
  4.3 Preparing for the fight .................................. 70
    4.3.1 Access Control .................................... 70
    4.3.2 Source Node Authentication ......................... 71
    4.3.3 Integrity ......................................... 71
    4.3.4 Rate Control ..................................... 72
  4.4 Spamming through the Corruption of Messages ............ 73
    4.4.1 The threat of malicious insiders ..................... 73
    4.4.2 Checking all messages at every hop .................... 73
    4.4.3 Batch Verification ................................. 74
  4.5 Probabilistic Verification ................................ 74
    4.5.1 Selection of entries to verify ....................... 74
    4.5.2 Attack model ..................................... 75
  4.6 Experimental Results .................................... 75
    4.6.1 Amount of Spam in the Network ....................... 77
    4.6.2 Reach of Spam in the Network ....................... 78
  4.7 Related Work ........................................... 79
4.8 Concluding Remarks ............................................. 80

5 ENFORCING DATA INTEGRITY .................................. 83
  5.1 Introduction ..................................................... 83
    5.1.1 The Cost of Guaranteeing Data Integrity ................. 84
    5.1.2 Contribution ............................................. 84
    5.1.3 Related Work ........................................... 85
  5.2 System Model .................................................. 86
    5.2.1 Example: Gossip-based News ............................. 87
    5.2.2 Probabilistic Verification ............................... 87
    5.2.3 Verifying the Integrity of Data in a Dynamic Environment............... 88
    5.2.4 Experimental Setup .................................... 89
  5.3 Malicious vs. Benign Behavior ................................. 90
    5.3.1 Attack Model ........................................... 90
    5.3.2 Can Malicious Nodes Be Identified? ..................... 90
  5.4 Effect of Spammers on the Network ............................. 92
    5.4.1 The Effect of Alpha .................................... 92
    5.4.2 Varying Spamming Rates ................................ 93
    5.4.3 Workload Caused By Spammers ......................... 95
  5.5 Detecting Malicious Behavior .................................. 96
    5.5.1 Detection Mechanism ................................... 96
    5.5.2 Flying under the Radar .................................. 99
  5.6 Conclusions .................................................... 101

6 BROADCAST-BASED EPIDEMICS .................................... 103
  6.1 Introduction .................................................. 104
  6.2 Related Work .................................................. 106
  6.3 System Model .................................................. 108
  6.4 SharedState ................................................... 110
    6.4.1 The Protocol .......................................... 111
    6.4.2 Active Thread ......................................... 111
    6.4.3 Passive Thread ........................................ 113
    6.4.4 Early Strategies ....................................... 113
  6.5 Basic Properties .............................................. 115
    6.5.1 Discovery Rate ......................................... 115
    6.5.2 Fairness in Replication ................................. 118
  6.6 Comparison to the Shuffle Protocol ............................ 118
  6.7 Density Awareness ............................................ 120
  6.8 Performance Evaluation ....................................... 122
  6.9 Uniform Node Distribution .................................... 124
    6.9.1 Discovery Rate and Node Density ...................... 124
## List of Figures

2.1 Skeleton of an epidemic protocol. ........................................... 18
2.2 Convergent behavior illustrated by having 10 nodes that generate news items in a network of 10000 nodes (arranged as a 100 × 100 grid). ................................................................. 22
2.3 Number of rounds required for all the routers in the backbone to have seen an item using shuffling on a grid topology. .............. 22
2.4 Number of rounds required for all the routers in the backbone to have seen an item using shuffling on three different topologies. ... 23
2.5 Number of rounds required for all the nodes in the network to have seen an item for different cache sizes with (a) grid topology (range 1), (b) grid topology (range 2) and (c) random topology (range 2). All entries in the cache are exchanged. ......................... 24
2.6 Number of entries per news item. 49% of the routers go down at round 200 and recover at round 300. ................................. 25
2.7 Average number of entries per item. Routers go down at round 200 and recover at round 300. ................................. 25
2.8 Recovery times when an increasing number of routers fail. ......... 27
2.9 Discovery rate of news items. All entries in the cache are exchanged. 29
2.10 Probability of finding an item in a randomly filled cache. ......... 30
2.11 Discovery rate of news items. ........................................... 31
2.12 Discovery rate of news items. ........................................... 31
3.1 Distribution of replicas over time. ........................................ 43
3.2 Symbolic representation for caches of gossiping nodes. .......... 45
3.3 k items in $S_A \cap C_B$ .................................................. 47
3.4 $\hat{s}$ items in $S_A \cap S_B$ .................................................. 47
3.5 The relative error of the difference of the accurate $P_{drop}$ and its approximation, for $c = 250$ (left) and $c = 500$ (right) and different values of $n$. .................................................. 50
3.6 Optimal value of exchange buffer size for $c = 100$, depending on $n$. 52
3.7 The shuffle protocol (left) and the model (right), for $N = 2500$, $n = 500$, $c = 100$ and different values of $s$. .......................... 54
3.8 Percentage of nodes in the network with a replica of item $d$ in their cache, for $N = 2500$, $c = 100$, $s = 50$, and $n = 500$, $n = 1000$ or $n = 2000$. .................................................. 56
3.9 Percentage of nodes in the network that have already seen a replica of item $d$, for $N = 2500$, $c = 100$, $s = 50$, and $n = 500$, $n = 1000$ or $n = 2000$. .................................................. 59
3.10 Distribution of nodes according to the number of shuffles they execute per round, for a fully connected network of $N = 2500$ nodes. ............................................. 60
3.11 Percentage of nodes in the network that have already seen a replica of item $d$, for $N = 2500$, $c = 100$, $s = 50$, and $n = 500$, $n = 1000$ or $n = 2000$. .................................................. 62
4.1 Spread of spam after 5, 50 and 450 rounds of gossiping. Nodes are arranged in a 50 x 50 grid with 1% of nodes being spammers. The level of pollution of their caches is indicated by the height and brightness of the surface. ........................................ 67
4.2 Number of corrupted entries in the network when 1% of nodes are malicious. .................................................. 76
4.3 The percentage of corrupted messages in the network decreases with the probability of checking an entry, $P_{\text{check}}$. ................. 76
4.4 Spread of spam after 450 rounds since the appearance of 25 spammers (1% of nodes in the network). ................................. 77
4.5 Average distance (in hops) of corrupted entries from their source over time (in rounds), with 1% of malicious nodes in the network. 79
4.6 Average distance (in hops) of corrupted entries from their source for different values of $P_{\text{check}}$. Results for spammers accounting for 1%, 2% and 5% of nodes in the network are shown. ................. 80
4.7 Distribution of corrupted entries according to their average distance (in hops) from the source. ........................................ 81
5.1 If a node applies probabilistic verification properly, the amount of spam and checked items in its cache should be similar. ............... 91
5.2 Percentage of corrupted entries in the network over time for different values of $\alpha$. .................................................. 93
5.3 Amount of corrupted entries in the network over time, for different spamming rates ($\alpha = 0.90$). ........................................ 94
5.4 Average number of corrupted entries in the network for different spamming rates. .................................................. 95
5.5 Histogram of average values of $P_{\text{check}}$ for every node in the network (malicious nodes excluded). 250 malicious nodes insert corrupted entries with $P_{\text{spam}} = 0.90$. .......................... 96

5.6 Results of applying the detection algorithm over time for different values of $\delta$: a) $\delta = 0.05$, b) $\delta = 0.10$, c) $\delta = 0.15$ and d) $\delta = 0.20$. .......................... 98

5.7 Spammers try to prevent being discovered by executing integrity checks: a) average value of $\text{diff} \pm \sigma$ for spammers and normal nodes as neighbors ($P_{\text{spam}} = 0.5$, $\delta = 0.20$), b) spammers that avoid detection. .......................... 98

5.8 Spammers try to avoid detection by lowering their spamming rate to 0.1. .......................... 100

5.9 Amount of corrupted entries in the network over time for different malicious behaviors. .......................... 101

6.1 Visual representation. .......................... 109

6.2 SharedState pseudocode. .......................... 112

6.3 Discovery rate for different rates of cache size $c$ versus number of items $d$. .......................... 116

6.4 Histogram of the number of replicas per item in a network of 900 nodes with 180 items (cache size = 18). .......................... 117

6.5 Number of replicas for one item published at round 300 in a network of 900 nodes with 180 items (cache size = 18). .......................... 117

6.6 Discovery rate for different rates of cache size $c$ versus number of items $d$: a) SharedState and b) Shuffle. .......................... 119

6.7 Number of replicas for one item published at round 300 in a network of 900 nodes with 180 items (cache size = 18): a) SharedState and b) Shuffle. .......................... 120

6.8 Discovery rate over time for networks of different sizes: a) sparse topologies, b) dense topologies. .......................... 124

6.9 Average number of broadcasts received per node for networks of different sizes. .......................... 125

6.10 Number of entries in the input buffer after receiving $k$ transmissions ($s = 4, d = 80$). .......................... 126

6.11 Comparing the original algorithm and the improved version in terms of discovery rate for: a) 256 nodes and b) 784 nodes. .......................... 127

6.12 Average number of broadcasts received per node for networks of different sizes. .......................... 128

6.13 Number of broadcasts sent per round for networks of different sizes. .......................... 128

6.14 Layout of a network of 576 nodes with a higher density centered at (50, 50). Nodes with a higher $\text{ProbSkip}$ appear in a darker tone. .......................... 129
LIST OF FIGURES

6.15 Comparing the original algorithm and the improved version in terms of discovery rate. .................................................. 130
6.16 Distribution of nodes according to their probability of skipping a round. ................................................................. 130
6.17 Probability of skipping a round in relation to the distance from the center. ............................................................... 131

7.1 Discovery rate for different rates of cache size $c$ versus number of items $d$: a) Shuffle and b) modified Shuffle. ............... 134
ACKNOWLEDGEMENTS

First of all, I would like to thank all my friends and colleagues. I know I will forget somebody if I try to name all of you, but you know who you are. If you are reading this, you probably fit here. We see each other in the hallway and the coffee room, have lunch together and, sometimes, even talk about our work. These last few years would not have been as fun and enjoyable without you.

Amsterdam is such a lovely city. It is no surprise that so many people want to live here, making it a struggle to find a nice place to live. I consider myself lucky in this regard, as I have (mostly) lived in nice places. But, most importantly, I have had the pleasure of sharing a roof with some great people. A tremendous thank you goes to the people that have had to put up with me at one time or another: Ting and Hailiang, Rosanne, Rena and Lucas. You are some of my favorite people in the world.

I would also like to express my gratitude to the members of my reading committee, Luis Rodrigues, Koen Langendoen, Wan Fokkink, Spyros Voulgaris and Rob van der Mei. I know you are all busy people and I thank you for taking the time to read my dissertation and provide valuable feedback. Your input was very helpful for finalizing my dissertation.

A very special thank you goes out to Maarten van Steen, my supervisor. Your enthusiasm for research is contagious. Things always seem easier and make more sense after discussing them with you. Thank you for your support and encouragement through all these years and for allowing me to do things at my own (somewhat slow) pace.

And last, but most definitely not least, I would like to thank my family. A big thanks to my mother and sister (who designed the cover!) for always being supportive of me in whatever I do. For my mom: Muchísimas gracias por tu apoyo y tu fe en mí. A pesar de estar lejos, sé que siempre piensas en mí y me deseas lo mejor. Te quiero mucho.

Daniela Gavidia
Amsterdam, May 2009
Modern-day computers are the result of a fast-paced and sometimes unpredictable evolution. The twentieth century saw the term applied to wildly different devices, from massive number-crunchers like the ENIAC, which filled an entire room and weighted thirty tons, to portable laptop computers that could fit in a backpack and be lifted with one hand. The advances in technology that allowed for computers to be shrunk in size also made them more affordable, making them a ubiquitous presence in the home and at the office. Furthermore, the continued miniaturization of processing units has given rise to embedded systems, just as powerful as personal computers of a few years ago. These special-purpose computer systems can be found anywhere from mobile phones to kitchen appliances and it is not unreasonable to think that their numbers will increase in coming years. In the same way as the widespread use of personal computers prompted the development of computer networks (such as the Internet), we can envision that the ubiquitous presence of powerful embedded devices will lead to the desire to network them and harness their collective power to create innovative applications.

Personal computers are often networked by means of a fixed infrastructure of routers and switches. In the physical world, copper and fiber optic cables are needed to connect all these devices together. With the advent of portable, mobile devices, this type of connectivity becomes a limitation, leading to the use of the wireless spectrum to connect to the infrastructure. We can view this as a wireless extension to the wired network rather than an autonomous wireless network. Purely wireless networks, unable to rely on the services provided by the infrastructure, face their own particular challenges. Chiefly among them is the need to self-organize in order to carry out a task. In this dissertation, we focus on purely wireless networks and, specifically, on how to disseminate information effectively and efficiently in these types of networks.

Devices capable of wireless communication run the gamut from powerful lap-
tops to simple wireless sensor nodes. As can be expected, different types of devices are used with different objectives in mind. Nevertheless, in order to form self-reliant autonomous networks, these devices need to communicate using robust distributed algorithms. The characteristics of the wireless medium (limited range of the radios, unreliable communication, dynamic topologies) complicate the use of centralized solutions for anything other than small deployments. While centralized solutions might work well in a small setting (such as a group of users with laptops in a coffee shop), we envision that the ubiquity of wireless embedded systems will lead to the possibility of having large-scale wireless networks in the order of thousands of nodes. For these networks, centralized administration would be anything but trivial.

The choice of a communication model for large-scale wireless environments is largely influenced by the characteristics of the medium itself, namely the limited range of the radios, unreliable communication and dynamic topologies. These three aspects suggest that the chosen communication model should have certain characteristics:

- **Robustness** Given the unreliable nature of the communication channel and the nodes themselves, the network should not depend on the presence of any one node to carry out a desired task. It is a desirable property that the network can continue to operate effectively under adverse conditions such as sudden decreases in membership (which could be caused by nodes moving out of range or nodes running out of power) or degradation of the communication channel (due to congestion or interference). Graceful degradation of the network operation under unexpected network changes should be a goal and intrinsic to the chosen communication model.

- **Scalability** Unless the network is deployed in a controlled environment, it is not easy to predict the form that it will take. The number of participants may vary over time and, with potentially thousands of participants, it is desirable for the network to be able to handle large numbers of participants regardless of the way they are arranged (very sparsely or physically close). This suggests the need for algorithms that impose a fixed load on each node which should remain unaffected with regard to the size of the network.\(^1\)

- **Locality** Key to achieving a scalable network is the ability of nodes to carry out their tasks using only locally available information. As neighborhoods are defined by physical proximity, communication within a neighborhood is

\(^1\)In this dissertation, we assume that the nodes in our networks are similar in terms of computational power, storage space and energy consumption. For heterogeneous networks, where some nodes may be considerably more powerful than others, having a load distribution proportional to the nodes’ capabilities may be more appropriate.
bound to be more reliable and immediate than communication with distant nodes. Given that we envision the creation of very large networks, it is crucial that a node does not rely on feedback from nodes outside its immediate neighborhood.

1.1. DRAWING INSPIRATION FROM NATURE

A fundamental problem in wireless environments is efficient information dissemination. Although this might seem trivial in a small network, complications arise when dealing with very large deployments. As pointed out in [Ni et al. 1999], plain flooding is guaranteed to cause congestion in the medium leading to poor communication. While structured approaches, like building multicast trees and overlays, have been proposed to alleviate the problem, we argue that these approaches are too fragile, especially for large systems. In a wireless setting, a single node failure could invalidate several paths. A route repair algorithm would then be triggered. This approach is acceptable in a wired setting, where the topology of the network is fairly stable and the communication overhead for repairing the structure is low. On the other hand, wireless environments are prone to experiencing failures and disconnections which could turn into a continuous cycle of build/break/repair steps. Additionally, the cost per node for maintaining a structure can increase with the size of the network, for example, when optimal structures, such as a shortest-path tree, are desired. In cases like these, a link failure may have far-reaching implications, as a large number of nodes may have to be updated in order to repair the structure. While a structured approach could be suitable for a small deployment (tens of nodes), large wireless networks require more robust solutions.

The challenges presented by wireless environments can be tackled by drawing inspiration from nature. Biological systems composed of large numbers of elements regularly organize themselves to achieve a task without a central authority. These systems (e.g., a collection of cells or a population of organisms) are fully distributed, adaptive to changing conditions, resilient to failures of individual elements, and based on local interactions. By devising algorithms inspired by nature, we can obtain these useful properties for free and, at the same time, be assured that our solution has been tested and proven to be successful over generations. In this work, we take a cue from epidemic theory to tackle the information dissemination problem in wireless environments.
1.2. GOSSIPING IN WIRED NETWORKS

Epidemic techniques were first introduced into the realm of distributed systems in the pioneering work of [Demers et al. 1987] in the late 80’s. As part of the Clearing House project, an epidemic protocol was used to remove inconsistencies in tables in wide-area database systems. Since then, popularity of epidemic protocols in the distributed systems domain has flourished. In addition to their elegant simplicity, the appeal of epidemic protocols lies in the fact that they can be easily implemented in a fully decentralized way and exhibit desirable properties, namely reliability, robustness and scalability.

In most of the current literature, the terms gossip and epidemics are generally used interchangeably. Formally, gossiping is a subgroup of epidemic protocols. Analogous to rumor spreading in real life, gossiping means that a rumor is spread when entities interact. This interaction occurs at random and each time the rumor is communicated, the receiving entity will spread it further with a certain probability. This results in the rumor being spread rapidly, but without any hard guarantees that it will reach all entities. In computer systems, the definition of gossip is not as clear [Costa et al. 2007]. Prototypical characteristics of a gossip protocol include: random peer selection, periodic execution and symmetry (meaning that the same algorithm is executed by all nodes). However, variations based on these characteristics have been developed according to the specific goal of the protocol.

Since their introduction into the domain of distributed systems, a wealth of work in epidemic protocols has been produced. The majority of this work has been designed with traditional wired networks in mind, in part fueled by the rise in popularity of peer-to-peer applications. Like wireless networks, peer-to-peer networks can be highly dynamic. Due to their unpredictable nature, an unstructured communication model like gossip is an appealing solution and has been successfully applied to problems ranging from overlay construction and maintenance [Jelasity et al. 2003; Voulgaris et al. 2005] to membership management [Jelasity et al. 2004] and aggregation [Jelasity and Montresor 2004; Montresor et al. 2004].

1.3. GOSSIPING IN WIRELESS ENVIRONMENTS

Given the successful experiences in applying epidemic techniques to tackle a variety of problems in wired networks, it is not surprising that these principles were soon applied to the wireless domain. It can be argued that epidemic communication is a natural fit for wireless networks, where information can be exchanged only between nodes within physical proximity, resembling the way information is
disseminated through social interactions. Like in social networks, nodes in a wireless network may join and leave frequently, and often unpredictably, resulting in dynamic network topologies. As a consequence, the delivery of a message to a particular destination might only be possible with the intervention of other nodes as intermediaries. While there might be several paths to the destination, the most optimal path might not always be evident without up-to-date knowledge of the topology and network conditions. The complications of obtaining this information, especially as the network grows, makes an epidemic approach appealing, in that nodes proactively spread the message through several paths leading to a high probability of it reaching the destination.

Early work on applying epidemics in wireless environments concentrated on broadcasting. Delivering a message to all nodes in the network can be achieved easily through flooding, where each node that receives the message for the first time rebroadcasts it to its neighbors. As demonstrated in [Ni et al. 1999], plain flooding can be specially troublesome in wireless environments, bringing upon problems with redundant broadcasts, contention and collisions. The authors categorize the possible solutions to these problems as probabilistic, counter-based, distance-based, location-based and cluster-based schemes. Probabilistic flooding schemes, where nodes rebroadcast a message after receiving it for the first time with a certain retransmission probability, result in an epidemic spread of the message. In their landmark paper [Haas et al. 2002], Haas and colleagues used an epidemic (or gossip-based) approach for the dissemination of routing messages, showing that gossiping can use up to 35% fewer messages than flooding. They also demonstrated that there is a value for the retransmission probability that roughly determines whether a message is delivered to almost all the nodes or dies out. This bimodal behavior when using epidemics for broadcasting was also studied in [Sasson et al. 2003] and has been exploited to achieve reliable broadcast in wireless networks [Wang et al. 2005]. More recent work [Drabkin et al. 2007] shows that, by adding deterministic correction actions to the probabilistic dissemination and using node density to adjust the retransmission probability, high delivery rates can be achieved and selfish and malicious behavior can be handled as well.

Besides broadcast flooding, gossiping has been applied to a variety of other areas, such as reliable multicast [Luo et al. 2003] and information dissemination [Khelil et al. 2002; Datta et al. 2004]. Anonymous Gossip, one of the earliest works on gossip-based multicast for ad hoc networks, improves the reliability of the underlying best-effort multicast protocol (MAODV [Royer and Perkins 1999]) by using gossip to recover missing messages. By contrast, Route Driven Gossip [Luo et al. 2003] uses a purely gossip-based approach to multicasting, relying solely on a basic unicast routing protocol, e.g., Dynamic Source Routing (DSR),
being available. Autonomous Gossiping [Datta et al. 2004] uses gossip to update node profiles. These profiles are used to decide whether data items should be kept, replicated or migrated to another node with a more suitable profile. In the area of wireless sensor networks, Heinzelman and colleagues [Heinzelman et al. 1999] studied gossiping and flooding for information dissemination and compared them against their own dissemination protocol, which relies upon meta-data negotiations to reduce redundant data transmissions.

A concrete application area where gossiping shows promising results is code propagation. Updating software code in resource-constraint wireless sensor networks is a specially daunting challenge. Given that a deployed sensor network is expected to operate for long periods of time, updating the code that nodes run may be required. While this might be easily achieved in a small network, with increasing network size and the inclusion of mobile nodes the problem becomes more complex. Recognizing the need for fully decentralized and robust algorithms for code propagation, Trickle [Levis et al. 2004] and, more recently, GCP [Busnel et al. 2007] use gossip-based dissemination to deliver software updates.

1.4. RESEARCH GOALS

In this dissertation, we explore themes related to the use of epidemic techniques in wireless networks. Despite the similarities between peer-to-peer networks and wireless networks, wireless networks display peculiar characteristics stemming from the communication medium and the limitations of most mobile, portable or embedded devices. Because of this, the analysis of epidemic algorithms in wired networks does not necessarily hold for wireless environments. Wired networks offer a variety of services that are taken for granted by system designers, for instance naming and routing. A variety of projects to bring basic services to wireless networks have been undertaken over the years, but standard methods are still not widely agreed on. The wide variety of wireless platforms, with their varying capabilities and limitations, suggests that a one-size-fits-all solution is unlikely. With this in mind, the work presented in this dissertation focuses on minimalistic epidemic protocols that rely solely on local interactions.

In particular, we focus on the problem of information dissemination, as it is a basic service that serves as a building block for other services (e.g., code propagation or building routing tables). The wireless medium introduces a variety of obstacles that complicate the dissemination of data. As mentioned earlier, unpredictable topology changes make structures hard to maintain. Mobility and node failures may even cause temporary partitioning of the network. Links between nodes are not always reliable. For these reasons, the dissemination of informa-
tion is not just a matter of getting the information from a central repository or even flooding the network. This becomes more apparent as we consider increasingly larger networks. Successful dissemination of information in a wireless environment calls for the collaboration of the nodes in the network in a distributed manner. We believe that gossip protocols are a natural fit for disseminating data over large-scale wireless networks and can overcome the problems posed by their unpredictable behavior.

The particular research questions we address are the following:

- **How would a gossip protocol inspired by peer-to-peer behave in a wireless environment and to what extent would the switch from a wired to a wireless environment affect the desired properties of the protocol?**

When it comes to wireless environments, the limited communication range of radios restricts the randomness of the peer with which a node can gossip. Intuitively, we can infer that this alone will cause an epidemic algorithm to perform differently in wireless environments compared to wired ones. However, the effect that restricted connectivity (dictated by geographical proximity) will have on the emergent properties of the protocol cannot be easily predicted.

As a first step in our study of epidemic protocols, we propose a gossip protocol inspired by Cyclon [Voulgaris et al. 2005], a gossip protocol used for membership management in wired networks. While Cyclon gossips the addresses of nodes to maintain an overlay with the properties of a random graph, we use a similar mechanism to disseminate data items (instead of node addresses) through the network. We analyze the characteristics of the dissemination, scalability with respect to data items in the network, and resilience to failures.

- **Can the emergent behavior of the protocol be explained through formal analysis?**

In order to observe a protocol’s particular characteristics, two paths can be followed: a testbed deployment or a simulation study. The testbed approach has the benefit of providing a realistic experience, using the real wireless medium instead of a model and requiring the protocol to be tailored for use in actual nodes with specific resource constraints. On the downside, building a testbed can be expensive and, for this reason, testbeds are often small (tens of nodes). In addition, once the testbed is deployed, making changes may be cumbersome and time-consuming. Using simulations, the protocol can be tested in very large networks and making topology changes becomes trivial.

Due to their probabilistic nature, the characteristic behavior of an epidemic protocol cannot be easily predicted during the design phase. In order to discover
the trademark properties of an epidemic protocol, large networks (where trends can be observed) are needed. Simulations allow us to execute our epidemic protocols in large networks and easily change parameters and settings. However, even though simulations allow us to discover emergent behavior, we cannot be sure that the protocol will continue to exhibit the observed characteristics under a different set of parameters and settings. Moreover, the relationship between the parameters can only be inferred, but not proven.

We believe that formal analysis of the interactions between two gossiping nodes can provide insight into the mechanics behind the emerging behavior of epidemic protocols, shedding some light into the relationship between system parameters and ultimately enabling us to predict behavior and (hopefully) even shape it to obtain the desired results.

**What are appropriate counter measures against misbehaving or faulty nodes when gossiping in wireless networks?**

A fundamental property of gossip protocols is symmetry, which suggests that all nodes in the network are acting in good faith by executing the exact same protocol. In other words, there is implicit trust in the interaction between a node and its neighbor. This raises questions such as what would happen if this trust is broken and can a gossiping network remain in operation in the presence of malicious nodes?

The problem of malicious peers in a gossiping peer-to-peer network has proven to be complex and seriously damaging. In a wired network, nodes can resort to contacting a trusted party if they suspect that a neighbor is not conforming to the specified protocol [Jesi et al. 2006]. However, nodes in a wireless network, restricted to communicate only with their nearby neighbors, do not have this advantage. The wireless environment calls for local decision-making when tackling such a problem. We test the severity of an attack by malicious nodes compromising the integrity of the data items being disseminated and, in light of the devastating results, we propose the use of probabilistic verification of messages. Additionally, we propose measures for detecting and, ultimately, isolating nodes that deviate from the expected behavior.

**To what extent can we obtain the same emergent behavior of a gossip protocol with a broadcast-based probabilistic communication model?**

Random peer selection is a defining characteristic of gossip protocols, as implemented in wired networks. While it is possible to randomly select a neighbor to gossip with in a wireless environment, the wireless medium is naturally suited for
broadcast communication. Broadcasting, as opposed to point-to-point communication, does not require continuous neighbor discovery or for nodes to have unique ids. Once a message is sent, the recipients are determined by the link quality to the source at the moment of transmission. The simplicity of a broadcast-based protocol is appealing as it lowers the requirements on the wireless platform. A broadcast-based protocol can therefore be easily implemented for use in the most simple resource-constrained nodes.

The move from a push/pull gossip protocol to a broadcast-based protocol that retains the same emergent properties requires that the probabilistic behavior be shifted from the peer selection to the data management at the receiving node. The challenge of data management at reception lies in the fact that the incoming traffic that a node will receive is proportional to the number of neighbors it has, while the outgoing data that the node can send stays the same. This imbalance between the amount of incoming and outgoing traffic opens the door for experimentation in data management strategies.

1.5. RESEARCH METHODOLOGY

At first instance, the ideas put forward in this dissertation are evaluated using an experimental approach. Given that we are interested in the emergent properties of gossip protocols in large wireless networks, we test and validate our ideas through large-scale simulations. After gaining a thorough understanding of the observed behavior through experimentation, we attempt to describe the behavior of the protocols through theoretical analysis.

For the experimental part of our work, two simulation environments were used: Peersim [PeerSim 2008] and TOSSIM [Levis et al. 2003]. Peersim is an event-based simulator developed for the testing of peer-to-peer protocols in large-scale settings. In order to simulate wireless communication, we tailored our Peersim environment to create neighborhoods according to radio range. In other words, nodes are able to communicate only with nodes within a specified radio range (modeled as a disk). Peersim allowed us to explore multi-hop communication and its effect on the speed of data dissemination. Later on, the need to explore the effect of a more realistic radio model led us to use TOSSIM (short for TinyOS SIMulator). Using TOSSIM, a theoretical propagation model is applied to a physical layout of nodes providing a more realistic model of radio communication. As a consequence, the experimental results obtained using TOSSIM are closer to what we would expect from a large-scale deployment in real life.

The theoretical part of our work consists of modeling gossip protocols using Markov chains. The model is validated by comparing the results with a battery
of simulation experiments showing that, statistically, the model reproduces the behavior of the protocol. In addition, specific properties observed in large-scale data dissemination experiments are modeled and contrasted against simulations.

1.6. OUTLINE AND CONTRIBUTION

The following list presents the forthcoming chapters of this dissertation and summarizes their contribution.

Chapter 2 — The Shuffle Protocol Chapter 2 introduces the shuffle protocol, a gossip protocol for information dissemination in wireless mesh networks. In the spirit of peer-to-peer, the shuffle protocol follows a push/pull model in the same vein as Cyclon [Voulgaris et al. 2005]. We propose the use of the shuffle protocol to build a fully decentralized news service which is shown to be scalable and resilient to failures.

Chapter 3 — Modeling of the Shuffle Protocol - This chapter delves into a probabilistic analysis of the interactions between gossiping nodes executing the shuffle protocol. We develop a model of the interactions and show, through an extensive simulation study, that the model can faithfully reproduce the characteristic behavior of the dissemination of a data item. A closer look at the model reveals the precise relationships between system parameters, which we use to find optimal values for the parameters of the protocol.

Chapter 4 — Canning Spam - Building on the gossip-based shuffle protocol, Chapter 4 explores the implications of trusting neighboring peers blindly. After observing the damage that can be done by just a few malicious nodes in the network, we implement probabilistic security measures to curtail the negative effects of malicious behavior and provide a basic level of data integrity.

Chapter 5 — Enforcing Data Integrity - This chapter follows along the path set by Chapter 4 and presents an adaptive method for applying integrity checks with the goal of having stronger security measures when exchanging data with a suspicious node. After successfully proving that security measures can be adjusted according to the threat posed by a neighbor, we introduce a method for identifying and isolating misbehaving nodes.

Chapter 6 — Broadcast-based Epidemics - Radio communication is inherently broadcast-based. With this in mind, we move away from point-to-point communication in favor of a broadcast-based model while attempting to retain the useful properties of the shuffle protocol. In this chapter, we present
SharedState, a protocol for probabilistic replication of data items. In addition, we propose a technique for reducing unnecessary communication in order to optimize the conservation of resources.

To conclude, Chapter 7 presents a discussion about the challenges encountered during the development of the epidemic protocols introduced in this dissertation. From this discussion, we draw out the lessons we have learned regarding the nature of gossiping in wireless networks and delineate possible directions of future research.
CHAPTER 2

The Shuffle Protocol

The widespread acceptance of Internet technology has made it easier than ever to interconnect computers from all over the world, opening the doors to the creation of large ad hoc networks. The appearance of the first peer-to-peer applications proved that large numbers of computers could indeed create a network of collaborating peers. Those first successes sparked great interest in the deployment of truly large-scale distributed systems and also brought to light a major challenge: finding robust and scalable decentralized algorithms. Algorithms with those characteristics are needed to ensure that the network can add and lose nodes organically without much impact to its performance and that its operation is not compromised by the failure of any single node. Epidemic protocols are inherently distributed, scalable and have proven to be robust enough to handle the constant coming-and-going of nodes. As an added benefit, they are very simple and easy to implement. For these reasons, they have been used in recent years to tackle a variety of problems, such as membership management [Jelasity et al. 2004] and aggregation [Jelasity and Montresor 2004; Montresor et al. 2004].

Wireless ad hoc networks share many characteristics with these internet-based peer-to-peer networks. Unexpected failures, constant or sudden changes in membership and unreliable links, all contribute to creating dynamic, unpredictable topologies. Like peer-to-peer networks, wireless ad hoc networks require robust, scalable algorithms that can tolerate these problems. Given that epidemic protocols have been successfully used to build services in large-scale peer-to-peer networks, in this first chapter we explore using the same principles in a wireless environment.

We gather inspiration from Cyclon [Jelasity et al. 2004], a membership management protocol based on random gossiping. In Cyclon, each node keeps a small list of node addresses. By periodically exchanging a subset of its node addresses with a node from the list, each node is able to update its list, such that the over-
lay graph created by the links between each node and the addresses in its list has the characteristics of a random graph. As a result, by taking part in the Cyclon gossiping any node can easily join and become part of the peer-to-peer overlay.

In this work, we address the fundamental problem of information dissemination. Cyclon, like many gossip protocols, is essentially used for data dissemination. In particular, it takes care of disseminating node addresses to keep the overlay connected. In a wireless environment, connectivity is dictated by the physical proximity between nodes. Communicating with a node that is outside of radio range is therefore more expensive, as it requires multiple hops. Without cheap routing, maintaining a Cyclon overlay in a wireless environment would be prohibitively expensive. Instead, we use the principles of the Cyclon data exchange to disseminate data items through the wireless network. The choice of nodes to gossip with in a wireless network is restricted to a node’s nearby neighbors, but as long as the physical network is not partitioned, random gossip with nearby neighbors should result in epidemic dissemination of items through the network. This chapter studies the characteristics of this dissemination under the assumption of a stable wireless mesh network.

2.1. INTRODUCTION

As advances in wireless networking continue, we are gradually seeing a shift in which distributed (middleware) systems are moving from wired networks to heterogeneous or completely wireless systems. Notably, wireless mesh networks [Akyildiz et al. 2005] offer the facilities to quickly and cheaply set up a networking infrastructure that can easily span the size of a city. From a distributed systems perspective, the challenge lies in providing services that can hide the inherent unreliable nature of the underlying infrastructure. This unreliability is caused by failing links and a relatively high rate of joining and leaving nodes (purposefully or unintentionally), which continuously affect the topology of the network.

This instability requires that we seek new solutions to well-known problems. As a step in that direction, we are exploring how gossiping protocols can help in the construction of highly robust services. In this chapter, we consider the problem of providing a news service that runs entirely on a wireless mesh network. This service provides mobile users news items that are of interest to them. In our approach, we assume that a user, by means of a PDA or a similar small device, can connect to an access point (i.e., a router) of a wireless mesh network. When connected, the user can read news items as if accessing a central database where all items are stored. Using content-based filtering, for example by means of SQL-like queries, only the items of interest will be delivered.
The problem we address can best be described as setting up a simple, self-configuring news service in a mesh network, under the condition that it be fully decentralized. The reasons for avoiding a centralized implementation are, in a way, related to the nature of mesh networks. Wireless mesh networks are based on the principle of cooperation between routers, most notably exemplified by routers forwarding packets on behalf of other nodes. With that in mind, we want to steer away from a centralized solution where one node is solely responsible for the availability of the service. A decentralized solution effectively divides the workload (and responsibility) among the collection of nodes providing the service, allowing us to sidestep issues that may arise from having a single point of failure and single ownership of the service. We outline the requirements for a successful implementation of our distributed news service as follows:

- **Ease of deployment** A collection of nodes should be able to start providing the service with minimal configuration. Nodes should be able to join the system without going through complicated bootstrapping mechanisms. In essence, we desire to have a decentralized system where nodes can start making a contribution to the service as soon as they are operational.

- **Minimal requirements** Contributing to the service should not be a burden to the nodes in the mesh network. Memory and computational requirements should be small enough to allow any router to be part of the service. No powerful nodes are expected to be in place for high-performance tasks.

- **Robustness** The system should be minimally affected by nodes joining and leaving the network. Moreover, recovery from significant changes in membership should be prompt.

- **Scalability** The service should be able to perform adequately in the face of increasing number of nodes and news items being published.

- **Effectiveness** When an item is published, it should be made available to the interested users in a timely manner.

We expect to meet these requirements by having the routers in the mesh backbone exchange news items using the epidemic protocol we introduce in Section 2.3. Epidemic (or gossip-based) techniques have proved to be a robust, efficient, and scalable solution for disseminating information in peer-to-peer networks [Demers et al. 1987; Birman 2003; Jelasity et al. 2004]. Aside from the robustness and scalability inherent to gossiping, the protocol we present is characterized by simple, independent one-to-one interactions. The simplicity of this approach allows any router willing to participate in the service to start contributing as soon as they come in contact with a router that is already providing the service.
By basing our solution on gossiping, we expect to be less vulnerable to topology changes.

Our main contribution is that we embrace the unpredictable nature of wireless networks and attempt to use this to our advantage by implementing a gossip-based solution. Our approach skews deterministic routing in favor of probabilistic delivery of news. As a result, we can deliver a scalable and robust service with predictable behavior for large-scale deployments.

2.2. SYSTEM MODEL

The service we propose is provided by a mesh backbone composed of a large number of wireless routers. Users running the news service are able to publish events, which we call news items, of interest to other users. These users carry around clients, which are portable devices capable of connecting to the mesh backbone to retrieve news items. Essentially, the clients poll the routers for news items matching the interests of users. By specifying their preferences in advance and using them for filtering, users are able to receive in their portable devices only relevant news items.

When initially contacting a mesh router, clients are expected to send a filter to be used to identify the items of interest to the user. As long as the client maintains a connection to the router, it will receive updates whenever new items that match the users interests are received. Filtering is done at the router to avoid excessive communication with the client devices, which may have limited power supplies. Filters are not propagated through the network.

2.2.1. Assumptions

We assume the presence of a large collection of mesh routers forming a mesh backbone. These mesh routers are not mobile and, as a whole, provide coverage for an extensive geographical area. As part of the fixed infrastructure, they do not have strict constraints on power consumption. We expect these routers to have a dedicated amount of memory space to be used for storing news items. These caches will be updated periodically using the gossip protocol described in Section 2.3.

News items are propagated through the network in the form of news entries. While a news item is a piece of information, a news entry is the representation of the news item in the network and for each news item several news entries may exist. The dissemination of news entries is done primarily within the mesh backbone. Each router can communicate wirelessly with the routers within its range. These routers are called its neighbors. A unique id is associated with each router.
The entries that a router inserts into the network can be uniquely identified by a combination of the router id and a sequence number. In its most basic form, a news entry contains a unique id, a timestamp and a time-to-live. There may be other fields of information depending on the application. A limited number of these entries can be stored by each router in its local cache. In our experiments, the size of the cache is defined by the parameter $c$, which is the same for all routers. The storage capacity of the network as a whole is then $N \times c$, where $N$ is the number of routers in the network. Routers in the network gossip periodically, exchanging the entries in their caches. We define a round as a gossiping interval in which each router initiates an exchange once.

The clients in our system are, for the most part, portable devices, such as phones, laptops or PDAs. These devices have limited power supplies and, for that reason, do not participate actively in the dissemination of news items. They do, however, engage in communication with the routers to be updated on news events.

### 2.2.2. Application Description

To illustrate the usefulness of the service, we propose a possible application scenario: advertising in a shopping center where products on sale need to be promoted. In this scenario, routers could be located at any other shop. Some routers may already be in place for use as hotspots or as part of a store’s accounting system. As computers have become prevalent in business environments, we do not expect lack of infrastructure to be a major obstacle for the deployment of the mesh network. With the mesh network in place, news items advertising products would be disseminated through the mesh network and be picked up by the mobile devices that customers carry.

News entries have a limited lifetime. After this time period expires, the information they carry is no longer valuable to clients and should be flushed from the network. Going back to our example, the lifetime of entries could relate to the time period when a sale is effective (for example, drink at a discount price during lunch time).

At any point in time, a router will have a partial view of the complete set of news items in its cache. We do not expect each router to store all items. Instead, each router will devote a fixed amount of memory to store entries it discovers through communication with other routers. Periodically, this view will be refreshed with different news entries. According to the interests that customers have expressed when contacting a router, their mobile clients will be updated with relevant advertisements.
2.3. SHUFFLE PROTOCOL

When a router participates in a gossip exchange, it assumes either an active or a passive role. Each router initiates an exchange once per round. We refer to the router that initiates the exchange as the active one, while the one that is contacted assumes the passive role.

The data exchange between routers follows a predefined structure. Figure 2.1 shows the skeleton of the push-pull epidemic protocol we use for communication within the mesh backbone. Three methods represent the core of the protocol: selectPeer(), selectItemsToSend() and selectItemsToKeep(). By implementing different policies in these methods, various epidemic protocols, each with its own distinctive characteristics, can be instantiated.

Based on the structure shown in Figure 2.1, we introduce an epidemic protocol we call *shuffle*. The shuffle protocol is characterized by avoiding the loss of data during an exchange. It achieves this by establishing an agreement between peers that each peer will keep the entries received from the other after the exchange takes place. We will elaborate on the details of the exchange later on.

The shuffle protocol is partly based on a peer-to-peer protocol used for handling flash crowds [Stavrou et al. 2004], which we recently enhanced in order to maintain unstructured overlays that share important properties with random graphs [Voulgaris et al. 2005]. The most important observation to make is that any two nodes that engage in a shuffle essentially swap a number of entries. In doing so, they not only preserve the data that are collectively stored in the network, but also “move” these data around in a seemingly random fashion. The underlying idea is that by randomly shuffling data entries between nodes, all nodes will be able to see all news items eventually.

![Skeleton of an epidemic protocol](image)
2.3.1. Protocol Policies

In the shuffle protocol, each node agrees to keep the entries received from a neighbor for the next round. This might seem trivial, but given the limited storage space available in each node, keeping the entries received during an exchange implies discarding some entries that the node has in its cache. By picking the entries to be discarded from the ones that have been sent to the neighbor, we ensure the conservation of data in the network. The policies are summarized as follows:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectPeer()</td>
<td>Select a neighbor randomly</td>
</tr>
<tr>
<td>selectItemsToSend()</td>
<td>Randomly select ( s ) entries from the local cache. Send a copy of those entries to the selected peer.</td>
</tr>
<tr>
<td>selectItemsToKeep()</td>
<td>Add received entries to the local cache. Remove repeated items. If the number of entries exceeds ( c ), remove entries among the ones that were previously sent until the cache contains ( c ) entries.</td>
</tr>
</tbody>
</table>

2.3.2. Simulation Setup

In order to observe the behavior of the protocol in large-scale settings, a series of simulations was conducted. We have learned from earlier studies of other epidemic protocols [Voulgaris and van Steen 2003] that the results from emulations running in a cluster of hundreds of nodes yield strikingly similar results to simulation results when observing large-scale behavior. While these results were obtained for a wired environment, we have also conducted practical experiments with deployments of up to 100 gossiping nodes [Mandemaker 2008]. We observed that the dissemination of a data item through the network shows similar characteristics to the dissemination observed in simulations, which we view as an encouraging sign that simulation is a useful tool to capture the major properties of a gossip protocol.

The simulations in this chapter use the unit disc graph model, where nodes can only communicate with their neighbors located within a fixed communication range. Even though our simulations use this simple model (and unrealistic) model of the physical layer, they allow us to gain some insight into the behavior of the protocol by exploring a variety of scenarios with a large number of nodes. More realistic simulations\(^1\) would restrict our ability to explore the design space with

---

\(^1\)TOSSIM (the TinyOS simulator) is an example of a simulator that provides a more realistic MAC layer and radio model. We use TOSSIM in Chapter 6 for networks of up to 900 nodes.
very large networks. For this reasons, we decided to study the behavior of the protocol presented in this chapter through extensive high-level simulations of very large networks.

The results presented in this section correspond to a network of 10000 nodes with a cache size of $c$, which may vary in different experiments. Two types of topologies were used in the experiments:

- **Grid topology** The nodes were set up in a square grid topology, with 100 nodes on each side over an area of $100 \times 100$ units. Two cases were explored: (a) the range of each node was set to 1 unit, making communication possible with the node’s immediate neighbors to the North, South, East and West. On average, each node had 3.96 neighbors (due to the effect of boundary nodes with less than 4 neighbors); (b) the range of each node was set to 2 units, making communication with 12 immediate neighbors.

- **Random topology** The nodes were placed randomly in a square area of $100 \times 100$ units. Nodes were allowed to reach neighbors within a range of 2 units, which was enough to guarantee that each node had at least one neighbor and that a path between any two nodes existed. The average number of neighbors for each node was 12.19.

Both topologies were used to study the behavior of the protocols. The experiments that we conducted focused on two characteristics observed during the execution of each epidemic protocol (a) the replication of items in the network and (b) the time required to reach all the nodes in the network.

### 2.3.3. Properties

To understand the behavior of the protocol, we focus on the way a single news entry traverses the network. At first instance, a news entry is inserted into the network by a router. Subsequently, the entry takes a step (moves to the cache of another router) whenever the router that currently holds the entry participates in an exchange. For every execution of the protocol, the next step of the entry is chosen randomly. As a consequence, the path followed by an individual entry consists of a series of random steps. This behavior is analogous to a random walk in the space defined by the mesh network.

Additionally, as an entry moves from router to router, there is a chance that it will be replicated in the caches of the routers it has passed through, given that there was space available. It follows that a news item may have several news entries in the network at the same time. For that reason, when referring to an item in the network we are actually referring to all news entries that represent that news item. These entries have the same id. In the next sections we study the way these entries are replicated through the network.
Distribution of Storage Capacity

Let us first consider how different news items are distributed through the network. After running the protocol for several rounds, we observe that the storage capacity of the network is evenly divided between the different items. By this, we mean that the slots available to store news entries are used in a balanced way, with each news item being able to place approximately the same number of entries in the network. This behavior is not programmed into the algorithm, but it is an emerging property resulting from its repeated execution.

The value to which the number of entries of an item converges is dictated by the number of different news items in the network. Given a network of size $N$ where all nodes have a cache size of $c$, the network has a total capacity of $N \times c$. These $N \times c$ available slots have to be filled with $d$ different news items. Because of the randomness introduced when choosing which entries to exchange, the total capacity should eventually be evenly divided between the different items resulting in an average of $\frac{N \times c}{d}$ entries for each of the $d$ news items. Considering that the protocol does not allow more than one news entry representing the same news item in the same cache, this means that $c/d$ of the nodes should have an item of each of the $d$ different ids:

$$
\# \text{ entries per item} = \frac{\text{capacity of the network}}{\text{number of news items}} = \frac{N \times c}{d}
$$

Figure 2.2 shows the convergent behavior of the protocol. For the experiment, a collection of 10000 nodes were placed in a grid topology with 4 neighbors per node and 10 nodes were randomly selected to generate different news items. Time is measured in rounds, where a round is a gossiping interval in which each node executes the exchange protocol once. After an initial stabilization period, the number of entries in the system for each of the 10 items converges to the same value. According to our previous reasoning, this value should be $10000 \times 5/10 = 5000$, meaning that 50% of the nodes in the network have an entry from one of the 10 different news items available, which is confirmed by our experiments. Similar convergent behavior was observed when experimenting with other topologies.

Dissemination Speed as a Function of the Diversity of News Items

To demonstrate the effectiveness of shuffling for disseminating information, we have conducted experiments that show the effect of the number of different items on the dissemination speed of the items through the network. In this section, we look at the time needed for the news items to have reached all routers in the network. The results presented here correspond to a mesh backbone of 10000 routers. Unless explicitly stated, the routers were set up in a rectangular grid.
Figure 2.2: Convergent behavior illustrated by having 10 nodes that generate news items in a network of 10000 nodes (arranged as a 100 × 100 grid).

Figure 2.3: Number of rounds required for all the routers in the backbone to have seen an item using shuffling on a grid topology.
Figure 2.3 shows the time, measured in rounds, required for various different items to have passed through the caches of all the routers in the backbone. The cache size for all routers was set to 50 and all items in the cache were shuffled in each round. In each experiment, a different number of distinct items (starting at 50 and up to 600, with increments of 50) were inserted into the backbone by routers located in random locations. For each news item, the time required to traverse the backbone was measured. Due to randomness in the exchanges and the location of the routers inserting the news items, the time measured for an individual news item may vary significantly compared to the measurements for other items. By calculating the average time for a news item to go through the mesh backbone, we can observe that as the diversity of news items in the network increases the average time for a specific news item to reach all routers increases linearly. We observe that this linear behavior is maintained when conducting the experiments with different topologies, as shown in Figure 2.4.

In our third set of experiments, we focus on the effect of the cache size on the dissemination speed. As before, we look at the average time required for an item to have reached all routers in relation to the number of different items being gossiped. The results, shown in Figure 2.5, reveal that the slope of the curve of average values is directly related to the number of items being shuffled. There is an inversely proportional relationship between the number of items being exchanged and the slope of the curve. The four curves shown correspond to experiments with
a cache size of 30, 40, 50 and 60 items. In all cases, all entries in the cache were exchanged. By doubling the number of entries shuffled from 30 to 60, the average time for news items to pass through every router in the backbone is virtually divided in half. Such a characteristic, as well as the predictable behavior with an increasing number of different items, are important factors to consider when choosing an appropriate value for the cache size $c$ and the number of entries to shuffle.

**Robustness**

In order to test the robustness of our system in the case of node failures, we look at a scenario where the nodes within a limited area go down, not unlike what would happen in case of a power outage. The experiment, performed with 10000 routers arranged in a grid with range 1, consists of observing the number of entries per
Figure 2.6: Number of entries per news item. 49% of the routers go down at round 200 and recover at round 300.

Figure 2.7: Average number of entries per item. Routers go down at round 200 and recover at round 300.
item in the mesh backbone before, during and after the failure of all routers within a square area. We assume that when a router fails, all the entries in its cache are lost. When the router goes up again, its cache is empty and has to be populated again.

Figure 2.6 shows the results of the experiment when 49% of the routers experience a failure at the same time and recover 100 rounds later. The routers chosen for failure were arranged in a $70 \times 70$ square inside the $100 \times 100$ grid. Like the experiment in Figure 2.2, 10 items are being shuffled in the network and all routers have a cache size of 5. Once the numbers of entries per news item have converged to the same value, the routers chosen for failure go down. As could be expected, given that the entries were randomly located throughout the network, the number of entries per item is virtually cut by half once the failures occur. When the routers that failed rejoin the network, we observe a smooth transition to the previous state, with entries quickly populating their caches. Unlike the first rounds, the entries per item are replicated at roughly the same rate. This is due to the fact that the routers surrounding the area of failure already have their caches full of entries and can update the routers that failed as soon as they become operational again.

For a clear view of the recovery time, Figure 2.7 shows the average number of entries per item. In addition to the experiment presented earlier, we include the case where 25% of the routers fail. These routers are arranged in a $50 \times 50$ square. Comparing both curves, we observe the same behavior up to the moment of node failures. At that point, the average number of entries per items falls according to the loss of storage space. The recovery in both cases is quick despite the difference in the number of routers that failed.

The speedy recovery of the affected area can be attributed to information flowing in from multiple sides. For an affected square area, we would expect the recovery time to be proportional to the square root of $n$, where $n$ is the number of routers that experience a failure. As can be seen in Figure 2.8, this seems to be the case. The results shown in the figure were obtained using the random topology with range 2. Several experiments where routers within a square area failed were conducted. For each experiment, a square area of a different size was used, ranging from $20 \times 20$ up to $80 \times 80$ with increments of 5 distance units on each side. Figure 2.8 shows that, indeed, the recovery times obtained from the simulations tend to be proportional to the square root of the number of routers affected. To verify this, we also plot the curve $a*\sqrt{n}+b$ which was obtained through linear regression. The constants have values 0.504244, and 5.18383, respectively.
2.4. PERFORMANCE ON THE CLIENT SIDE

In this section, we take a look at the performance of the news service from the user’s perspective. Users of the service have access to news items through clients in the mesh network. These include portable devices such as laptops, PDAs or other hand-held devices. Due to the variety of news items available in the network, users need to configure their clients to retrieve items matching the users’ interests. Item retrieval is based on content filtering. Once a connection to a router is established, clients must submit filtering criteria for the router to identify which news items to forward to the client.

2.4.1. Discovery Rate

To evaluate the effectiveness of the news service from the user’s perspective, we observe the discovery rate of items over time. The discovery rate is defined as the number of items of interest to a user that a router delivers to the client device over a period of time versus the total number of relevant items. We test the discovery rate through the following experiment:

- A network of 2500 routers is arranged in a 50 × 50 grid. Each router can communicate with its neighbors to the North, South, East and West.
- 50 users are positioned at random locations.
- 500 news items are being shuffled in the network.

Figure 2.8: Recovery times when an increasing number of routers fail.

\[
f(x) = a \cdot \sqrt{x} + b
\]
• The interests of the users match 100 news items.

The news items are published at random locations in the network and shuffled until the number of entries for each item converges to roughly the same value (as seen in Section 2.3.3). At that point, the clients connect to the nearest router expressing interest in certain kinds of items. A router responds by forwarding the matching news items seen in its cache to the client. Caches are updated with every gossip round prompting the delivery of previously undiscovered items to the client. As a result, we expect the discovery rate to increase as the client spends more time connected to a router.

The results of repeating the experiment with different cache sizes can be seen in Figure 2.9. The figure shows the average discovery rates for the 50 users. In all cases, the discovery rates increase rapidly during the initial rounds and slow down when most items have already been discovered. As could be expected, larger caches lead to higher discovery rates of items. This is due to a higher storage capacity in the network that allows for more entries to be placed for each news item. Therefore, the probability of finding a particular item in the cache of a router increases.

It should be noted that an increase in the total number of news items would slow down the discovery rate, as dissemination speed decreases with the number of news items in the network. This effect can be countered by an increase in cache size. In the remainder of this section, we take a fixed number (500) of news items and explore the effect of modifying other parameters, such as the cache size, the number of items shuffled and the topology of the network, on the discovery rate.

2.4.2. Probability of seeing an item

Executing the shuffle protocol until the storage capacity of the network is full yields a probability of $c/d$ of finding a particular item when examining the cache of a router picked at random, for $c \leq d$. If we define the success of our experiment as finding a particular item in a random cache and knowing that the probability of success remains constant, the probability of succeeding after performing the experiment $k \geq 1$ times is:

$$p(k,c,d) = 1 - \prod_{i=1}^{k} (1 - \text{prob\_success}(i)) = 1 - \left(1 - \frac{c}{d}\right)^k$$

Figure 2.10 shows the probability of finding an item in a cache selected at random after $k$ attempts for different cache sizes. We observe a similar, although not identical, behavior to the discovery rate results presented in the previous section. This is not surprising, as the shuffle protocol ensures that after each round a router refills its cache with entries received from a neighbor chosen at random. However,
due to the locality of the gossip exchanges, when looking at the cache of the same router for several rounds, we are bound to discover the items held by our neighbors first. This limits the variety of items we might see as our neighbors are more likely to hold many of the same items as we do in comparison to a randomly chosen router in the network. This accounts for the slightly lower discovery rate in the experimental results compared to the probability of seeing an item when selecting a random cache every time.

### 2.4.3. Improving Discovery Rate

Shuffling provides a random sample of the collection of items in the network at every round for each router. However, as can be inferred from Figure 2.9 and 2.10, there is a correlation in the items seen from one round to the next, which accounts for less than optimal discovery rates. In other words, the reason for the discovery rate results not being identical to the probability in Figure 2.10 is due to the results from each round not being independent. This can be attributed to the lack of variety of items in the neighborhood of a router.

Figure 2.11 shows the effect of the neighborhood in the discovery rate. We confirm that the discovery rate was being hampered by each router having a limited neighborhood by showing that the probability of seeing an item when a cache is picked randomly is the same as the discovery rate when the range of a router is such that it can reach any other router in the network. In the graph, the results for a network of routers with range 100 and $p(k, 50, 500)$ overlap. We also show the impact in performance of doubling the range from 1 to 2 units, effectively in-
creasing the number of neighbors from 4 to 12. This experiment shows that it is not necessary to be able to reach every node in the network to achieve a close-to-optimal discovery rate. Finally, as a worse case scenario, we show what happens if the routers are arranged in a single line, where each router can only reach its neighbors to the left and right. In this case, the discovery of items after the first few rounds becomes increasingly slow. We attribute this to new items being hard to come by after the items of interest in the immediate neighborhood have been discovered. Having only two neighbors, the likelihood of new items reaching the neighborhood is reduced, requiring more iterations of the protocol to update a cache with different items. Obviously, this topology is not realistic and should be avoided.

Another option for improving the discovery rate without increasing the number of entries exchanged per round is to increase the cache size. The results presented previously assumed that all entries in the cache were exchanged in each round. Figure 2.12 shows how increasing the cache size while exchanging the same number of entries provides an initial boost in the discovery rate. However, since the speed at which new items move through the network depends on the number of entries exchanged per round, having a bigger cache does not provide any benefits for finding the last few items that were not originally in the vicinity of the router.
Figure 2.11: Discovery rate of news items.

Figure 2.12: Discovery rate of news items.
2.5. RELATED WORK

From a functional point of view, the news service we propose has some similarity with content-based distributed publish/subscribe systems [Carzaniga et al. 2001; Cugola et al. 2001; Pietzuch and Bacon 2002; Segall and Arnold 1997]. With the increase in popularity of wireless technology, some publish/subscribe systems have been extended to support mobile, wireless clients [Caporuscio et al. 2003]. For the most part, the systems proposed use a single tree-shaped overlay to interconnect a set of brokers which cooperate to deliver the events published to the appropriate subscribers. This approach, while efficient under static conditions, might face robustness and scalability issues in highly dynamic environments, such as wireless networks with mobile users. Efforts in maintaining a tree overlay under frequent topology changes aim at dealing with these issues [Picco et al. 2003]. However, depending on how frequently the changes occur, maintaining a tree may introduce additional overhead and complexity. Our approach offers robustness and scalability at the cost of periodic communication for gossiping.

Content-based publish/subscribe projects aimed explicitly at wireless, mobile environments [Cugola et al. 2005; Huang and Garcia-Molina 2003; Meier and Cahill 2002], especially systems that rely on probabilistic techniques for the delivery of events [Costa and Picco 2005], are more closely related to our work. Like our news service, these publish/subscribe systems rely on distributed algorithms to build their trees and deliver messages to subscribers. When designing a publish/subscribe system for wireless environments, unlike the wired case, the cost of communication between brokers varies depending on the wireless connectivity. Consequently, the creation of publish/subscribe trees is largely influenced by the wireless connectivity, driving an interest in building trees that can deliver the messages to subscribers at a lower cost [Huang and Garcia-Molina 2003]. Costa and Picco 2005 recognizes the fragility of a using a tree in a wireless environment and proposes a combination of deterministic routing with probabilistic techniques to increase resilience when faced with topology changes. In our work, while similar in spirit to publish/subscribe systems, we deal with dynamicity (due to mobility and node/link failures) introduced by the wireless environment through a purely probabilistic approach. As a result, our news service provides robustness and resilience to failures without increasing complexity.

Our work also relates, in a way, to efforts in distributed storage [Haeberlen et al. 2005; Adya et al. 2002]. Like our news service, these systems rely on data redundancy to ensure robustness when node failures occur. However, while most of these systems carefully place replicas based on the reliability of nodes, we replicate items and relocate them in a random fashion. We do, nevertheless, manage to use the storage capacity in a fair manner, dynamically adjusting the number of
replicas of an item according to the number of items in the network.

2.6. DISCUSSION

As mentioned in the introduction, one of the main advantages of having a decentralized system like the one we propose is avoiding single ownership of the service. Single ownership implies that one entity is fully responsible for the availability and quality of the service. This may not be a bad thing from the point of view of managing the system, however it restricts others from contributing to the service even when resources are available. One of the strengths of our news service is the flexibility it allows for routers to have some control over the quality of service they offer. As explained in Section 2.4, the quality of the service as perceived by the users can be improved by increasing the wireless range or the amount of memory allocated for storing entries. Decisions to do so can then be taken on an individual basis by the administrators of each router.

Taking a more active approach for the discovery of items is also a possible way of improving the perceived performance from the users point of view. As explained, clients connect to a nearby router and retrieve news items matching the user’s interests. While the matching news items from the router’s cache will be made available to the client immediately, discovering the totality of relevant news items may take several rounds and, depending on the time period between rounds, this delay might inconvenience some users. Instead of passively waiting for the news items to arrive, a router may decide to forward the user’s filter to other routers, thus increasing the chances of discovering relevant news items. For example, we can calculate that for \( c/d = 0.2 \) it would take at least 10 rounds to retrieve approximately 90\% of all items. In this case, by forwarding the filter to 4 other routers, the client could receive almost all news items in 2 rounds.

The flexibility of being able to independently decide on the amount of resources to invest in the news service coupled with the minimal requirements to participate opens up the possibility of deploying the service on a large scale using heterogeneous nodes. Deploying the service over large geographical areas, such as a campus or a city, may require some considerations in the dissemination of news items. As mentioned before, we impose a time limit for the validity of the entries in the network. However, when disseminating the entries over a large area, it may also be necessary to establish geographic constraints. By adding location-awareness to the shuffling of entries, news items could be dispersed over limited areas. For example, an entry may be forwarded only within a radius from the location where it originated. As a result of restricting the area over which an entry can travel, space which would otherwise be taken by these entries is freed, increasing
the number of different items that the network can hold.

Another consideration to keep in mind is security. A gossip-based system like the one we propose is specially susceptible to denial-of-service attacks. We can imagine a scenario where a malicious node generates bogus news items and inserts them into the network at a high rate. Having large numbers of items at the same time slows down the dissemination speed, as there are less entries per item in the network. Without any security mechanisms in place, a single node could virtually bring the service to a halt. It is clear that some kind of regulation regarding who can publish news items is necessary. We return to this issue extensively in subsequent chapters.

2.7. CONCLUSIONS

In this chapter, we presented a highly robust distributed news service suitable for wireless mesh networks. We have shown that by using an epidemic protocol at the core of our service, we can provide an efficient and scalable solution for delivering news items while, at the same time, offering the participating routers the flexibility of managing their own resources to better suit their clients needs. Through the use of simulations, we analyzed the effectiveness and robustness of the dissemination of items through the mesh backbone. We corroborated the effectiveness of the service by taking the user’s perspective and providing an analysis of the quality of the service in terms of the delivery of relevant news items to the client devices.

In the three next chapters, we take the shuffle protocol as a test case to explore two subjects related to gossip protocols: a) the study and modelling of local interactions as a way to understand large-scale behavior (Chapter 3) and b) the impact of malicious behavior in the properties of dissemination in a network of gossiping nodes (Chapters 4 and 5). Afterwards, in Chapter 6 we move away from the point-to-point model of communication used by the shuffle protocol and focus on recreating similar characteristics of dissemination with a broadcast-based protocol.
CHAPTER 3

Modelling the Shuffle Protocol

Through an extensive simulation study, we have gained some insight into the basic properties of the shuffle protocol. While executing large-scale simulations has shed some light into the emergent behavior of the protocol, the relationships between the performance of the protocol and the settings of the parameters are only understood at a high level. The mechanics of the gossip exchanges have yet to be fully analyzed and understood.

In this chapter, we develop an analytical model of information dissemination for the shuffle protocol. With this model we analyse how fast an item is replicated through a network, and how fast the item covers the network. We also determine the optimal size of the exchange buffer, to obtain fast replication. Our results are confirmed by large-scale simulation experiments.

3.1. INTRODUCTION

Today, large-scale distributed systems consisting of thousands of nodes are commonplace, due to the wide availability of high-performance and low-cost devices. In practice, these systems are often diagnosed through performing simulations to discover correlations between design parameters and observed behavior. Such experimental results provide essential data on system behavior, and can aid in understanding the emergent behavior of the system. However, experiments can be time consuming and the infinite space of the parameter settings for probabilistic systems is often too large to be explored experimentally. Consequently, the experiments do not always clarify how parameter settings influence the extra-functional properties of the system. As a result, it is very difficult to predict what the effects of certain design decisions are, as it is practically infeasible to explore the full range of input data. A challenge is to develop analytical models that capture (part
of the behavior of a system, and then subsequently optimise design parameters following an analytical rather than an experimental approach.

We are interested in developing and validating analytical models for gossip-based systems (cf. [Bakhshi et al. 2007]). These systems rely on epidemic techniques for the communication and exchange of information. These communication protocols, while having simple specifications, show complex and often unexpected behavior when executed on a large scale (e.g., [Jelasity et al. 2007; Drost et al. 2007]). Our analytical models of gossip protocols need to be realistic, yet, sufficiently abstract to allow for easy prediction of systems behavior. By ‘realistic’ we mean that they can be applied to large-scale systems and can capture functional and extra-functional behavior such as replication, coverage, convergence, and other system dynamics (see [Eugster et al. 2004]). Such models are amenable for mathematical analysis, to make precise predictions. Furthermore, we will exploit the fact that because an analytical model presents an abstraction of the original protocol, a simulation of the model tends to be much more efficient (in computation time and memory consumption) than a simulation of an implementation of this protocol.

In this chapter, we develop an analytical model the epidemic protocol from the previous chapter. To summarize briefly, nodes executing the protocol periodically contact each other and exchange data items. Concisely, a node initiates a contact with a random neighbor, pulls a random subset of items from the contacted node, simultaneously pushing its own random subset of items. This push/pull approach has a better performance than a pure push or pull approach [Jelasity et al. 2007; Karp et al. 2000]. The amount of information exchanged during each contact between two communicating nodes is limited. Replication ensures the availability of the data items even in the face of dynamic behavior. Thus, nodes not only conserve the data collectively stored in the network, but also relocate it in a random fashion; hence, nodes will eventually see all data items.

The central point of our study is a thorough probabilistic analysis of information dissemination in a large-scale network using the aforementioned protocol. Our modelling framework for a gossip protocol differs from others in that we do not follow the traditional modelling using the mathematical theory of epidemics [Eugster et al. 2004]. Instead, the behavior of the protocol is modelled at an abstract level as pairwise node interactions. When two neighboring nodes interact with each other (gossip), they may undergo a state transition (exchange items) with a certain probability. The transition probabilities depend on the probability that a given item in a node’s local storage has been replaced by another item after the exchange. We calculated accurate values for these probabilities, yielding a rather complicated expression. This expression depends not only on the amount of items, message size and local storage size, but also on the amount of items
both gossiping nodes have in common, in particular, how many of such items the contacted node receives during the exchange. The expression is complex because it incorporates the expected value of the amount of such items, and a connection between the parameters is not obvious. We also determined a close approximation that is expressed by a much simpler formula, as well as a correction factor for this approximation allowing for precise error estimations. Thus we obtain a better understanding of the emergent behavior of the protocol and how parameter settings influence its extra-functional behavior.

We investigated two properties characterizing the protocol, namely, the number of nodes that have ‘seen’ a given item over time (coverage), and the number of replicas of this item in the network at a certain moment in time (replication). Using the values of the transition probabilities, we determined the optimal number of items to exchange per gossip, for a fast convergence of coverage and replication. Moreover, we determined formulas that capture the dissemination of an item in a fully connected network. All our modelling and analysis results are confirmed by large-scale simulations, in which simulations based on our analytical models are compared with running the actual protocol. To the best of our knowledge, we are the first to develop an accurate, realistic formal model that can be used to optimally design and fine-tune a given gossip protocol. In this sense, our main contribution is demonstrating the feasibility of a model-driven approach to developing real-world gossip protocols.

Related work

Two areas of research are most relevant to the work described in this chapter: rigorous analysis of gossip (and related) protocols, and results from mathematical theory of epidemics [Bailey 1975; Daley and Gani 1999]. The results from epidemics are often used in the analysis of gossip protocols [Eugster et al. 2004] (e.g., the traditional gossiping paper by Demers and colleagues [Demers et al. 1987]). We restrict our overview to the most relevant publications from the area of (analysis of) gossip protocols.

Several works have focused on gossip-based membership management protocols. Allavena and colleagues [Allavena et al. 2005] proposed a gossip-based membership management protocol and analysed the evolution of the number of links between two nodes executing the protocol. The states of the associated Markov chain are the number of links between pairs of nodes. From the designed Markov chain they calculated the expected time until a network partition occurs. This case study also includes a model of the system under churn. A goal of that paper is to show the effect of mixing both pull and push approaches.

Eugster and colleagues [Eugster et al. 2003] presented a lightweight probabilistic broadcast algorithm, and analysed the evolution of processes that gos-
sip one message. The states of the associated Markov chain are the number of processes that propagate one gossip message. From the designed Markov chain, the authors computed the distribution of the gossiping nodes. Their analysis has shown that the expected number of rounds to propagate the message to the entire system does not depend on the out-degree of nodes. These results are based on the analysis assumption that the individual out-degrees are uniform. However, this simplification has shown to be valid only for small systems (cf. [Jelasity et al. 2007]).

Bonnet [Bonnet 2006] studied the evolution of the in-degree distribution of nodes executing the Cyclon protocol [Voulgaris et al. 2005]. The states of the associated Markov chain are the fraction of nodes with a specific in-degree distribution. From the designed Markov chain the author determined the distribution to which the protocol converges.

There are a number of theoretical results on gossip protocols, targeted to a distributed aggregation. In these protocols, a set of data is distributed over the nodes of a network and the nodes compute an aggregate of the data set. Kempe and colleagues [Kempe et al. 2003] proposed a push-only gossip-based aggregation protocol for the fully connected network. In this paper, the authors used Gaussian mixture modelling [Dempster et al. 1977; McLachlan and Peel 2000]. A performance of the protocol has been measured by how quickly a data originating with a node diffuses through a network (for uniform gossip). Each node locally maintains an aggregation vector $v_{i,t}$. A state of the associated Markov chain is the fraction of the vector node $i$ sends to other node. From the designed Markov chain, the authors studied the convergence rate. In addition, the authors showed that the diffusion speed for flooding corresponds to the mixing time of a random walk on the network. Validation of the theoretical results with practical experiments is left as a future work.

The protocol [Kempe et al. 2003] has been further tailored by Boyd and colleagues [Boyd et al. 2005] to work on an arbitrarily connected network. In their analysis, the Markov chain is defined by a weighted random walk on the graph. Every time step, a pair of nodes (connected by an edge) communicates with a transition probability, and sets their values equal to the average of their current values. A state of the associated Markov chain is a vector of values at the end of the time step. The authors considered the optimisation of the neighbor selection probabilities for each node, to find the fastest-mixing Markov chain (for fast convergence of the algorithm) on the graph.

Jelasity and colleagues [Jelasity et al. 2005] proposed a push-pull solution for aggregation in large dynamic networks, supported by a performance analysis of the protocol. A state of the system is represented by a vector, the elements of which correspond to the values at the nodes, a target value of the protocol...
calculated from the vector elements, and a measure of homogeneity characterising the quality of local approximations. The vector evolves at every step of the system according to some distribution. In the analysis, the authors considered different strategies (e.g., neighbor selection) to optimise the protocol implementation, and calculated the expected values for the above mentioned protocol parameters.

Deb and colleagues [Deb et al. 2006] studied the adaptation of random network coding to gossip protocols. The authors analysed the expected time and message complexity of two gossip protocols for message transmission with pure push and pure pull communication models.

3.2. A GOSSIP-BASED PROTOCOL FOR DATA DISSEMINATION

This section summarizes the main aspects of the shuffle protocol (introduced in the previous chapter), which we aim to model. The protocol itself is, at heart, a simple push-pull gossip protocol which can be used in wired or wireless networks. Its purpose is to disseminates data items of general interest to a collection of nodes. The protocol relies on replication to ensure the availability of data items in the face of dynamic behavior. We briefly summarize the protocol and explain the system model.

The system consists of a collection of nodes, each of which contributes a limited amount of storage space (which we will refer to as the node’s cache) to store data items. The nodes periodically swap (shuffle) data items from their cache with a randomly chosen neighbor. In this way, nodes update their caches on a regular basis, allowing nodes to gradually discover new items as they are disseminated through the network.

Items can be published by any user of the system, and are propagated through the network. An item is a piece of information, and for each item several copies may exist in the network. As items are gossiped between neighboring nodes, replication may occur when a node has available storage space to keep a copy of an item it just gossiped to a neighbor.

3.2.1. Protocol assumptions

All nodes have a common agreement on the frequency of gossiping. However, there is no agreement on when to gossip.

In terms of storage space, we assume that all nodes dedicate the same amount of storage space to keep items locally, and that all items are of the same size. Therefore, we say that each node has a cache size of $c$. When shuffling, each node sends a fixed number $s$ of the $c$ items in the cache.
The gossip exchange is performed as an atomic procedure, meaning that once a node initiates an exchange with another node, this pair of nodes cannot become involved in another exchange until the current exchange is finished.

### 3.2.2. Description

Nodes executing the shuffle protocol initiate a shuffle periodically. In order to execute the protocol, the initiating node needs to contact a gossiping partner. Such a random peer is delivered by an underlying layer that keeps track of the neighborhood membership. In a wired environment, this service could be provided by, for instance, a peer sampling service [Jelasity et al. 2007] running at each node. For wireless environments, the neighborhood is determined by the radio connectivity between nodes.

We describe the protocol from the point of view of each participating node. Node $A$ initiates the shuffle by executing the following steps:

1. picks a neighboring node $B$ uniformly at random;
2. selects randomly $s$ items from the local cache, and sends a copy of these items to $B$;
3. receives $s$ items from the local cache of $B$;
4. checks whether any of the received items are already in its cache; if so, these received items are eliminated;
5. adds the rest of the received items to the local cache; if the total number of items exceeds cache size $c$, removes items at random among the ones that were sent by $A$ to $B$, but not those that were also received by $A$ from $B$, until the cache contains $c$ items.

In response to being contacted by $A$, node $B$ executes the following steps:

1. receives $s$ items from the local cache of $A$;
2. selects randomly $s$ items from its local cache, and sends a copy of these items to $A$;
3. checks whether any of the received items are already in its cache; if so, these received items are eliminated;
4. adds the rest of the received items to the local cache; if the total number of items exceeds cache size $c$, removes items at random among the ones that were sent by $B$ to $A$, but not those that were also received by $B$ from $A$, until the cache contains $c$ items.
According to the protocol, each node agrees to keep the items received from a neighbor. Given the limited storage space available in each node, keeping the items received during an exchange implies discarding some items that the node has in its cache. By picking the items to be discarded from the ones that have been sent to the neighbor, the conservation of data in the network is ensured.

3.2.3. Properties

We are interested in the characteristics of the dissemination of data items when the protocol is executed at a large scale, i.e., with a large set of nodes. For this reason, we focus on two properties that can be observed in large deployments: i) the number of replicas of an item in the network, and ii) the coverage achieved by an item over time.

Replication

This property is defined as the fraction of nodes that hold a copy of a generic item \( d \) in their cache, at a given moment. After an item is introduced into the network, with every shuffle involving a node that has the item in its cache, there is a chance that a new copy of the item will be created, or that the item will be discarded. As a result, with every passing round the number of copies in the network for a particular item fluctuates. Given that the storage space at the node is limited, items are in constant competition to place copies in the network. Since competition is fair (all items have the same chance of being replicated or discarded), eventually the storage capacity is evenly divided between the existing items. To be more precise, consider a network of \( N \) nodes, in which \( n \) different items have been published in total. Since there are \( N \cdot c \) cache entries in the network in total, the average number of copies that an individual item has in the network will converge to \( \frac{N}{n} \). So the fraction of nodes that have a replica of an item in their cache will converge to \( \frac{c}{n} \) on average.

Coverage

This property is defined as the fraction of nodes in the network that have seen a generic item \( d \) since it was introduced into the network. As explained earlier, several copies of an item are generated after the item is first published. Due to the periodic nature of the protocol, these copies continually move through the network. This results in nodes discovering item \( d \) over several rounds. With each passing round, more nodes will have seen \( d \). Eventually, \( d \) will have been seen by all nodes (i.e., the coverage is equal to 1). The speed at which the coverage grows is influenced by several factors (as will be explained later on) including the
number of different items in the network (i.e., competition), cache size, and the size of the exchange buffer.

### 3.2.4. Experimental observations

Before moving on to the analysis of the protocol, we would like to focus on an important aspect that we have observed during our extensive simulation study of the shuffle protocol: the tendency of items to replicate to even levels. In other words, a newly published item generates replicas in the network over time until its number of replicas reaches a level comparable to the number of replicas of other items. This comes as a consequence of the random selection of items to gossip and to keep in local storage. By applying random selection, no item is favored over the others resulting in a fair division of the storage space. That is, once the system has reached equilibrium, each item in the network will have, on average, the same number of replicas \( \frac{Nc}{n} \).

Figure 3.1 shows a set of experiments where the distribution of replicas for all the items in the system is tracked over several gossip rounds. In all cases, the network consists of \( N = 2500 \) nodes with a cache size of \( c = 100 \) and each node sends \( s = 50 \) items when it gossips. The number of different items that are inserted in the network is \( n = 500 \). Each graph shows three curves corresponding to the following initial scenarios:

1. Nodes are arranged in a \( 50 \times 50 \) grid. The insertion of items to be gossiped occurs simultaneously at round 0. All nodes start with an empty cache, except for 500 nodes randomly chosen nodes. A unique item is placed in the cache of each of these 500 nodes. As a result, at round 0 our network contains 500 different items and each of these items has a single replica.

2. Nodes are arranged in a \( 50 \times 50 \) grid. The insertion of items to be gossiped occurs at randomly chosen times before round 100. At round 100, all 500 items will be present in the network. However, items that were inserted earlier will have more replicas due to having been shuffled for a longer time.

3. Topology with higher density at the center. The insertion of items to be gossiped occurs at randomly chosen times before round 100. In this topology, nodes near the center have more neighbors to gossip with.

As can be seen from the graphs, the different initial conditions result in different replication patterns early on. However, note how as time progresses the replication patterns become more and more similar. By round 350, it is clear that items have on average \( \frac{Nc}{n} = 500 \) replicas, regardless of the initial conditions of the
Figure 3.1: Distribution of replicas over time.
experiment. This convergence to an equilibrium where the storage space \((N \cdot c)\) is evenly divided between the number of items present \(n\) is a result of the repeated execution of the protocol.

In the shuffle protocol, there is no loss of information during the gossip exchange. We assume that the shuffle operation between two nodes is atomic and since nodes swap items, no item can possibly disappear once inserted into the network. For this reason, once the system has reached equilibrium in terms of replication of items and assuming that there is a path between any two nodes in the network, the replicas will continue to be shuffled, moving from one node to another in a random fashion. Under these conditions, it is reasonable to assume that the probability of finding any given item in a node’s cache is the same for all items in the networks. To be more specific, based on our observations we make the assumption that items are uniformly distributed throughout the network and that any item can be found in a node’s cache with probability \(\frac{c}{n}\). This is the starting point for our probabilistic analysis. The observation of uniform distribution is not entirely unexpected. Similar observations about uniform distribution after repeated shuffling have been made regarding the shuffling of decks of cards in [Bayer and Diaconis 1992; Assaf et al. 2008].

3.3. ANALYTICAL MODEL OF INFORMATION DISSEMINATION

In this section, we analyse the dissemination of a generic item \(d\) in a network in which the nodes execute the shuffling protocol.

3.3.1. Probabilities of state transitions

We present a model of the shuffle protocol that captures the presence or absence of a generic item \(d\) after shuffling of two nodes \(A\) and \(B\). There are four possible states of the caches of \(A\) and \(B\) before the shuffle: both hold \(d\), either \(A\)’s or \(B\)’s cache holds \(d\), or neither of the caches holds \(d\).

We use the notation \(P(a_2b_2|a_1b_1)\) for the probability that from state \(a_1b_1\) after a shuffle we get to state \(a_2b_2\), with \(a_i, b_i \in \{0, 1\}\). The indices \(a_1, a_2\) and \(b_1, b_2\) indicate the presence (if equal to 1) or the absence (if equal to 0) of a generic item \(d\) in the cache of an initiator \(A\) and the contacted node \(B\), respectively. For example, \(P(01|10)\) means that node \(A\) had \(d\) before the shuffle, which then moved to the cache of \(B\), afterwards. Due to the symmetry of information exchange between nodes \(A\) and \(B\) in the shuffle protocol, \(P(a_2b_2|a_1b_1) = P(b_2a_2|b_1a_1)\).

Fig. 3.2 depicts all possible outcomes for the caches of gossiping nodes as a state transition diagram. If before the exchange \(A\) and \(B\) do not have \(d\) \((a_1b_1 = 00)\), then clearly after the exchange \(A\) and \(B\) will not have \(d\) \((a_2b_2 = 00)\). Otherwise,
if A or B has d \((a_1 = 1 \lor b_1 = 1)\), the shuffle protocol guarantees that after the exchange A or B or both will have d \((a_2 = 1 \lor b_2 = 1)\). Therefore, the state 00 has a self-transition, and no other outgoing or incoming transitions.

We now determine values for all probabilities \(P(a_2|a_1b_1)\). They are expressed in terms of probabilities \(P_{\text{select}}\) and \(P_{\text{drop}}\). The probability \(P_{\text{select}}\) expresses the chance of an item to be selected by a node from its local cache when engaged in an exchange. The probability \(P_{\text{drop}}\) represents a probability that an item which can be overwritten (meaning it is in the exchange buffer of its node, but not of the other node in the shuffle) is indeed overwritten by an item received by its node in the shuffle. Due to the symmetry of the protocol, these probabilities are the same for both initiating and contacted nodes. In Sec. 3.3.2, we will calculate \(P_{\text{select}}\) and \(P_{\text{drop}}\). We write \(P_{\neg \text{select}}\) for \(1 - P_{\text{select}}\) and \(P_{\neg \text{drop}}\) for \(1 - P_{\text{drop}}\).

**Scenario 1:** \(a_1b_1 = 00\)

Before shuffling, neither node A nor node B have d in their cache.

\(a_2b_2 = 00\): neither node A nor node B have item d after a shuffle because neither of them had it in the caches before the shuffle: \(P(00|00) = 1\)

\(a_2b_2 \in \{01, 10, 11\}\): cannot occur, because none of the nodes have item d.

**Scenario 2:** \(a_1b_1 = 01\)

Before shuffling, a copy of d is only in the cache of node B.
\( a_2b_2 = 01 \): node \( A \) does not have \( d \) because \( B \) had \( d \) but did not select it (to send) and, thus, \( B \) did not overwrite \( d \), i.e., the probability is \( P(01|01) = P_{\text{select}} \).

\( a_2b_2 = 10 \): only node \( A \) has \( d \) because \( B \) selected \( d \) and dropped it; that is, the probability is \( P(10|01) = P_{\text{select}} \cdot P_{\text{drop}} \).

\( a_2b_2 = 11 \): both nodes \( A \) and \( B \) have a copy of \( d \) because \( B \) selected \( d \) and kept it; that is, \( P(11|01) = P_{\text{select}} \cdot P_{\text{drop}} \).

\( a_2b_2 = 00 \): cannot occur as completely discarding \( d \) is not possible in the protocol; that is, if either node sends an item, its partner keeps this copy, and if an item is not among the selected for a shuffle, the item is not replaced by another one (see Sec. 3.2.2).

**Scenario 3:** \( a_1b_1 = 10 \)

Before shuffling, \( d \) is only in the cache of node \( A \). Due to the symmetry of nodes \( A \) and \( B \), this scenario is symmetric to the previous one with \( P(a_2b_2|10) = P(b_2a_2|01) \).

**Scenario 4:** \( a_1b_1 = 11 \)

Before shuffling, \( d \) is in the cache of node \( A \) as well as in the cache of node \( B \).

\( a_2b_2 = 01 \): only node \( B \) has \( d \) because node \( A \) selected \( d \) and dropped it and node \( B \) did not select \( d \); that is, \( P(01|11) = P_{\text{select}} \cdot P_{\text{drop}} \cdot P_{\text{\neg select}} \).

\( a_2b_2 = 10 \): this outcome is symmetric to the previous one: \( P(10|11) = P_{\text{select}} \cdot P_{\text{\neg select}} \cdot P_{\text{drop}} \).

\( a_2b_2 = 11 \): after the shuffle both nodes \( A \) and \( B \) have \( d \), because:

\( \circ \) nodes \( A \) and \( B \) had \( d \) but both did not select it, i.e., \( P_{\text{\neg select}} \cdot P_{\text{\neg select}} \).

\( \circ \) both nodes \( A \) and \( B \) selected \( d \) (thus, both kept it), i.e., \( P_{\text{select}} \cdot P_{\text{select}} \).

\( \circ \) node \( A \) selected \( d \) and kept it and node \( B \) did not select \( d \): \( P_{\text{select}} \cdot P_{\text{\neg drop}} \cdot P_{\text{\neg select}} \).

\( \circ \) symmetric case with the previous one: \( P_{\text{\neg select}} \cdot P_{\text{select}} \cdot P_{\text{\neg drop}} \).

Thus, \( P(11|11) = P_{\text{select}} \cdot P_{\text{\neg select}} + P_{\text{select}} \cdot P_{\text{select}} + 2 \cdot P_{\text{select}} \cdot P_{\text{\neg select}} \cdot P_{\text{\neg drop}} \).

\( a_2b_2 = 00 \): cannot occur, discarding of an item is not permitted by the protocol (see Sec. 3.2.2).
3.3.2. Probabilities of selecting and dropping an item

The following analysis assumes that all node caches are full (that is, the network is already running for a while). Moreover, we assume a uniform distribution of items over the network. This assumption is supported by experiments in subsection 3.2.4 and [Jelasity et al. 2007].

Consider nodes $A$ and $B$ engaged in a shuffle, and let $B$ receive the exchange buffer $S_A$ from $A$. Let $k$ be the number of duplicates (see Fig. 3.3), i.e., the items of an intersection of the node cache $C_B$ and the exchange buffer of its gossiping partner $S_A$ (i.e., $S_A \cap C_B$). Recall from Sec. 3.2.1 that $C_A$ and $C_B$ contain the same number of items for all $A$ and $B$, and likewise for $S_A$ and $S_B$; we use $c$ and $s$ for these values. The total number of different items in the network is denoted as $n$.

The probability of selecting an item $d$ in the cache is the number of selected items (i.e., $s$) divided by the total number of items in the cache (i.e., $c$): $P_{\text{select}} = \frac{s}{c}$. Thus, the probability that an item $d$ in the cache is not selected is: $P_{\neg\text{select}} = 1 - P_{\text{select}} = \frac{c - s}{c}$.

Consider Figs. 3.3 and 3.4. The shuffle protocol demands that all items in $S_A$ are kept in $C_B$ after the shuffle. This implies that: a) all items in $S_A \setminus C_B$ will overwrite items in $S_B \subseteq C_B$, and b) all items in $S_A \cap C_B$ are kept in $C_B$. Thus, the probability that an item from $S_B$ will be overwritten is determined by the probability that an item from $S_A$ is in $C_B$, but not in $S_B$. Namely, the items in $S_B \setminus S_A$ provide a space in the cache for items from $S_A \setminus C_B$. We would like to express the probability $P_{\text{drop}}$ of a selected item $d$ in $S_B \setminus S_A$ (or $S_A \setminus S_B$) to be overwritten by another item in $C_B$ (or $C_A$). Due to symmetry, this probability is the same for $A$ and $B$; therefore, we calculate only the probability that an item in $S_B \setminus S_A$ is dropped from $C_B$. The expected value of this probability depends on how many duplicates
a node receives from its gossiping partner:

\[ E[P_{\text{drop}}] = \begin{cases} \sum_{k=0}^{s} (P^{|S_A \cap C_B| = k} \cdot P^{|S_A \cap C_B| = k}) & \text{if } s + c \leq n \\ \sum_{k=(s+c)-n}^{s} (P^{|S_A \cap C_B| = k} \cdot P^{|S_A \cap C_B| = k}) & \text{otherwise} \end{cases} \]

where \( P^{|S_A \cap C_B| = k} \) is the probability of having exactly \( k \) items in \( S_A \cap C_B \), and \( P^{|S_A \cap C_B| = k} \) is the probability that an item in \( S_B \setminus S_A \) is dropped from \( C_B \) given \( k \) duplicates in \( S_A \cap C_B \). The case distinction is because if \( s + c > n \), then clearly there are at least \( (s + c) - n \) items in \( S_A \cap C_B \).

From the \( \binom{n}{k} \) possible sets \( S_A \), we compute how many have \( k \) items in common with \( C_B \). Firstly, there are \( \binom{s}{k} \) ways to choose \( k \) such items in \( C_B \). Secondly, there are \( \binom{n-s}{s-k} \) ways to choose the remaining \( s - k \) items outside \( C_B \). So in total, \( \binom{s}{k} \cdot \binom{n-s}{s-k} \) possible sets \( S_A \) have \( k \) items in common with \( C_B \). Hence, under the assumption of a uniform distribution of the data items over the caches of the nodes,

\[ P^{|S_A \cap C_B| = k} = \binom{s}{k} \frac{n-s}{n-s-k}. \]

The expected value of \( P^{|S_A \cap C_B| = k} \) is:

\[ E[P^{|S_A \cap C_B| = k}] = \begin{cases} \sum_{\tilde{s}=0}^{\min(c,k)} P^{|S_A \cap S_B| = \tilde{s}} \cdot P^{|S_A \cap S_B| = \tilde{s}} & \text{if } s + k \leq c \\ \sum_{\tilde{s}=(s+k)-c}^{\max(c,k)} P^{|S_A \cap S_B| = \tilde{s}} \cdot P^{|S_A \cap S_B| = \tilde{s}} & \text{otherwise} \end{cases} \]

where \( \tilde{s} \) is the number of items in \( S_A \cap S_B \) (see Fig. 3.4). The case distinction is because if \( s + k > c \) (with \( k \) the number of items in \( S_A \cap C_B \)), then clearly there are at least \( (s + k) - c \) items in \( S_A \cap S_B \).

Among the \( s \) items in \( S_B \), there are \( \tilde{s} \) items also in \( S_A \), and thus only the \( s - \tilde{s} \) items in \( S_B \setminus S_A \) can be dropped from \( C_B \). \( P^{|S_A \cap S_B| = \tilde{s}} \) is the probability that an item in \( S_B \setminus S_A \) is dropped from \( C_B \), given \( \tilde{s} \) items in \( S_A \cap S_B \):

\[ P^{|S_A \cap S_B| = \tilde{s}} = \begin{cases} 0 & \text{if } \tilde{s} = \tilde{s} \\ \frac{\tilde{s}}{s-\tilde{s}} & \text{otherwise} \end{cases} \]

\( P^{|S_A \cap S_B| = \tilde{s}} \) is the probability of having exactly \( \tilde{s} \) items in \( S_A \cap S_B \): \( P^{|S_A \cap S_B| = \tilde{s}} = \binom{s}{\tilde{s}} \frac{(n-s)^{\tilde{s}}}{(n-k)^{\tilde{s}}} \). The intuition behind this expected value of \( P^{|S_A \cap S_B| = \tilde{s}} \) is similar to the

1The other case is presented for the sake of completeness.

2Here we use a generalisation of the usual definition of binomial coefficients to negative integers. That is, for all \( m \) and \( l \geq 0 \), \( \binom{m}{l} = (-1)^l \binom{-m+l-1}{l} \) (cf. [Hilton et al. 1997])

3The other case is presented for the sake of completeness.
one of $P_{|S_A\cap C_B|=k}$. From the $\binom{c}{k}$ possible sets $S_A$, we compute how many have $\hat{s}$ items in common with $S_B$. That is, there are $\binom{s}{\hat{s}}$ ways to choose $\hat{s}$ items in $S_B$, and $\binom{s}{k-\hat{s}}$ ways to choose the remaining $k-\hat{s}$ items outside $S_B$.

Let us assume $s+c \leq n$ and $s+k \leq c$. Then, substituting in the expression for $E[P_{\text{drop}}]$ in case $s+c \leq n$, and noting that in the summand $k=s$ the factor $P_{\text{drop}}^{S_A\cap S_B=\emptyset}$ is equal to zero, we get:

$$E[P_{\text{drop}}] = \sum_{k=0}^{s-1} \binom{c}{k} \frac{(n-c)}{\binom{n}{k}} \sum_{\hat{s}=0}^{k} \frac{s-k}{s-\hat{s}} \binom{s-\hat{s}}{s-k-\hat{s}} \binom{s}{\hat{s}}$$
$$= \frac{n-c}{\binom{n}{s}} \sum_{\hat{s}=0}^{k} \left( \binom{n-c}{s-k-\hat{s}} \right) \sum_{\hat{s}=0}^{k} \binom{s-\hat{s}}{s-k-\hat{s}} \binom{s}{\hat{s}}$$

(3.1)

The probability of keeping an item $d$ in $S_B \setminus S_A \subseteq C_B$ can be expressed as $P_{\text{select}} = 1 - P_{\text{drop}}$.

### 3.3.3. Simplification of $P_{\text{drop}}$

In order to gain a clearer insight into the emergent behavior of the gossiping protocol we make an effort to simplify the formula for the probability $P_{\text{drop}}$ of an item in $S_B \setminus S_A$ to be dropped from $C_B$ after a shuffle. Therefore, we re-examine the relationships between the $k$ duplicates received from a neighbor, the $\hat{s}$ items of the overlap $S_A \cap S_B$, and $P_{\text{drop}}$. Let’s estimate $P_{\text{drop}}^{S_A\cap S_B=\emptyset}$ by considering each item from $S_A$ separately, and calculating the probability that the item is a duplicate (i.e., is also in $C_B$). The probability of an item from $S_A$ to be a duplicate (also present in $C_B$) is $\frac{c}{n}$. In view of the uniform distribution of items over the network, the items in a node’s cache are a random sample from the universe of $n$ data items; so all items in $S_A$ have the same chance to be a duplicate. Thus, the expected number $k$ of items in $S_A \cap C_B$ can be estimated by $s \cdot \frac{c}{n}$. The expected number $\hat{s}$ of items in $S_A \cap S_B$ can be estimated by $k \cdot \frac{c}{n}$, because only the $k$ items in $S_A \cap C_B$ may end up in $S_A \cap C_B$; $\frac{k}{c}$ captures the probability that an item from $C_B$ is also selected to be in $S_B$. It follows that the probability of an item in $S_B \setminus S_A$ to be dropped from $C_B$ after the shuffle is $P_{\text{drop}} = \frac{s-k}{s} = \frac{s-\hat{s}}{s} = \frac{n-s}{n-\hat{s}}$. The complementary probability of keeping an item is $P_{\text{select}} = 1 - \frac{n-s}{n-\hat{s}} = \frac{\hat{s}}{n-\hat{s}}$. These estimates are valid for general $s \leq c \leq n$.

Substituting the expressions for $P_{\text{select}}$ and the simplified $P_{\text{drop}}$ into the formulas for the transition probabilities in Fig. 3.2, we obtain:
MODELLING THE SHUFFLE PROTOCOL

\[
\begin{align*}
\Pr(01|01) &= \Pr(10|10) = \frac{c-s}{c} \\
\Pr(10|01) &= \Pr(01|10) = \frac{c(n-c)}{c(n-s)} \\
\Pr(11|01) &= \Pr(11|10) = \frac{c-s}{c(n-s)}
\end{align*}
\]

In order to verify the accuracy of the proposed simplification for \(E[\Pr_{\text{drop}}]\), we compare the simplification and the accurate formula (3.1) for different values of \(n\). We plot the difference of the accurate \(\Pr_{\text{drop}}\) and the simplification, for cache sizes \(c = 250\) and \(c = 500\) (Fig. 3.5). These figures show that the simplification gives a close approximation of the accurate formula for \(\Pr_{\text{drop}}\) (note the log y-scale).

### 3.3.4. Correction factor

We now investigate how closely the simplified formula of \(\Pr_{\text{drop}}\), that is \(\frac{n-c}{n-s}\) (here referred to as \(S(n, c, s)\)) approximates formula (3.1) (here referred to as \(E(n, c, s)\)).

We compared the difference between these two formulas using a Java package based on common fractions, which provides loss-less calculation [Gilleland 2002]. We observe that the inverse of the difference of the inverse values of both formulas, i.e., \(e_{cs}(n) = (E(n, c, s)^{-1} - S(n, c, s)^{-1})^{-1}\), exhibits a certain pattern for different values of \(n\), \(c\) and \(s\).

For \(s = 1\) and arbitrary values of \(n\) and \(c\), \(E(n, c, 1) = \frac{n-c}{n-1}\), whereas \(S(n, c, 1) = \frac{n-c}{n-1}\). This leads us to investigate the correction factor \(\theta\) as in \(E(n, c, s) = \frac{n-c}{(n-s)+\theta}\).

For \(s = 1\), clearly the factor \(\theta = 1\). Yet, for \(s > 1\) the situation is more complicated. We therefore calculate the first, the second and other (forward) differences over

---

4 A forward difference of discrete function \(f : \mathbb{Z} \to \mathbb{Z}\) is a function \(\Delta f : \mathbb{Z} \to \mathbb{Z}\) with \(\Delta f(n) = f(n+1) - f(n)\) (cf. [Abramowitz and Stegun 1972]).
For $s = 1$, the result of the first difference of the function $e_{c,1}(n)$ is 1. However, for $s = 2$ the first difference is, e.g., if $c = 4$: $e_{4,2}(7) - e_{4,2}(6) = 3.5$, $e_{4,2}(8) - e_{4,2}(7) = 4$, $e_{4,2}(9) - e_{4,2}(8) = 4.5$ and so on. Thus, we observe that the second difference of $e_{c,2}(n)$ is $\frac{1}{2}$. By calculating higher differences for $s > 2$, we conclude that the $s$-th difference of the function $e_{c,s}(n)$ is always $\frac{1}{s}$.

Moreover, at the point $n = 0$ the first, . . . , $s$-th differences of the function $e_{c,s}$ exhibit a pattern similar to the Pascal triangle [Graham et al. 1994]. That is, for $d \geq 1$ the $d$-th difference is $(\Delta^d e_{c,s})(0) = \frac{1}{s \cdot \binom{n}{d}}$ (assuming $\binom{n}{b} = 0$, whenever $b > a$). Knowing the initial difference at point $n = 0$, we were able to use the Newton forward difference equation [Abramowitz and Stegun 1972] to derive the following formula for $n > 0$: $E[P_{\text{drop}}] = \frac{n - c}{(n - s) + \gamma}$, where

$$\gamma = \sum_{d=0}^{s-1} \frac{\binom{n}{d}}{s \cdot \binom{n}{d}} = \frac{\binom{n}{s}}{(n - s) + 1} \cdot \sum_{d=0}^{s-1} \frac{1}{\binom{n-d}{s-1}}. \tag{3.2}$$

In this equation the sum is finite because due to the observation that the $s$-th difference is constant $\frac{1}{s}$, all higher differences are 0.

Thus, the correction factor is $\gamma = \frac{1}{s}$. Extensive experiments with Mathematica and Matlab indicate that $\frac{n - c}{(n - s) + \gamma}$ and formula (3.1) indeed coincide. We can also see from Fig. 3.5 that the correction factor is small.

### 3.3.5. Optimal size for the exchange buffer

We study the optimal value of the exchange buffer $s$ for fast convergence of replication and coverage with respect to an item $d$. Since $d$ is introduced at only one node in the network, one needs to maximize the chance that an item is duplicated. That is, the probabilities $P(11|01)$ and $P(11|10)$ should be maximized (then $P(01|11)$ and $P(10|11)$ are maximized as well, intuitively because for each duplicated item in a shuffle, another item must be dropped).

We give a rigorous argument for this claim in the case of a fully connected network. Let $\alpha$ be a replication of item $d$ at a given moment, and $0 \leq \alpha \leq 1$. In the long run, $\alpha$ converges to $\frac{1}{n}$. Suppose that a shuffle takes place. The chance that after the shuffle there is one more copy of item $d$ in the system is

$$P(11|01) + P(11|10) \cdot \alpha \cdot (1 - \alpha) = 2 \cdot P(11|01) \cdot \alpha \cdot (1 - \alpha)$$

Here $\alpha \cdot (1 - \alpha)$ expresses the chance that exactly one of the nodes in the shuffle contains a copy of the item. Likewise, the chance that after the shuffle a copy of the item has been removed from the system is

$$P(01|11) + P(10|11) \cdot \alpha^2 = 2 \cdot P(01|11) \cdot \alpha^2 = 2 \cdot \frac{n - c}{c} \cdot P(11|01) \cdot \alpha^2$$
MODELLING THE SHUFFLE PROTOCOL  

52  

CHAP. 3

Figure 3.6: Optimal value of exchange buffer size for $c = 100$, depending on $n$.

So, after a shuffle, the change in the number of copies of the item in the system is on average

$$2 \cdot P(11|01) \cdot \alpha \cdot (1 - \alpha) - 2 \cdot \frac{n - c}{c} \cdot P(11|01) \cdot \alpha^2 = 2 \cdot P(11|01) \cdot \alpha \cdot (1 - \frac{n}{c} \cdot \alpha)$$

It follows that as long as $\alpha < \frac{n}{c}$, maximizing $P(11|01)$ maximizes replication of the item.

The probability of replication $P(11|01)$, and its symmetric case $P(11|10)$, are both equal to $\frac{c - n + s}{n}$.

In order to find the value of $s$ that maximizes them, we compute the $s$-derivative of this expression and equal to zero. This yields the equation $s^2 - 2ns + nc = 0$.

Taking into the account that $s \leq n$, the only solution of this equation is $s = n - \sqrt{n(n - c)}$. We conclude that this is the optimal value for $s$ to obtain fast convergence of replication (see Fig. 3.6). This will also be confirmed by the experiments and analyses in the following sections.

3.4. EXPERIMENTAL EVALUATION

In order to test the validity of the analytical model of information spread based on the shuffle protocol presented in the previous section, we follow an experimental approach. We compare properties observed while running the shuffle protocol in a large-scale deployment with simulations of the model under the same conditions. These experiments show that the analytical model indeed captures the characteristics of the information spread due to the shuffle protocol. We note that a simulation of the analytical model is much more efficient (in computation time and memory consumption) than a simulation of the implementation of the shuffle protocol. We discuss this further in Sec. 3.7.
The experiments simulate the case where a new item \( d \) is introduced at one node in a network, in which all caches are full and uniformly populated by \( n = 500 \) items. We use an event-based simulator that takes as input the topology of the network to determine which pairs of nodes can gossip. The experiments are performed on a network of \( N = 2500 \) nodes, arranged in a square grid topology (50\( \times \)50), where each node can communicate only with its four immediate neighbors (to the North, South, East and West). This configuration of nodes is arbitrary, except for the fact that we require a large number of nodes for the observation of emergent behavior. Our aim is to validate the correctness of our analytical model, not to test the endless possibilities of network configurations. The model and the shuffle protocol do not make any assumptions about the network. The network configuration is provided by the simulation environment and can easily be changed into something different, e.g., another network topology. For this reason, we have chosen this large grid for testing, although other configurations could have been possible. In the experiments that follow, after each gossiping round, we measure the total number of occurrences of \( d \) in the network (replication), and how many nodes in total have seen \( d \) (coverage).

**Simulations with the shuffle protocol** Each node in the network has a cache size of \( c = 100 \), and sends \( s \) items when gossiping. In each round, every node randomly selects one of its neighbors and shuffles. In order to make a fair comparison with the simulations with the model, we let the nodes gossip for 1000 rounds before initiating the measurements of the properties. After this start-up period of 1000 rounds, items are replicated and the replicas fill the caches of all nodes fulfilling the uniform distribution requirement of the model. At round 1000, item \( d \) is inserted into the network at a random location. From that moment on we track its replication and coverage.

**Simulations with the model** For the simulations with the model, \( n, c \) and \( s \) are only system parameters. Instead of maintaining a cache, each node in the network only maintains a variable that represents whether it holds item \( d \) or not (state 1 or 0, respectively). Nodes update their state in pairs according to the transition probabilities introduced before (Fig. 3.2). This mimics an actual exchange of items between a pair of nodes according to the shuffle protocol. While in the protocol this results in both nodes updating the contents of their caches, in a simulation using the analytical model updating the state of a node refers to updating only one variable: whether the node is in possession of the item \( d \) or not. To sum up, we use transition probabilities to update the state of one variable. Since we do not need a start-up time for the simulations with the model, at round 0 we set the state of a random node to 1 (while all the other have state 0) and track the state of the
Figure 3.7: The shuffle protocol (left) and the model (right), for $N = 2500$, $n = 500$, $c = 100$ and different values of $s$.

nodes for the remainder of the simulation.

Fig. 3.7 shows the behavior of both the shuffle protocol and the analytical model in terms of replication (upper row of Fig. 3.7) and coverage (lower row of Fig. 3.7) of $d$, for various values of $s$. Each curve in the graphs represents the average and standard deviation calculated over 100 runs. The experiments with the model calculate $P_{\text{drop}}$ using the simplified formula $\frac{n-c}{n-s}$ described in Sec. 3.3.3. It can be observed very clearly that the results obtained from the model (right) resemble closely the ones from executing the protocol (left).

We note that in all cases, the network converges to a situation in which there are 500 copies of $d$, meaning that replication is $\frac{500}{2500} = 0.2$; this agrees with the fact that $\frac{c}{n} = \frac{100}{500} = 0.2$. Moreover, our experiments show that replication and coverage display the fastest convergence when $s = 50$; this agrees with the fact that $n - \sqrt{n(n-c)} = 500 - \sqrt{500 \cdot 400} \approx 50$ (cf. Sec. 3.3.5).

3.5. MODELLING WITH DIFFERENTIAL EQUATIONS

In this section, we exploit the analytical model of information dissemination to perform a mathematical analysis of replication and coverage with regard to the
shuffle protocol. For the particular case of a network with full connectivity, we can find explicit expressions for the dissemination of a generic item $d$ in terms of the probabilities presented in Sec. 3.3. We construct two differential equations that capture replication and coverage of item $d$ from a round-based perspective. The advantage of this approach is that we can determine the long-term behavior of the system as a function of the parameters. Differential equations have been previously used, for example, in [Mitzenmacher 2001; Gupta et al. 2007; Ko et al. 2008] to model probabilistic protocols for large-scale distributed systems.

The characteristics of the replication and coverage are influenced by the topology of the network. It is the topology of the network that will dictate the likelihood of any two nodes to gossip. In order to model the properties of the protocol, it is crucial that we know the probabilities of a node in a given state (0 or 1) to interact with another node in a given state. For this reason, we choose to model the properties for a fully-connected network, where a node can gossip with any other node in the network. This topology allows us to easily calculate the probability of a node in state 1 to interact with a node in state 0 or to interact with another node in state 1. Knowing this, we concentrate on applying the state transition probabilities that we calculated in Section 3.3. The aim of this section is to provide an example of how the transition probabilities can be used to model the properties of dissemination for a specific topology.

### 3.5.1. Replication

One node introduces a new item $d$ into the network at time $t = 0$, by placing $d$ into its cache. From that moment on, $d$ is replicated as a consequence of gossiping among nodes.

Let $x(t)$ represent the percentage of nodes in the network that have $d$ in their cache at time $t$, where each gossip round takes one time unit. The variation in $x$ per time unit $\frac{dx}{dt}$ can be derived based on the probability that an item $d$ will replicate or disappear after an exchange between two nodes, where at least one of the nodes has $d$ in its cache:

$$\frac{dx}{dt} = [P(11|10) + P(11|01)] \cdot (1 - x) \cdot x - [P(10|11) + P(01|11)] \cdot x \cdot x$$

The first term, $P(11|10) \cdot x \cdot (1 - x)$, represents duplication of $d$ when a node that has $d$ in its cache initiates the shuffle, and contacts a node that does not have the item. The second term, $P(11|01) \cdot (1 - x) \cdot x$, represents the opposite situation, when a node that does not have the item $d$ initiates a shuffle with a node that has $d$ and item $d$ is replicated as a result. The third and fourth terms in the equation (which decrease the replication) represent the cases where both gossiping nodes
Figure 3.8: Percentage of nodes in the network with a replica of item $d$ in their cache, for $N = 2500$, $c = 100$, $s = 50$, and $n = 500$, $n = 1000$ or $n = 2000$.

have $d$ in their cache, and after the exchange only one copy of $d$ remains. Substituting $P(11|10) = P(11|01) = \frac{c-s}{c-n-s}$ and $P(10|11) = P(01|11) = \frac{s-c}{c-n-s}$, we obtain

$$\frac{dx}{dt} = 2 \cdot \frac{s}{c} \cdot \frac{c-s}{c-n-s} \cdot x \cdot (1 - \frac{n}{c} \cdot x)$$

The solution of this equation, taking into account that $x(0) = \frac{1}{N}$ with $N$ the number of nodes in the network, is

$$x(t) = \frac{e^{\alpha t}}{(N - \frac{n}{c}) + \frac{n}{c} e^{\alpha t}}$$

(3.3)

where $\alpha$ denotes $2 \frac{s}{c} \frac{c-s}{c-n-s}$. By imposing stationarity, i.e., $\frac{dx}{dt} = 0$, we find the stationary solution $\frac{c}{n}$. Hence, this calculation confirms the observation in Sec. 3.2.3 that the network converges to a situation in which replication of $d$ is $\frac{c}{n}$.

We evaluate the accuracy of $x(t)$ as a representation of the fraction of nodes carrying a replica of $d$, by running a series of experiments where $N = 2500$ nodes execute the shuffle protocol, and their caches are monitored for the presence of
d. Unlike the experiments in Sec. 3.4, we assume full connectivity; that is, for each node, all other nodes are within reach. After 1000 rounds, where items are disseminated and replicated, a new item d is inserted at a random node, at time $t = 0$. We track the number of replicas of d for the next 1000 rounds. The experiment is repeated 100 times and the results are averaged. These simulation results (average and standard deviation) for the protocol, together with the predicted value $\xi$ from the stationary solution.

We repeat the calculation from Sec. 3.3.5, but now against $x(t)$, to determine which size of the exchange buffer yields the fastest convergence to the steady-state for both replication and coverage. That is, we search for the $s$ that maximizes the value of $x(t)$. We first compute the derivative of $x(t)$ with respect to $s$, and then derive the value of $s$ that maximizes $x(t)$, by taking $\frac{dx}{ds}|_{s} = 0$: $x(t) = \frac{\partial}{\partial s} = \frac{2^{d}(cN-n)(c+x(1-2n)+st)}{(c+(c+e^{k})n)(n-s)^{2}}$, where $k = 2\frac{t}{e^{k-n}}$. Let $x(t, s) = 0$. For $t > 0$, $cn = s(2n - s)$. Solving this equation we get $s = n \pm \sqrt{n(n - c)}$. Taking into the account that $s \leq n$, the only solution is $s = n - \sqrt{n(n - c)}$. This also coincides with the optimal exchange buffer size found in Sec. 3.3.5.

### 3.5.2. Coverage

We use the term coverage to denote the percentage of nodes in the network that have seen item d from the moment it was introduced into the network. Let $y(t)$ represent the coverage of d at time t. The variation in coverage per time unit, $\frac{dy}{dt}$, is determined by the fraction of nodes that have not seen d, 1 - y, that interacts with nodes that have d in their cache, x. Let $* \in \{0, 1\}$, then:

$$\frac{dy}{dt} = \frac{1}{2} \cdot (P(1*|01) \cdot P(1*|01) \cdot (1-y) \cdot x + P(1*|10) \cdot P(1*|10) \cdot x \cdot (1-y))$$  (3.4)

The expression above is divided in two parts. The first part represents the nodes that discover item d by initiating a shuffle with nodes that have d in their cache ($P(1*|01)$) and the second represents the nodes that discover d after being contacted by nodes that have d ($P(1*|10)$). Each of these scenarios has a 50% chance of occurring, hence the $\frac{1}{2}$. In both cases, to increase the coverage it is necessary that the node that discovers d does not give away its copy of d within the same round to another node, as indicated by the probabilities $P(1*|01)$ and $P(1*|10)$, respectively. This is because coverage is measured only at the end of a gossiping round, meaning that a node that sees item d for the first time, and drops it in the same round, is considered not to have seen item d yet.\(^5\) Since nodes

\(^5\)The reason for this is that the application has an opportunity to read from the lower-level cache
shuffle, on average, twice per round (once when they initiate the shuffle and again if they are contacted by a neighbor), giving away a copy of \(d\) that a node has just discovered can occur under two scenarios: i) the node acquired \(d\) by initiating an exchange with a node that had \(d\) \((P(1*|01))\) and next lost its copy of \(d\) when shuffling with a node that contacted it, or ii) the node was first contacted by a node that sent a copy of \(d\) \((P(*1|10))\) and afterwards initiated a shuffle and gave away its copy of \(d\).

The value of the probability \(P(*1*1)\) can be calculated from the following cases: a) the gossip partner of the node does not have \(d\), and: i) the state of two nodes does not change after the gossip \((P(01|01) \cdot (1-x))\), and ii) the gossip partner obtains a copy of \(d\) after the gossip \((P(11|01) \cdot (1-x))\); and b) the gossip partner of the node has \(d\), and: i) two nodes have the same state after the exchange \((P(11|11) \cdot x)\), and ii) the gossip partner loses its copy of \(d\) after the gossip \((P(01|11) \cdot x)\). Hence,

\[
P(*1*1) = (P(01|01) + P(11|01)) \cdot (1-x) + (P(11|11) + P(01|11)) \cdot x
\]

Due to the symmetry of both gossiping nodes, \(P(*1*1) = P(1*1*)\).

Substituting these probabilities into (3.4), we obtain

\[
\frac{dy}{dt} = \frac{s}{c} \cdot \left(1 - \frac{s}{c} + \frac{s}{c} \cdot \frac{c-s}{n-s} \cdot \left(2 - \frac{s}{c}\right) + \left(\frac{s}{c} \cdot \frac{c-s}{n-s}\right) \cdot x \right) \cdot (1-y) \cdot x
\]

The solution of this equation, taking into account that \(y(0) = \frac{1}{N}\), is

\[
y(t) = 1 - (N-1) \cdot N^{\beta-1} \cdot \left(\frac{n}{c} + \frac{n}{c} \cdot e^{\alpha w}\right)^{-\beta} \cdot e^{-\lambda t}
\]

where \(\lambda\) denotes \(\frac{\alpha}{\beta} \cdot \frac{\varepsilon}{\beta} \cdot \left(1 - \frac{n}{c} \cdot e^{\alpha w}\right)\), and \(\beta\) denotes \(\frac{\alpha + \varepsilon}{\beta} \cdot \left(1 - \frac{n}{c} \cdot e^{\alpha w}\right)\), wherein \(\kappa\) is \(\frac{\varepsilon}{\alpha} \cdot (1 - \frac{n}{c} \cdot e^{\alpha w})\). By imposing stationarity \(\frac{dy}{dt} = 0\), we find the stationary solution 1, meaning that eventually all nodes will see \(d\).

In order to evaluate how closely \(y(t)\) models coverage, we use the traces from the simulations executed for Sec. 3.5.1. At every round, the nodes that carry a replica of \(d\) are identified, and a record of the nodes that have seen \(d\) since it was published is kept.

Fig. 3.9 presents the coverage measured for three sets of experiments, each set with a different value for \(n\). As \(n\) increases, a newly inserted item requires more time to cover the whole network. This is due to having more competition only once every round.
from other items to create replicas in the limited space available, as was previously shown in Fig. 3.8. However, as predicted by the stationary solution, in all cases the coverage eventually reaches 1.

As shown in Fig. 3.9, the solution $y(t)$ models the behavior observed in simulations in case of $n = 2000$ and $n = 1000$, falling nicely within the standard deviation of the simulation results. However, for smaller $n$ (i.e., $n = 500$), the solution $y(t)$ becomes less accurate, converging slower than the simulation results.

Next, we describe a more accurate version of the coverage model, which considers that nodes may be contacted an arbitrary number of times with a certain probability defined by the network topology.

### 3.6. COVERAGE MODEL (REVISITED)

The coverage model in Sec. 3.5.2 has an implicit assumption that a node shuffles two times per round (once when it initiates the gossip and once when it is
contacted by another node. Although this is true on average, what actually happens is that a node initiates a shuffle once per round (with some random neighbor), but can be contacted an arbitrary number of times ($\leq N - 1$). In Fig. 3.10, it is depicted for $k \geq 1$ which percentage of nodes, on average, shuffle $k$ times in a given round. The number of times a node shuffles in a round has an impact on coverage. In this subsection, we revisit our coverage model to take into account that there is a probability distribution for the number of times a node is contacted.

We compute the probability that a node is contacted $i$ times by other nodes in the network:

$$C(i) = \binom{N - 1}{i} \left( \frac{1}{N - 1} \right)^i \left( \frac{N - 2}{N - 1} \right)^{(N-1)-i}, \quad i \leq N - 1$$

Namely, there are $\binom{N-1}{i}$ ways to choose $i$ nodes in a network of $N - 1$ nodes that contact the given node. Those $i$ nodes contact the given node with probability $\frac{1}{N - 1}$ each, and the remaining $(N - 1) - i$ nodes do not contact the given node with probability $\frac{N - 2}{N - 1}$.

A node may discover $d$ during any of its shuffles. However, coverage is only measured at the end of a gossip round, meaning that a node that sees item $d$ for the first time and drops it in the same round is considered not to have seen item $d$ at all.\footnote{The reason for this is that in general the application has an opportunity to read from the lower-level cache only once every round.} Consequently, coverage increases only under the following conditions: i) a
node that has not seen item $d$ obtains a copy of item $d$ during one of the contacts, and ii) by the end of the round (after $i$ exchanges), it still holds on to a copy of $d$.

In order to model coverage, we need to express the probabilities $P_{\text{get}}$ that a node that does not have a copy of $d$ gets this copy in a shuffle, and $P_{\text{lose}}$ that a node that has a copy of $d$ loses this copy in a shuffle.

$$P_{\text{get}} = P(1^*|0^*) = x \cdot P(1^*|01) = x \cdot (P(10|01) + P(11|01))$$

$$P_{\text{lose}} = 1 - P_{\text{get}}$$

$$P_{\text{¬lose}} = P(1^*|1^*) = x \cdot P(1^*|11) + (1-x) \cdot P(1^*|10)$$

$$= x \cdot (P(10|11) + P(11|11)) + (1-x) \cdot (P(10|10) + P(11|10))$$

$$P_{\text{¬lose}} = 1 - P_{\text{¬lose}}$$

The increase in coverage from one round to the next can be modelled by identifying all the possible cases where a node that has previously not seen item $d$ discovers it by the end of the round. Let $\Phi_i$ express the probability that a node that does not hold $d$, does hold $d$ after performing $i$ shuffles. We have

$$\Phi_i = \sum_{m=0}^{i-1} (1 - \Phi_m) \cdot P_{\text{get}} \cdot (P_{\text{¬lose}})^{(i-m)-1}, \quad i \geq 0$$

where $\Phi_0 = 0$. Namely, the expression $(1 - \Phi_m) \cdot P_{\text{get}} \cdot (P_{\text{¬lose}})^{(i-m)-1}$ captures the case where a node does not have the item at the end of $m < i$ shuffles (with probability $1 - \Phi_m$), then discovers it in the $m$th shuffle (with probability $P_{\text{get}}$), and does not lose it in the remaining shuffles (with probability $(P_{\text{¬lose}})^{(i-m)-1}$).

In a given round, only the fraction $1 - y$ of nodes in the network that did not yet see item $d$ can contribute to an increase in coverage. Such a node is contacted $i \geq 0$ times in the round with probability $C(i)$, and then performs $i+1$ shuffles in total. With probability $\Phi_{i+1}$ the node will hold item $d$ at the end of the round. Thus, coverage can be modelled by the equation

$$\frac{dy}{dt} = (1 - y) \cdot \sum_{i=0}^{k} C(i) \cdot \Phi_{i+1}$$

where $k$ is the maximum number of times that a node is contacted in a round, i.e., $k = N - 1$.

As before, $y(0) = \frac{1}{N}$, as initially only one of the $N$ nodes holds $d$.

For a network of 2500 nodes with full connectivity, the probability of a node being contacted more than four times in one round is negligible (less than 1%). We therefore use the aforementioned coverage model with a limit of $k = 4$ to estimate the coverage and compare with our simulation traces. The results can be seen in Fig. 3.11.

Unlike the results from Fig. 3.9, the current model not only falls into the standard deviation of the shuffle simulation results, but also closely reproduces the curve of average values in all three cases ($n = 500$, $n = 1000$ and $n = 2000$).
Figure 3.11: Percentage of nodes in the network that have already seen a replica of item \(d\), for \(N = 2500\), \(c = 100\), \(s = 50\), and \(n = 500\), \(n = 1000\) or \(n = 2000\).

### 3.7. TIME COMPLEXITY OF EXPERIMENTS

In this section we discuss some topics/issues that arose during the development and testing of the model for the shuffle protocol.

During the experimental phase of this work, we observed a remarkable disparity between the time required to run an experiment with the shuffle protocol or with the model. In both cases, the experiment was the same in terms of properties being measured and parameters used (cache size \(c\), exchange buffer size \(s\), number of different items \(n\) and network size \(N\)).

The difference in execution times can be traced back to the two different algorithms executed at each simulated node. From a node’s point of view, the shuffle protocol requires the selection of items to send to the gossiping partner and the selection of items to keep for the next round. The first operation can be done in linear time, \(O(c)\). The second operation requires checking if the incoming items are already in the cache and removing entries from the cache if space is needed. These steps can be done in \(O(c \cdot \log c)\) time. With the second operation dominat-
Now, let us look at the time complexity of the model. Unlike the protocol implementation, the model requires very little state, namely the value of the parameters and a variable to indicate the presence or absence of item $d$. At each round, each simulated node and its gossiping partner only have to determine their current state $(a_1, b_1)$ and transition to a new state $(a_2, b_2)$ according to the appropriate transition probabilities. The transition probabilities themselves do not change during the simulation (since they depend solely on the parameters $c, s$ and $n$), so they are precomputed at the initialisation phase. As a result, the execution of the model for each node has a constant time complexity, $O(1)$.

The defining factor in the execution time of the simulations with the shuffle protocol is the size of the cache. With a cache size of 100 for all of our experiments, the execution times for our simulations with the shuffle protocol and the model differed by approximately two orders of magnitude. Considering that large networks are needed to clearly observe emergent behavior and that this behavior evolves over many rounds, the value of having a model becomes evident. Being simply parameters in the model, the cache size and exchange buffer size have no effect on the memory requirements and execution time of the simulation, thus freeing computational resources to experiment with larger networks and different topologies.

3.8. CONCLUSIONS

In this chapter, we have demonstrated that it is possible to model a gossip protocol through a rigorous probabilistic analysis of the state transitions of a pair of nodes engaged in the gossip. We have shown, through an extensive simulation study, that the dissemination of a data item can be faithfully reproduced by the model. Having an accurate model of node interactions, we have been able to carry out the following:

- After finding precise expressions for the probabilities involved in the model, we provide a simplified version of the transition probabilities. These simplified, yet accurate, expressions can be easily computed, allowing us to simulate the dissemination of an item without the complexity of executing the actual shuffle protocol. These simulations use very little state (only some parameters and variables, as opposed to maintaining a cache) and can be executed in a fraction of the time required to run the protocol.

- The model reveals the relationships between the parameters of the system.
Armed with this knowledge, we successfully optimized one of the parameters (the size of the exchange buffer) to obtain the fastest convergence of the observed properties.

- Under the assumption of full connectivity, we are able to use the transition probabilities to model the properties of the dissemination of a generic item. Each property is ultimately expressed as a formula which is shown to display the same behavior as the average behavior of the protocol, verifying the validity of the model.

While gossip protocols are easy to understand, even for a simple push/pull protocol, the interactions between nodes are unexpectedly complex. Understanding these interactions provides insight into the mechanics behind the emergent behavior observed in gossip protocols. We believe that understanding the mechanics of gossiping is the key to optimizing (and even shaping) the emergent properties that make gossiping appealing as communication paradigm for distributed systems.
Thus far, we have engaged in a thorough study of gossiping as a mechanism for information dissemination. At first instance, we explored the properties of the dissemination experimentally, simulating a variety of scenarios within the context of an application (the Gossip-based News Service). The positive results from that stage led us to delve into a probabilistic analysis of the dissemination of a single data item that resulted in the modelling of the interactions between gossiping nodes. Having verified the properties observed in simulation through theoretical analysis and having observed that the shuffle protocol successfully replicates and disseminates an item throughout the network, we can conclude that gossiping is a promising mechanism for dissemination in wireless networks. However, the underlying assumption behind our analysis is that the nodes participating in the gossip do so in good faith and, as such, adhere to the requirements of the gossip protocol. Unfortunately, this might not always be true.

The assumption of good behavior excludes two very likely scenarios: the possibility of faulty nodes or, more troubling, the presence of malicious nodes in the network. Faulty nodes may deviate from the original protocol in unpredictable ways (e.g., gossiping at different frequencies or sending more data items than required.) On the other hand, malicious nodes may have a specific motivation for misbehaving. In a network of gossiping nodes, a clear motive for misbehaving would be to replicate and disseminate a data item faster than the rest. In other words, spamming the network with certain data items.

Once a problem associated only with email, spam is now affecting other media, such as instant messaging, blogs, newsgroups and mobile phone messaging. As wireless networks become more commonplace, we can expect that spam will find its way into upcoming wireless communication services. This chapter stud-
ies the threat posed by malicious nodes inserting spam in a wireless network using gossiping as a method for information dissemination. We identify the security mechanisms needed to protect our gossip network against the proliferation of spam, reducing the problem to a matter of finding and removing corrupted messages. Finally, we propose a probabilistic method of integrity checking to contain the spread of spam which we evaluate through extensive simulations.

4.1. INTRODUCTION

Being an extremely robust and scalable communication model, gossiping appears to be an ideal solution for information dissemination in highly dynamic environments, such as wireless networks. The simplicity and distributed nature of gossiping has already sparked interest for its use in wireless environments, ranging from sensor networks to MANETs. However, while gossip networks are often described as being robust to failures, their ability to cope with malicious behavior is rarely addressed. They can gracefully handle the departure of more than half of their members, but this strength would not be as impressive if a few malicious insiders could cause serious damage.

The effectiveness of gossiping is based on the collective effort by the nodes in the network, which results in the workload (and responsibility) being divided among the collection of nodes. With every node playing an equal role in the network, adhering to the agreed code-of-conduct is essential. However, assuming that every node will behave appropriately would be naive. It can be expected that the introduction of malicious nodes will disturb the balance in the gossip network. The extent of the disruption is the focus of this study.

In this chapter, we explore the effect of having malicious nodes in a wireless gossip network used for information dissemination. The attack of choice for these malicious nodes is spamming. In the broadest sense of the word, spam is defined as unsolicited email. While spam often refers to unrequested emails of commercial nature that are sent in bulk, the term is also used to describe irrelevant or inappropriate messages in newsgroups or message boards, as well as non-commercial emails (religious, political, etc.) or junk mail. Nowadays, spam is not restricted to email anymore. It has made its way into other media, such as instant messaging, blogs, newsgroups, p2p networks and mobile phone messaging. For the purpose of this chapter, we refer to any kind of message placed in the network as a result of malicious behavior as spam.

For the most part, nodes in a wireless network have limited resources compared to the average wired workstation making spam a serious threat and not just a nuisance. As our network is being used to disseminate information, it is only
natural that selfish nodes would try to exploit the system by overloading it to suit their needs. The intent of these malicious nodes may be to achieve maximum exposure or even to destroy the system by polluting it with junk. Regardless of their motivation, our interest lies in determining the extent of the threat and minimizing the damage as much as possible without resorting to expensive and complex solutions.

4.1.1. The Problem with Spam in Wireless Gossip Networks

Gossiping as a method for information dissemination relies strongly on information being forwarded through randomly chosen paths. At each step, information is passed along to another peer selected on-the-go, making it virtually impossible to anticipate the path that a piece of information will travel. This random movement of information works in favor of dissemination as it ensures that information will find its way to all peers with certain probabilistic guarantees.

The problem with spam in a gossip network is intrinsically related to the dis-
semination properties of gossiping. As has been noted before [Demers et al. 1987], once a piece of information is gossiped it is extremely hard to remove it from the network unless special mechanisms for removal are in place. This makes the spam problem much more severe in a gossip network than spam email on the internet, from a theoretical point of view. Gossip networks used for data dissemination reduce the amount of work for the spammer to the bare minimum of injecting the spam and then sit back and watch as all other peers collaborate to deliver the spam. There is no need for the spammer to go through the process of trying to collect the addresses of potential targets. Knowing only one node in the network is enough for the spammer to start operating. After all, all other nodes will make sure that his/her message is delivered.

As for accountability, the spammer is in an enviable position. In gossip networks, the nodes themselves act as routers for the delivery of data. As a result, nodes can not be held accountable for the data they deliver. This makes tracking down the source of the spam (or any other piece of data for that matter) very difficult. In addition, gossip networks are often promoted on the merits of being decentralized (no central authority) and robust (being able to deal with nodes coming and going gracefully). This works in favor of a malicious node too, as there is no central authority to keep track of its behavior and its sudden joining or leaving will not disrupt the network or be seen as suspicious.

4.1.2. Motivating example

The magnitude of the damage that can be caused by a few malicious nodes in a gossip network can best be illustrated through an example. Consider a collection of nodes arranged in a grid, gossiping with their four neighbors to the North, South, East and West. They gossip according to the shuffle protocol, which was introduced in Chapter 2. The most important observation to make about the shuffle protocol is that any two nodes that engage in a shuffle essentially swap a number of data entries from their caches. In doing so, they not only preserve the data that are collectively stored in the network, but also “move” these data around in a seemingly random fashion. The underlying idea is that by randomly shuffling data entries between nodes, all nodes will be able to see all data eventually.

Nodes gossip periodically, swapping half of the contents of their caches with a randomly chosen neighbor. In our scenario shown in Figure 4.1, after nodes have been shuffling for some time (50 cycles or rounds), malicious nodes appear. These spammers account for 1% of the network, but the effect of their actions is devastating to the network. Instead of forwarding the messages from their peers, they drop them and replace them with spam. Figure 4.1 shows how the caches of the nodes become polluted with spam. In just 5 rounds their presence can be felt. Gradually, they replace valid items with their own, filling up the network with
spam. Eventually, the network will be saturated with spam at 100% and all valid items will have been lost.

4.1.3. Contribution

Our contribution is twofold. First, we show that wireless gossip networks are highly vulnerable to the proliferation of spam. In fact, we claim that without any security mechanisms, spammers could easily take advantage of the dissemination properties of gossiping to overwhelm the network, consuming valuable resources (storage space, bandwidth and processor cycles) at the same time. Second, by means of well-established security measures, we reduce the spam problem to a matter of integrity checking. Additionally, we propose a probabilistic solution for verifying the integrity of messages which succeeds not only in reducing the amount of spam in the network, but also in restricting its dissemination.

4.2. SYSTEM MODEL

4.2.1. General Description

We focus on a system where a heterogeneous mix of fixed and mobile nodes, ranging from mobile devices such as PDAs and smart phones to PCs with internet access, collaborate by volunteering storage space for the creation of a collective data space. Users in the system are able to publish events, which we call items, of interest to other users. The nodes in the system devote a limited amount of space, which we refer to as their caches, to store items. The collection of caches of all nodes in the network makes up a collective data space.

The caches are updated periodically using the shuffle protocol, introduced in Chapter 2. As a result, the items in a node’s cache are in transit, which means that they could be exchanged for other items at any moment. Items are not purposefully routed. Once published, they become part of the collective data space, replicating themselves (the number of replicas is dictated by the storage capacity of the network) and moving freely through the network (geographic restrictions for the dissemination of items are also possible).

Taking part in gossip exchanges results in a node populating its cache with a collection of items. As the cache size is limited, the contents of a cache constitute a sample of the totality of items available in the network.

Users of the system can discover items of interest by going through the items in the local cache. Depending on the number of items in the network, the local cache may not contain all items of interest to a user at a particular time. Nevertheless, as demonstrated experimentally (in Chapter 2) and through modelling
(in Chapter 3) all items of interest can be discovered after participating in enough gossip exchanges. Items of interest can then be stored separately in the node’s private data store.

4.2.2. Assumptions

Items can be published by any user of the system and are propagated through the network in the form of entries. While an item is a piece of information, an entry is the representation of the item in the network and for each item several entries may exist. The dissemination of entries occurs between neighboring nodes that exchange entries. As entries are gossiped, replication may occur naturally if a node has available storage space to keep a copy of an entry. As a result, after an item is published and gossiped, many entries for this item may be present in the network. The number of entries per item is dictated by the capacity of the network and the number of items published, as explained earlier in 2.3.3.

A unique id is associated with each node. The entries that a node inserts into the network can be uniquely identified by a combination of the node id and a sequence number. In its most basic form, an entry contains a unique id, a timestamp and a time-to-live. There may be other fields of information depending on the application. A limited number of these entries can be stored by each node in its local cache. A node can store, at most, $c$ entries in its cache. For our experiments, all nodes have the same cache size $c$. Nodes in the network gossip periodically, exchanging the entries in their caches. We define a round as a gossiping interval in which each node initiates an exchange once.

4.3. PREPARING FOR THE FIGHT

Securing a gossip-based system like the one we propose against malicious nodes requires 1) regulating the entry of nodes into the system (access control), 2) being able to accurately identify the source of an item (source node authentication) 3) ensuring the integrity of messages and 4) enforcing fair use of the system (rate control). We discuss these issues in turn.

4.3.1. Access Control

To ensure that only authorized nodes can join the network, issuing credentials for these nodes is required. One possible solution is to have a Certification Authority (CA) certify a public key for each node. This procedure would only take place once, establishing the identity of each node and allowing nodes to refuse communication with outsiders. In a CA-based solution like this the nodes would need to
pass along their public key as well as a signature by the CA on the public key.

It can be argued that having a centralized component like a CA, even if only during the initial phase, goes against the spirit of ad hoc networking. This problem has been addressed with the proposal of distributed CAs. Zhou and Haas [Zhou and Haas 1999] propose the idea of having a distributed certificate authority by using threshold cryptography, where a cryptographic operation is split among multiple users. Later works build on this idea and propose refinements to improve the access to the CA services [Kong et al. 2001] and improve efficiency [Khalili et al. 2003]. The latter combines ideas of threshold and ID-based cryptography allowing nodes to use their ID as their public key. The key generation service is distributed among the participants allowing them to obtain the private key corresponding to their identity by contacting a predefined number of nodes in the network.

4.3.2. Source Node Authentication

By the time an item arrives at a particular node’s cache, it has most likely been shuffled around several times by other nodes. As a result, when a node receives an item from a neighbor, it cannot make any assumptions about the item’s origin. In order to be able to identify the source of an item, it is necessary for the item to be digitally signed by the original node who published it.

4.3.3. Integrity

Given that most likely an item has been forwarded several times before reaching an interested user, the item has to be protected against malicious insiders who may want to modify its contents. By having the source sign the item, a user can check if the item has been modified along the way.

An ideal solution for preserving the integrity of items in the network would be to verify the integrity of each item at every hop. This would require that every node executes a public key signature verification operation for every item it receives from a neighbor. The computational workload of such a solution could be prohibitive for mobile and embedded devices. [Roman et al. 2007] evaluates the suitability of a collection of cryptographic primitives for resource-constrained devices. Using two different platforms (MicaZ and TelosB) to test software implementations of public key cryptography, the results for verification of signatures measure in the order of seconds. More recent work [Granjal et al. 2008], evaluates several well-known symmetric key and hashing algorithms on MicaZ motes. The memory requirements for the algorithms are, in most cases, too high for these generation of nodes and the computational and energy demands imposed by cryptography are significant. Unless hardware implementations of the algorithms are
available, the cost of verifying every item, considering that in the gossip protocol we use several items are received from a neighbor, is unfeasible.

Instead of verifying every item, a more efficient way of ensuring the network remains free of forged items is for every node to do a batch verification of the signatures on items received from a neighbor. Verifying multiple signatures in batches is less expensive than verifying each signature at a time. We elaborate more on this and propose our own solution for ensuring the integrity of items in Sections 4.4 and 4.5, respectively.

4.3.4. Rate Control

The shuffle protocol ensures that, on average, each item has the same number of entries in the network. Given that the collective storage space is limited, a larger number of different items in the network results in a smaller number of entries per item. Therefore, a node producing an excessive number of items would occupy a large portion of the storage space with its items reducing the number of entries that other nodes can place. To ensure that nodes do not abuse the system by flooding the network with their own items, a mechanism for rate control is needed.

This flooding of items by a node is a form of a denial-of-service attack. As shown in Chapter 2 (see 2.3.3), the dissemination speed of items through the network is inversely proportional to the number of items published. It follows that the insertion of an excessive number of items by one node has a negative effect on the performance of the system, given that dissemination speed is sensitive to the number of items in the network. In essence, more items in the network (due to one node’s excessive publishing) result in the dissemination speed of all items slowing down. For this reason, it is necessary to prevent a node from publishing an excessive number of items. Otherwise, a single “overactive” node could cause the service to slow down to the point of not being useful anymore.

Rate control can be enforced by restricting the id space of items per node. This way, a node would be allowed to have at most \( x \) items in the network at any point in time, where \( x \) is the size of the id space of items per node. For example, the id space of items per node could be restricted to \( n \) bits resulting in \( 2^n \) items. After a node has published \( 2^n \) items with different ids, the next published item will have the same id as one of the previously published items. Since nodes are only allowed to hold one entry per item based on the item id, the more recently published item will overwrite the older item in the network resulting in an upper bound for the amount of storage space occupied by a node’s items. With \( d \) published items in the network, a node could only occupy at most \( x/d \) of the collective storage space.
4.4. SPAMMING THROUGH THE CORRUPTION OF MESSAGES

The shuffle protocol ensures fairness, meaning that each node can use up the same fraction of collective storage space for its items. As a result, a malicious node can insert only so much spam under its own identity. In order to place more spam in the network, a malicious node would have to utilize the ID space of items per node assigned to other nodes. Analogous to the way an email spammer uses false email identities to increase the likelihood that his spam makes its way into our inboxes, a malicious node in our gossip network can place more spam by corrupting the content of the entries that pass through its cache. In essence, a malicious node would be replacing the content of other nodes’ entries with its own while keeping the entries’ metadata (ID, signatures, ...) intact. This way, the spammer can steal the storage space of other nodes and create more instances of its messages. The spam problem then becomes a problem of preserving the integrity of messages.

4.4.1. The threat of malicious insiders

With the measures to prevent unauthorized nodes from infiltrating the network in place, being able to cope with attacks from malicious insiders becomes the biggest challenge. Our gossip network obtains its desirable properties from the periodic execution of a specific gossip protocol at every node. Having nodes in the network behave differently can pose a major threat to the system.

Unlike fixed networks, wireless networks rely on nodes forwarding messages for their neighbors. Without a trusted routing infrastructure available, a great deal of responsibility is placed on the forwarding nodes to deliver a message. Assuming that malicious nodes are present, at every hop there is a chance that the message might be tampered with. In addition to this, we are dealing with a gossiping system that relies heavily on randomness to forward its messages. The combination of these two aspects result in malicious insiders having plenty of opportunities to corrupt the content of messages and be safe from detection due to the random nature of gossiping.

4.4.2. Checking all messages at every hop

A conventional approach to security can be applied to ensure the integrity of items. Under this scheme, all entries in the network are required to be signed by their publisher and are subject to integrity checks. Integrity checks can be used to fight attacks based on replays of old entries and modification of entries, as the checks would discover that the content of the entry has been tampered with. However, given that in our system items are constantly being gossiped, verifying all entries
received during a gossip exchange would be computationally very expensive. In essence, an entry would have to be verified at every hop. Even though doing this would permit the identification of malicious nodes as soon as they appear, the cost of following this approach would be prohibitive.

### 4.4.3. Batch Verification

An alternative to verifying the signatures of the entries received one-by-one is to do a batch verification [Bellare et al. 1998]. Verifying multiple digital signatures simultaneously, instead of verifying each one individually, can be done at lower costs with different schemes for fast verification of digital signatures in batches. These schemes test the validity of all signatures in a batch and the test would succeed only if all signatures are valid. The drawback is that batch verification does not identify which signatures are invalid in the batch. This is, however, not critical since discovering any invalid signature in the batch would be enough to conclude that we have come in contact with a malicious node.

Having all nodes do batch verification of signatures after receiving items from a neighbor would allow nodes to discard any invalid batch and take measures against the neighbor who forwarded the dubious entries. However, the benefits of using batch verification are only evident when a large number of signatures are tested. Therefore, unless nodes are exchanging a large number of items at a time, batch verification could still be expensive. Furthermore, it is necessary to use a digital signature scheme where its batch verification algorithm allows a batch of signatures from different signers.

### 4.5. PROBABILISTIC VERIFICATION

As an alternative solution to checking all entries at every hop, we propose a more flexible and cost efficient approach to combating malicious nodes. Our solution is based on a probabilistic selection of the entries to be checked.

#### 4.5.1. Selection of entries to verify

The verification phase is incorporated into the shuffle protocol in the following way:

- In `selectItemsToKeep()`, each node decides how to merge the entries in its cache with the entries received from the selected peer. Before merging, a probabilistic verification phase is executed.
SEC. 4.6 EXPERIMENTAL RESULTS

- Each of the received entries is checked with a probability $P_{\text{check}}$. The integrity of an entry is checked by verifying its digital signature. If the entry is valid, then it is marked as checked. Otherwise, the entry is discarded.

- The entries that were not selected for checking and the ones that passed the check are merged into the local cache.

Note that as a result of executing the probabilistic verification phase whenever a set of entries is received, a subset of the entries in a node’s cache will be marked as checked. The next time the node executes the shuffle (and sends a random selection of entries from its cache to a neighbor) some of the entries it sends will be marked as checked. The receiving node may confirm that its neighbor is properly checking and marking entries if it selects one of these checked entries for verification and it passes the test. Before placing any received entries in its cache, the receiver removes the checked marks generated by the sender and only flags as checked the entries that it has verified itself.

4.5.2. Attack model

To test the validity of probabilistic verification as a technique to counter malicious behavior in the form of corruption of entries, we assume that a small percentage of the nodes in the network are malicious insiders while the rest behaves according to our gossip protocol.

Malicious nodes execute a slightly different version of the shuffle protocol. In selectItemsToSend(), the selected entries are corrupted, with the exception of the entries marked as checked. The reason for this is that sending a corrupted entry marked as checked will raise suspicion if the receiving node executes an integrity check on that entry. In this chapter, other nodes do not make an effort to detect malicious nodes and take measures against them (this is the subject of current study). Nevertheless, we assume that malicious nodes are cautious. As malicious nodes do not want to be trivially discovered, they will execute integrity checks with a $P_{\text{check}}$ probability and will only corrupt entries that are not marked as checked, thus avoiding direct responsibility for any corrupted entry they have forwarded.

4.6. EXPERIMENTAL RESULTS

The results presented in this section correspond to a network of 2500 nodes with a cache size of 100. The nodes were arranged in a square grid topology, with 50 nodes on each side over an area of $50 \times 50$ units. The range of each node was set to 1 unit, making communication possible with the node’s immediate neighbors.
Figure 4.2: Number of corrupted entries in the network when 1% of nodes are malicious.

Figure 4.3: The percentage of corrupted messages in the network decreases with the probability of checking an entry, $P_{\text{check}}$. 
to the North, South, East and West. Nodes placed at random locations in the grid were selected to be malicious. Experiments were conducted for different concentrations of malicious nodes (1%, 2% and 5%).

4.6.1. Amount of Spam in the Network

Malicious nodes carry out a very simple attack: corrupt as many entries that pass through as possible, taking into account that some entries will need to be checked. In the absence of any measures to counter the pollution of the network with corrupted entries, this kind of attack is extremely effective.

Figure 4.2 shows the spread of corrupted entries through the network over time. In the experiment, 25 malicious nodes (1% of the network) appear at round 50 and from that moment start corrupting entries. Entries do a random walk through the network which leads to each item eventually visiting every node in the network, including the malicious ones. As a result, without any integrity checks, the number of corrupted entries keeps increasing until all entries in the network are corrupted. On the other hand, when nodes execute probabilistic checks, the num-
ber of corrupted entries soon reaches an equilibrium where the amount of spam generated matches the amount of spam dropped by non-malicious nodes. Experiments with 50 and 125 malicious nodes (2% and 5% of the network, respectively) show similar behavior, but converging to different levels of spam. In all cases, spam spreads through the network after the appearance of malicious nodes, but after an initial period of growth, the amount of spam settles at a level inversely proportional to $P_{\text{check}}$.

The number of spammers present during an experiment directly affects the amount of spam in the network, as we show in Figure 4.3, which summarizes our experiments regarding the amount of spam in the network. For each number of spammers (25, 50 and 125) and value of $P_{\text{check}}$ (from 5% to 35%, with increments of 5), the level to which the amount of spam converges was recorded (by averaging the last 200 rounds) in order to show the relationship between the amount of spam in the network and the probability of checking a received entry. We observe that the amount of spam is inversely proportional to $P_{\text{check}}$. As can be expected, it is also proportional to the number of spammers in the network. This is due to each spammer creating an independent “spam heap” in its surroundings.

The effect of probabilistic verification is that corrupted entries are restricted from spreading too far away from the source, as they become more likely to be removed by a non-malicious node with every hop. Figure 4.4 shows how spam is contained within an area surrounding the spammer. The snapshots, taken at round 500 for different values of $P_{\text{check}}$, clearly illustrate the benefit of probabilistic verification not only in reducing the amount of spam, but also in decreasing its reach.

4.6.2. Reach of Spam in the Network

In the same way as the amount of spam reaches a particular equilibrium point depending on the checking probability, the average distance (in hops) from the source that spam travels also reaches a stable state, as can be seen in Figure 4.5. In this experiment, for each corrupted entry we record the distance (in hops) from its source and calculate an average distance at every round. This average distance serves as an indicator of how far away spam travels before being discovered and removed. After an initial period where corrupted entries find their way into the caches of nodes in the vicinity of a spammer, the corrupted entries start being dropped, preventing their dissemination any further. Notice how the steady state to which the average distance converges in Figure 4.5 is inversely proportional to $P_{\text{check}}$. Experiments with 2% and 5% of malicious nodes converge to very similar values.

By measuring the value to which the average distance converges (by averaging the last 200 rounds), we can observe its relationship with the checking probability
Figure 4.5: Average distance (in hops) of corrupted entries from their source over
time (in rounds), with 1% of malicious nodes in the network.

as well as the number of spammers. Figure 4.6 shows the average number of hops
away from the source that spam travels with respect to the probability of checking
for 25, 50 and 125 malicious nodes. An important observation depicted in this
graph is that the distance traveled by the spam is independent of the number of
malicious nodes present. In fact, the heaps of spam generated by spammers may
overlap, as seen previously in Figure 4.4.

Figure 4.7, which shows the number of spam entries that have traveled a cer-
tain number of hops, summarizes the effectiveness of probabilistic verification as
a way of reducing and containing spam. In this graph, we observe (after 500
rounds) the distribution of spam according to the distance from the source for var-
ious values of $P_{\text{check}}$. It is evident from the area covered by each curve that higher
values of $P_{\text{check}}$ reduce the amount of spam in the network as well as reduce the
area affected by spam.

4.7. RELATED WORK

Previous work has looked at malicious behavior in wireless ad hoc networks as
a problem of lack of cooperation and selfish behavior (for example, not forwarding
messages to save energy). Security in wireless environments has concentrated
mostly on the routing layer by modifying existing routing protocols, such as DSR
and AODV.

Efforts to alleviate the problem of malicious behavior by enforcing coopera-
Figure 4.6: Average distance (in hops) of corrupted entries from their source for different values of $P_{\text{check}}$. Results for spammers accounting for 1%, 2% and 5% of nodes in the network are shown.

4.8. CONCLUDING REMARKS

In this chapter, we explored the vulnerability of wireless gossip networks to spamming attacks. We showed that the probabilistic nature of information dissemination in gossip networks makes these networks specially susceptible to the proliferation of spam. In an effort to secure the network, we proposed that only accredited nodes be allowed to gossip. With only authorized nodes gossiping, our
efforts focused on dealing with malicious insiders, as these malicious nodes could only spam by taking over the identity of other nodes. This resulted in our proposal of probabilistic verification of messages as a way to fight spam. We evaluated this technique through extensive simulations showing that the amount of spam is effectively reduced and its spread restricted.

As mentioned earlier when describing the attack model, malicious behavior could raise suspicion in neighboring nodes. However, the current protocol does not include a mechanism to react when faced with changes in the amount of spam received. In that sense, the current protocol takes a proactive approach to fighting spam. The problem with this approach lies in the constant toll it takes on the nodes, requiring a fixed number of checks to be performed regardless of the threat.

It is clear that during periods when the threat is low, it would be desirable to lower the number of checks performed. Likewise, when faced with heavy spamming, an increase in the checking would be appropriate. The next chapter, Enforcing Data Integrity, focuses on making this possible by observing the traffic from neighbors and maintaining a variable $P_{\text{check}}$ for each neighbor that a node has. The new goals are then to a) dynamically adjust the checking probabilities for each neighbor and b) be able to detect suspicious behavior by analyzing the traffic at each link, which will allow us to take action against suspicious nodes.

Figure 4.7: Distribution of corrupted entries according to their average distance (in hops) from the source.
CHAPTER 5

Enforcing Data Integrity

Ad hoc networks rely on nodes forwarding each other’s packets, making trust and cooperation key issues for ensuring network performance. As long as all nodes in the network belong to the same organization and share the same goal (in military scenarios, for example), it can generally be expected that all nodes can be trusted. However, as wireless technology becomes more commonplace, we can foresee the appearance of very large, heterogeneous networks where the intentions of neighboring nodes are unknown. Without any security measures in place, any node is capable of compromising the integrity of the data it forwards. Our goal for this chapter is to ensure the integrity of the data being disseminated without resorting to complex and expensive solutions. Building on the work presented in the previous chapter, we aim to enforce data integrity by discouraging malicious behavior in two ways: a) enforcing integrity checks close to the source and b) refusing to communicate with obviously malicious nodes. We find that by having nodes sample their traffic for corrupted messages, malicious nodes can be identified with high accuracy, in effect transforming our collection of nodes into a self-policing network.

5.1. INTRODUCTION

Given the dynamic nature, often unreliable links and lack of a central authority that characterize wireless ad hoc networks, giving any hard guarantees regarding their performance is a difficult task. The situation becomes even more complex if we envision very large networks of heterogeneous nodes. In this scenario, not only would we have to deal with the issue of scale, but also with the fact that we cannot be certain of the willingness of all nodes to cooperate towards a common goal. The lack of a central authority to oversee the good behavior of nodes is a
clear disadvantage.

A common assumption is that nodes adhere to executing the chosen communication protocol. Under this condition, content can be disseminated through the network in a reliable way. However, when some nodes decide not to play by the rules, the characteristics of the dissemination as well as the reliability of the content being forwarded might change. As before, we refer to any kind of message placed in the network as a result of malicious behavior as *spam*, as these unsolicited messages serve only the interest of the malicious node(s) and waste the already limited resources in the network. We use the term malicious node and spammer interchangeably.

This chapter studies the effect of having misbehaving nodes that compromise the integrity of the data being disseminated in an ad hoc network and the measures that can be taken to counteract such malicious behavior. Since we are dealing with large-scale networks, our intention is to develop effective solutions that scale easily. Therefore, we favor simplicity and the use of local interactions and decisions only.

### 5.1.1. The Cost of Guaranteeing Data Integrity

The conventional approach to ensuring the integrity of a message is to require that the message be signed by its creator. By verifying the digital signature on the message, the receiver can be assured of its integrity. However, this procedure is computationally expensive. In a wireless ad hoc network, where nodes act themselves as routers, a message may have traveled several hops before reaching its destination. Due to the lack of a trusted infrastructure for routing, the message might have become corrupted along the way. If that is the case, and the receiver verifies this with an integrity check, the cost incurred due to the corrupted message is not just limited to the verification of the signature, but it also includes the cost of routing. This situation could be avoided by executing integrity checks at every hop. As a result, data integrity would be guaranteed and malicious nodes could be easily detected.

The downside of this approach is the heavy computational load inflicted on the nodes, as each node would have to check every message it forwards. Therefore, even at times when no malicious nodes are present, the nodes in the network would be wasting resources checking valid messages.

### 5.1.2. Contribution

The ideas presented in this chapter build on the previous chapter, which explored the vulnerability of wireless gossip networks to spamming attacks. In Chapter 4,
nodes deal with corrupted content (spam) generated by malicious nodes by executing integrity checks on their incoming traffic with a certain fixed probability $P_{\text{check}}$. The result of these probabilistic checks is a reduction in the amount of spam and its spread away from the source. As can be expected, the effectiveness in fighting spam is highly related to the value of $P_{\text{check}}$ used by all nodes. It should be noted that all nodes execute integrity checks with a probability $P_{\text{check}}$ blindly, regardless of the presence of spammers or corrupted content.

Our current work strives to find a middle ground with regard to the workload imposed on nodes to guarantee the integrity of data in the network. First, we present a probabilistic data verification scheme, which dynamically adapts the workload of each individual node according to the threat of malicious nodes in its surroundings, in essence reducing the amount of work required by nodes that are not in the vicinity of malicious nodes. As a result, the overall workload in the network is kept low and it concentrates around the malicious nodes. Second, we take a proactive approach to enforcing data integrity in the network by having the nodes constantly monitor the good behavior of their neighbors. In addition, we show that the immediate neighbors of malicious nodes are able to detect their suspicious behavior with high accuracy, enabling them to take measures to prevent further corruption of data.

Nodes make their own decisions to regulate traffic according to perceived adherence to good behavior by their neighbors. As a consequence, suspicious behavior is penalized and the malicious nodes are faced with the decision of adhering to the rules or be isolated.

5.1.3. Related Work

In our work, discovery of malicious nodes is made possible by statistical analysis of incoming messages. Unlike reputation-based systems [Buchegger and Le Boudec 2005] where nodes rely on second-hand reputation reports (which could be false) to determine if a neighbor is misbehaving, our approach avoids issues of trust by relying only on first-hand observations to assess the behavior of a neighbor. In that sense, our work lies closer to [Bansal and Baker 2003], which relies only on first-hand observations to build the reputation of a node. Other efforts to alleviate the problem of malicious behavior by enforcing cooperation include payment systems [Buttyan and Hubaux 2000] which assume that a node can be swayed away from his selfish behavior through economic incentives.

From the data integrity point of view, our work is closely related to the efforts to counteract content pollution in peer-to-peer networks [Walsh and Sirer 2005]. A related problem due to malicious behavior is index poisoning [Liang et al. 2006], which could lead to effective distributed denial-of-service attacks [Naoumov and Ross 2006].
Previous work in data aggregation has also addressed the threat of malicious nodes by using cryptographic and statistical techniques [Garofalakis et al. 2007; Przydatek et al. 2003]. In this case, the goal of the attackers is to make a user accept false aggregation results without the user being aware that the results deviate significantly from the true result of the aggregation. Work in this area aims for the user to reject a result with a high probability if it is not a good approximation of the true value of the aggregation. Unlike these projects, the work we present in this chapter is not concerned with in-network processing. In other words, we do not assume that data is processed by intermediate nodes. To the contrary, we want to prevent intermediate nodes from modifying information as it is being forwarded. Form this angle, [Zhu et al. 2004] is closer to our work. The focus in [Zhu et al. 2004] is to detect and filter out false data packets either at or on the way to the base station. While [Zhu et al. 2004] requires node collaboration (to agree on and authenticate a report), our approach is based purely on local observations and decisions. Probabilistic validation methods have also been recently proposed for VANETs [Picconi et al. 2006].

5.2. SYSTEM MODEL

We focus on store-and-forward systems where nodes devote a limited amount of space to store messages. We refer to this space as the node’s cache. The communication medium is wireless and, therefore, messages are disseminated through the network in a multi-hop fashion. We assume that nodes forward a batch of messages at a time.

Concerns regarding authentication and integrity are addressed through conventional security measures. Nodes that publish information are required to digitally sign their messages. Consequently, all messages in the network have a digital signature and are subject to integrity checks. We assume that executing an integrity check for every message received is prohibitively expensive for a node.

Wireless gossip networks fall under these assumptions. The goal of these networks is to achieve reliable, robust and scalable data dissemination. To this end, nodes are required to engage in communication with their neighbors on a regular basis. Executing integrity checks under these conditions would be too expensive and undesirable. With this in mind, we propose a probabilistic solution for integrity verification. Using wireless gossip networks as an example platform, we evaluate the effectiveness of our proposed data integrity enforcement solution.
5.2.1. Example: Gossip-based News

The Gossip-based News Service, as introduced in Chapter 2, serves as the experimental platform to evaluate the data integrity enforcement measures described in this chapter. The service is provided by a mesh backbone made up of a large number of wireless routers that communicate through gossiping using the shuffle protocol. In order to prevent a malicious router from overloading the network with spam, we incorporate security measures into one of the core shuffle methods, `selectItemsToKeep()`. The remaining two core methods remain unchanged. To summarize, our new implementation of the shuffle protocol retains the same structure. Of its three core methods, only one is slightly modified to include a data integrity enforcement mechanism, as follows:

- **selectPeer()**: Select a neighbor randomly
- **selectItemsToSend()**: Randomly select $s$ entries from the local cache and send a copy of those entries (buff\_send) to the selected peer.
- **selectItemsToKeep()**: **Probabilistically verify data integrity.** Add received entries (buff\_recv) to the local cache and remove repeated entries. If the number of entries exceeds $c$, remove entries among the ones that were previously sent (unless they were also in buff\_recv) until the cache contains $c$ entries.

We highlight, in bold letters, the added step to enforce data integrity as nodes disseminate data through the network. In the remainder of the chapter, we examine the effect of adding measures for data integrity enforcement for the case of a wireless gossiping network where a number of malicious nodes compromise the integrity of messages.

5.2.2. Probabilistic Verification

Upon receiving a set of messages from a neighbor, nodes execute integrity checks by verifying a digital signature on each received message with probability $P_{\text{check}}$. If the message is valid, then it is marked as checked and stored in the node’s cache. Otherwise, the message is discarded. We call this process **probabilistic verification.** As introduced in the previous chapter, probabilistic verification proves to be an effective method for reducing the amount of corrupted content and restricting its spread. While we proved that probabilistic integrity checks could reduce the impact of spam in the network, the nodes did not have a role in determining the strength of the measures taken against malicious nodes. The probability of checking an entry $P_{\text{check}}$ was a fixed network-wide parameter, chosen at the beginning of each experiment. As a result, nodes had to do the same amount of work regardless of the conditions in their surroundings (being flooded with corrupted messages or not).
In this chapter, nodes are given the autonomy to apply probabilistic verification on an individual basis, in effect transforming our collection of nodes into a self-policing network. By making nodes self-aware with respect to malicious behavior in the network (in particular, corrupt messages being disseminated), small adjustments to local behavior can be made. The result of these adjustments is that security measures are applied where needed, while nodes in safe areas keep their work to a minimum (but always maintaining a watchful eye).

Discouraging malicious behavior in the network involves two steps: a) executing probabilistic verification and b) updating the checking probability \( P_{check} \) for the next round. \( P_{check} \) is a local parameter and its value is updated according to the observations made during the probabilistic verification stage. The dynamic nature of ad hoc networks makes it necessary to continuously adjust the value of \( P_{check} \).

### 5.2.3. Verifying the Integrity of Data in a Dynamic Environment

We define the level of pollution in a collection of messages as the fraction of corrupted messages found in the collection. Nodes get an insight into the pollution levels in their neighborhood during the probabilistic verification phase. When participating in a gossip exchange, a node \( P \) receives \( s \) entries from a neighbor \( Q \). These entries are subject to integrity checks with a probability \( P_{check} \), resulting in a fraction of the \( s \) entries being checked. Since the neighbor selected the \( s \) entries at random from its cache, this sample gives node \( P \) an estimate of the level of pollution in the neighbor’s cache. Node \( P \) can then use this information to update \( P_{check} \) for the next round in the following manner:

\[
P_{check_{t+1}} = (1 - \alpha)P_{check_t} + \alpha P'
\]

\[
P' = \frac{\text{numRemoved}}{\text{numChecked}}
\]

\( P' \) is the level of pollution calculated after checking \( \text{numChecked} \) items in the probabilistic verification phase. \( \text{numRemoved} \) is the number of items that did not pass the integrity test and were removed. The value of \( P_{check} \) for the next round \( P_{check_{t+1}} \) is updated as a weighted sum of its previous value \( P_{check_t} \) and the level of pollution \( P' \). The parameter \( \alpha \) determines the sensitivity of parameter \( P_{check} \) to changes in the pollution levels in the neighborhood.

In general, nodes are bound to have more than one neighbor. For this reason, each node should maintain a different \( P_{check} \) for each neighbor. In essence, for each neighbor \( i \) node \( P \) maintains a \( P_{check_i} \). As described in the previous chapter, some security measures are in place before tackling the problem of data integrity (see 4.3). The measures taken to impose access control (discussed in
4.3.1) are of particular interest now, as the identity of each neighbor plays a role in the mechanism we describe. By requiring that nodes be accredited by a certification authority and that they include their identity certificate (signed by the certification authority) when gossiping, any node can verify the identities of its neighbors. In this way, the use of identity certificates protects the network against malicious nodes sending messages under many different identities (as the use of unauthorized identities would deem the messages invalid).

5.2.4. Experimental Setup

All nodes in the network start gossiping with no knowledge of their environment besides the identity of their immediate neighbors. Since nodes have no preconceptions about their neighbors, they start gossiping with little caution. This means that they apply a low level of checking at $t = 0$. For the experiments presented in the upcoming sections, $P_{\text{check}_{i,j}}[t] = P_{\text{check}_{\text{min}}} = 0.05$ for all nodes and all neighbors $i$. $P_{\text{check}_{\text{min}}}$ is also the lower bound for $P_{\text{check}}$. A lower bound for $P_{\text{check}}$ is necessary, since some checking is needed to monitor any changes in the behavior of neighbors. The implications of this are that there is a minimum workload imposed on the network, even in the absence of malicious nodes, and that there is a reaction time upon appearance of malicious nodes during which $P_{\text{check}}$ does not match the amount of spam being received. The disadvantage of this approach is that nodes can be caught off-guard by a spammer.

The damage caused by the sudden appearance of a spammer will undoubtedly depend on the level of checking applied by its neighbors at the moment of the attack. We argue that being overly cautious when starting to gossip with a new neighbor will not necessarily protect a node from being spammed, as a spammer could easily act like a normal node for a few rounds to gain a node’s trust before starting its attack. For this reason, we prefer to start with and maintain a minimum level of checking until malicious behavior is detected. As we will see later on, the introduction of a detection mechanism for spammers will deter attacks at high spamming rates, reducing the damage that a spammer can cause by catching its neighbors off-guard (before they adjust their checking probabilities).

In our experiment, nodes are arranged in a square grid topology, with 50 nodes on each side, over an area of $50 \times 50$ units. Each of the 2500 nodes has a range of 1 unit, making communication possible with its immediate neighbors to the North, South, East and West. From this collection of nodes, 250 are selected to be malicious at the beginning of each experiment. The selection is random, resulting in malicious nodes being placed at random locations in the grid. For all experiments, nodes have a cache size of $c = 100$ and during each gossip exchange they exchange $s = 50$ entries.
5.3. MALICIOUS VS. BENIGN BEHAVIOR

We define malicious behavior as the execution of a variation of the gossip protocol with the intent of gaining an unfair advantage in the use of a shared resource. In our system, the shared resource is storage space. By deviating from the data exchange rules defined in the shuffle protocol that the majority of nodes are executing, a malicious node can increase its share of storage space. The method used by a malicious node to place large quantities of its own content in the network is compromising the data integrity of the messages it forwards.

5.3.1. Attack Model

In order to test the effectiveness of the proposed method for enforcing data integrity, we assume that a relatively small number of nodes in the network are malicious. These malicious nodes, which we also call spammers, are randomly placed in the network and execute a slightly different version of the shuffle protocol. Their basic attack model is to corrupt entries before forwarding them to the nodes they communicate with. A spammer may deviate from the normal behavior of the general population in the following ways:

- Corrupt outgoing entries (in selectItemsToSend()) with a probability \( P_{\text{spam}} \) (also referred to as spamming rate).
- Fail to execute any integrity checks (in selectItemsToKeep()).

Although intuitively the most effective attack appears to be spamming with \( P_{\text{spam}} \) equal to 1, we will show later that a high spamming rate is actually counterproductive. In Section 5.4, we illustrate through simulation results that the only way for a malicious node to place more spam in the network is by spamming less (i.e., “behaving better”).

5.3.2. Can Malicious Nodes Be Identified?

This section describes the expected composition of the cache of a node that properly applies our spam removal algorithm and how that could help differentiate well-behaved nodes from malicious ones.

Cache Contents of a Well-behaved Node

The probability of having a corrupted entry after the probabilistic verification phase, \( P_{\text{spam in cache}} \), is directly related to the probability with which a node’s neighbor forwards spam to the node. For a node \( Q \) with a neighbor \( i \) that forwards a corrupted entry with a probability \( P_i \), the probability of a corrupted entry
Figure 5.1: If a node applies probabilistic verification properly, the amount of spam and checked items in its cache should be similar.

making its way into Q’s cache can be expressed in terms of $P_i$ and the probability of Q checking an entry received from a neighbor $i$, $P_{\text{check}}[i]$:

$$P_{\text{spam in cache}}[i] = P_i (1 - P_{\text{check}}[i])$$ (5.1)

In a similar way, we can determine the probability of Q marking an entry as checked and placing it into its cache, $P_{\text{checked in cache}}$, by calculating the probability that an entry received by Q is selected to be checked and is not corrupted. The probability of this occurring is:

$$P_{\text{checked in cache}}[i] = P_{\text{check}}[i] (1 - P_i)$$ (5.2)

Assuming that node Q is executing the probabilistic verification properly, $P_{\text{check}}[i]$ should approximate $P_i$, the probability that a received entry is corrupted. Therefore, we can expect that when dealing with neighbor $i$ the percentage of spam sent by $i$ that makes it into Q’s cache roughly approximates the percentage of entries marked as checked by Q and placed in Q’s cache. This comes as a result of (5.1) and (5.2) being approximately the same when $P_{\text{check}}[i] \approx P_i$.

As a general case, node Q has many neighbors and each neighbor may forward a different amount of spam. For example, neighbor A may be malicious and send many corrupted messages, while B may pass along a few corrupted messages sporadically. Nevertheless, the interaction with each neighbor should result in similar amounts of spam and checked entries arriving into Q’s cache. Therefore, neighbor A may be responsible for a large number of corrupted and checked entries in Q’s cache, while neighbor B, which rarely forwards spam, is responsible only for a few corrupted and checked entries. As a result, if node Q is properly filtering the content received from its neighbors, Q’s cache should have similar amounts of corrupted and checked entries. Figure 5.1 illustrates this scenario.
Expected Incoming Traffic

When exchanging entries with a malicious node, spam may come in two forms: as checked entries or as unchecked entries. Marking a corrupted entry as checked is a big risk as it effectively proves that the neighbor is indeed misbehaving. If a node runs an integrity check on an entry marked as checked by a neighbor and the test fails, the neighbor identifies itself as a spammer. Therefore, it is unlikely that a malicious node would send spam marked as checked. We can expect corrupted entries to arrive as unchecked.

The shuffle protocol specifies that entries to shuffle are selected at random from a node’s cache. Therefore, given that a well-behaved node has a similar amount of spam and checked entries in its cache, we can reasonably expect to receive a similar amount of corrupted entries and checked entries when engaging in a gossip exchange with a well-behaved neighbor. A significant difference between the two should raise concerns. The spammer detection mechanism (introduced later on) is built on this principle.

5.4. EFFECT OF SPAMMERS ON THE NETWORK

This section shows the extent of the damage caused by spammers in terms of the amount of spam they can place in the network.

5.4.1. The Effect of Alpha

The parameter $\alpha$, introduced in Section 5.2.3, determines the sensitivity of the integrity enforcement mechanism to the current levels of pollution observed. In essence, $\alpha \in [0, 1]$ controls the speed with which $P_{\text{check}}[i]$ adjusts to the pollution levels observed in the link corresponding to neighbor $i$.

Figure 5.2 shows the effect of $\alpha$ on the proliferation of corrupted entries through the network. For this experiment, 250 malicious nodes start corrupting entries with a probability $P_{\text{spam}} = 0.50$ at round 50. It is important to point out that the parameter $\alpha$ has no effect on the final amount of spam in the network. It only affects the speed at which the level of spam stabilizes. The level to which spam converges is dictated by the number of malicious nodes and the rate at which they place spam in the network.

While it might appear from Figure 5.2 that setting $\alpha$ to its highest value is the best policy, doing so would effectively mean that a node would adjust the checking probability for its neighbor solely based on their previous interaction. This strategy would be quick to react to the appearance of a spammer, but at the same time it would be quick to forget previous spamming by a neighbor. A
spammer could then easily reset the $P_{\text{check}}$ used by its neighbor by simply not sending spam in one round, opening the doors for more serious spamming in the next round. Remembering previous behavior of a neighbor (by having $\alpha < 1$) makes this kind of attack less harmful.

### 5.4.2. Varying Spamming Rates

Whenever a node exchanges entries with a malicious node, the malicious node has the opportunity to send spam. The amount of spam included in the collection of entries sent by the malicious node is regulated by the parameter $P_{\text{spam}}$, which is the probability that an entry sent by a spammer is corrupted.

Figure 5.3 shows the amount of corrupted content in the network over time, after the appearance of spammers. The results of six independent experiments, each with a different spamming rate, are shown. In each experiment, 250 spammers (10% of the network) at random positions start generating spam at round 50. Prior to that moment, all nodes in the network are checking the traffic in each of their links at the minimum level of $P_{\text{checkmin}} = 0.05$. The appearance of spammers is followed by a fast increase in the amount of spam in the network. However, in all cases the amount of spam stabilizes after the initial period of growth. An important observation is that the value to which the amount of spam converges is not proportional to the spamming rate of the malicious nodes. In fact, a closer look reveals that spamming at a high rate is actually detrimental to the spammer’s ability to place corrupted entries in the network. By measuring the value to which
the amount of spam converges (by averaging the last 200 rounds), we can observe its relationship to the spamming rate $P_{spam}$ more clearly (see Figure 5.4). Starting from a low spamming rate, we observe that initial increases of $P_{spam}$ yield positive results for the malicious nodes. However, after reaching the mid-point of $P_{spam} = 0.5$, increasing the spamming rate turns counterproductive.

After an initial period of adjustment following the appearance of spammers, well-behaved nodes tune their filters to the actual amounts of spam observed through each of the links to their neighbors. Consequently, neighbors of spammers create a barrier that filters out corrupt messages produced by malicious nodes. The strength of the filtering is proportional to the amount of corrupted entries observed. This explains the inability of spammers to disseminate corrupt data when using a very high spamming rate. In essence, the large amounts of spam produced are being filtered out after a hop or two.

In contrast, a somewhat lower spamming rate results in neighbors lowering their checking, allowing more spam into the network. The key to understanding this behavior lies in equation (5.1). In the stable state, where nodes have adjusted their filters ($P_{check}[i] = P_i$), the percentage of spam in a well-behaved node’s cache is dictated by the value of $P_{check}[i]$, peaking when $P_{check}[i] = 0.5$ and reaching 0 at the extremes ($P_{check}[i] = 0$ and $P_{check}[i] = 1$).
5.4.3. Workload Caused By Spammers

The previous sections have established that nodes regulate the number of checks they perform according to the amount of corrupted content they observe. Since the nodes’ wireless transceivers have a limited range, nodes that are in physical proximity of spammers are more likely to receive corrupt entries and, therefore, execute more integrity checks. This causes some nodes to have a higher workload than others with regard to data integrity enforcement.

The amount of checking done by a node is dictated by the values of $P_{\text{check}}$ on its incoming links. Using the average value of $P_{\text{check}}$ per node as a metric, Figure 5.5 shows the imbalance in the workload placed on the nodes in the network. While the majority of nodes has a low average $P_{\text{check}}$ (with a high number doing only the minimum amount of checking, $P_{\text{check}_{\text{min}}} = 0.05$), a considerable number checks more than a quarter of their incoming traffic. Even a few nodes check up to 50% or more of the entries they receive. These nodes are directly affected by having malicious nodes as neighbors. Even though their work prevents malicious nodes from disseminating corrupted entries, the malicious nodes succeed at disrupting the network by wearing out their neighbors by increasing their workload. For this reason, it is not enough to monitor and reduce the amount of corrupted entries, but also to take active measures towards isolating misbehaving nodes.
5.5. DETECTING MALICIOUS BEHAVIOR

Identifying a neighbor as a spammer boils down to being able to differentiate between a node that actively corrupts entries and one that simply forwards the corrupted content received from somebody else. Since nodes do not check 100% of the entries they receive from a neighbor, they cannot assume that receiving spam from their neighbor makes the neighbor a spammer. In fact, forwarding some spam is a perfectly valid situation in our network. In a similar way, the number of checked entries by itself is not enough to determine if a neighbor is a spammer or not. In fact, nodes doing minimal checking will forward very few checked entries.

5.5.1. Detection Mechanism

The key to detecting a spammer lies not in the amount of spam or the number of checked entries received, but in the relationship between these two values. As explained previously in Section 5.3.2, if a node is behaving properly, there should be a balance between the amount of spam and the number of checked entries it sends. With this in mind, a node can monitor the behavior of its neighbors by individually tracking the difference between the amount of spam received and the number of checked entries for each neighbor.

In a similar way as keeping track of the value of $P_{\text{check}}$ for every neighbor,
a node $Q$ can keep track of the number of checked entries it receives from each neighbor $i$. Node $Q$ does not do any additional checking, it just counts the number of entries flagged as checked. After every exchange with neighbor $i$, node $Q$ updates its estimate of the number of checked entries received from $i$:

$$\text{checked}[i]_{t+1} = (1 - \alpha)\text{checked}[i]_t + \alpha \text{fractionChecked}$$

The fraction of checked entries for the next round ($\text{checked}[i]_{t+1}$) is updated as a weighted sum of its previous value ($\text{checked}[i]_t$) and the percentage of checked entries found ($\text{fractionChecked}$).

The spammer detection mechanism is based on monitoring the difference between $P_{\text{check}}[i]$ (which is an approximation of the amount of spam received from neighbor $i$) and $\text{checked}[i]$ (which reflects the fraction of checked entries expected from neighbor $i$). An acceptable difference is defined as the parameter $\delta$. In every round, node $Q$:

- Calculates $\text{diff} = |P_{\text{check}}[i] - \text{checked}[i]|$
- If $\text{diff} \leq \delta$, neighbor $i$ is behaving properly. Otherwise, $i$ is a suspected spammer.

Figure 5.6 shows the result of applying the proposed detection method for different values of $\delta$, with $P_{\text{spam}}$ set to 0.5 (the value that allows for the most spam to be placed in the network, as shown in Figure 5.4). The graphs depict the number of spammers detected, as well as the number of false negatives (spammers that avoid detection) and positives (well-behaved nodes that are confused with spammers). The results are counted per link, since nodes do separate analyses for each of their neighbors. The threshold $\delta$ affects the detection of spammers in the following ways:

- A small $\delta$ results in spammers being detected quickly after their appearance, but could also lead to well-behaved nodes being mistakenly identified as spammers, i.e., false positives [see Figure 5.6(a)].

- With larger values of $\delta$, spammers can operate for a longer period of time before being identified. However, a larger $\delta$ prevents well-behaved nodes from being confused with spammers, i.e., false negatives.

The initial period after the appearance of spammers is characterized by well-behaved nodes struggling to adjust their values of $P_{\text{check}}$ for every neighbor to the appropriate level. As a result, good nodes may store and forward more corrupted entries than expected and could easily be confused with spammers if the threshold $\delta$ is too restrictive.
Figure 5.6: Results of applying the detection algorithm over time for different values of $\delta$: a) $\delta = 0.05$, b) $\delta = 0.10$, c) $\delta = 0.15$ and d) $\delta = 0.20$.

Figure 5.7: Spammers try to prevent being discovered by executing integrity checks: a) average value of $\text{diff} \pm \sigma$ for spammers and normal nodes as neighbors ($P_{\text{spam}} = 0.5$, $\delta = 0.20$), b) spammers that avoid detection.
5.5.2. Flying under the Radar

We say that a malicious node is “flying under the radar” if the system consistently detects it as a false negative. As seen previously, after a short initial period where nodes adjust to the presence of spammers, spammers can be accurately discovered given an appropriate value of $\delta$. In order for a malicious node to fly under the radar, it needs to modify its behavior enough to be confused with a well-behaved node.

Simple Strategies to Avoid Detection

We identify two possible strategies for a malicious node to avoid detection:

- Checking entries just as a normal node would do and spamming with probability $P_{spam}$ in the unchecked entries.
- Reducing $P_{spam}$ in the hopes of reducing $\text{diff}$ enough so that $\text{diff} \leq \delta$.

Figure 5.7 shows the results of the first approach. The experiment records statistical information for $\text{diff}$ (average value and standard deviation) when the neighbor is a spammer and when the neighbor is a normal node. By executing integrity checks just like a well-behaved node (now selectItemsToKeep() is the same for spammers and normal nodes), spammers can lower the value of $\text{diff}$ [see Figure 5.7(a)] to the point where some nodes avoid detection [Figure 5.7(b)], using $\delta = 0.20$. However, only a very small number of nodes are not discovered as being malicious. Moreover, this does not happen consistently.

The second approach is more effective in terms of avoiding discovery. With a spamming rate of 0.1, the value of $\text{diff}$ for a considerable number of spammers falls below the threshold $\delta = 0.20$, as can be seen in Figure 5.8(a). As a consequence, many spammers are not discovered as such [see Figure 5.8(b)]. However, at $P_{spam} = 0.1$ the amount of spam they can place in the network is low. And after discovered spammers are removed, the amount of spam will decrease even more. In addition, the spammers that are not discovered are not always the same. Therefore, a well-behaved node could identify spammers by taking as a policy to stop communication with another node if the node qualifies as a spammer $x$ number of times over a period of time.

A Clever Attack: Carefully Controlled Output

Reducing $P_{spam}$ appears to be a natural way of avoiding detection. With a sufficiently small spamming rate, the value of $\text{diff}$ falls below the $\delta$ threshold. However, when the value of $P_{spam}$ is low, the amount of spam that a malicious node can place in the network is also limited.
Figure 5.8: Spammers try to avoid detection by lowering their spamming rate to 0.1.

In an effort to place as much spam as possible while avoiding detection, a spammer may intentionally send checked entries along with the corrupted entries. The reasoning behind this behavior is that if neighbors observe as much spam as checked entries coming from the spammer, they will not suspect that their neighbor is malicious. For example, a spammer may send 50% of spam in every gossip exchange and fill in the remaining 50% with checked entries. In our experiments, spammers accomplish this by intentionally checking all the entries they send to achieve a $50\% - 50\%$ balance of spam and checked entries.

By behaving in this manner, spammers are able to avoid detection and at the same time place almost the same amount of spam in the network as if they simply spammed with $P_{\text{spam}} = 0.50$ (see Figure 5.9). This type of attack takes advantage of the fact that in our solution we suspect node $i$ based on the relation between $P_{\text{check}}[i]$ and $\text{checked}[i]$ and do not take into account the actual value of $P_{\text{check}}[i]$. In other words, a malicious node could send as much as 50% of spam without being detected as long as it also sends just as much checked entries. Referring back to section 5.3.2, we find that there is a limit to the amount of spam that we can expect from a well-behaved node. Given that when a node executes probabilistic verification properly $P_{\text{check}}$, should approximate $P_i$, we can derive from equation 5.1 that the amount of spam in a well-behaved node’s cache should not exceed the maximum value of $x(1-x)$, with $x$ in $[0,1]$. Consequently, a well-behaved node should not send more than 25% of spam among its entries. If we incorporate this observation into our detection mechanism, malicious nodes would be forced to lower the level of the spam they send to 25% at most. This results in a decrease in the amount of spam in the network as can be seen in Figure 5.9.

Unfortunately, there seems to be no obvious method to discipline the malicious nodes even more. By sending 25% of spam in every exchange along with 25% of checked entries, the spammer can simulate the behavior of a well-behaved...
node that is struggling with malicious neighbors. Being more strict about the admissible values of $P_{\text{check}}$ could result in the increase of false accusations.

5.6. CONCLUSIONS

In this chapter, we explored the feasibility of ensuring data integrity in very large ad hoc networks by means of a simple and inexpensive solution based on probabilistic integrity checks and traffic analysis. Our approach has proven to be effective in containing the spread of corrupted content without imposing a burden for all nodes in the network. In fact, the workload placed on nodes is proportional to the amount of corrupted content they receive, affecting mostly nodes in the neighborhood of spammers. By keeping track of their incoming traffic, nodes affected by malicious neighbors can discover and isolate those misbehaving nodes.

We also explored to what extent a malicious node can avoid detection. Although possible, avoiding detection implies a considerable decrease in the level of malicious behavior, lessening the impact of spammers in the network. By sending a combination of 25% of spam along with 25% of checked entries in every exchange, a malicious node can fool our detection mechanism. Nevertheless, this situation is not as dire as it appears, as the damage caused by the spammer is purely local. To illustrate, we can calculate (using equation 5.1) that after shuffling with a spammer that sends 25% of spam the probability that a corrupted entry makes its way into a given node’s cache is 18.75%. However, the node is bound to have

![Figure 5.9: Amount of corrupted entries in the network over time for different malicious behaviors.](image-url)
other neighbors besides the spammer, diluting the impact that a spammer has on the node. Moreover, this filtering is repeated at every hop, relegating the spam problem to the surrounding area of the spammer and, thus, allowing the rest of the network to keep operating with a reduced burden.
CHAPTER 6

Broadcast-based Epidemics

Nodes in wireless ad hoc networks are often limited in terms of resources, such as storage, power, and bandwidth. A downside of this is the fact that local storage at one node cannot accommodate the vast amount of data contained in the network. In this chapter, we present SharedState, a scheme for storage, replication, and distribution of common-interest data in wireless networks of resource-constrained devices (e.g., sensor nodes or embedded devices). Unlike the shuffle protocol, which represents a peer-to-peer style of communication, SharedState uses broadcast-based communication to achieve the same effect of epidemic dissemination. SharedState takes advantage of the fact that wireless communication is broadcast-based in nature. Messages are not explicitly addressed to a particular node, like in the shuffle protocol. Instead, when a node sends a message, the message is received by all nodes within radio range that are listening to the channel. Our aim is to replicate the desirable properties observed in a peer-to-peer gossip protocol like the shuffle, which is based on data exchange, while exploiting the broadcast nature of radio communication. In other words, we aim to benefit from the fact that messages are delivered to all nodes within radio range. The downside of this approach is the uncertainty of whether an item that has been broadcast will be received and kept by a neighbor.

As a consequence of relying on broadcast, the incoming traffic is greater than the amount of data that a node can broadcast. This is the fundamental difference between the shuffle protocol and SharedState: while a shuffling node sends as many items as it receives from its gossip partner, SharedState nodes receive many more items as they listen to all of their neighbors. By selecting a random neighbor to gossip with, shuffling nodes control their incoming traffic. SharedState nodes accept all incoming traffic and apply randomness in the selection of items to store and forward. The result, as this chapter presents, is epidemic dissemination of data items through the network in the same fashion as the shuffle protocol.
SharedState works under the assumption that individual nodes would greatly benefit from having access to the wealth of information in the network, but are unable to store it locally at once. SharedState strives to make data available to every node by providing local access to a subset of the whole collection of data items in the network at any moment in time and ensuring that this subset is updated periodically. This is accomplished by probabilistic propagation and replication of data items, ensuring the availability and persistence of information in the face of changing network conditions. We evaluate the performance of SharedState by studying the effectiveness with which nodes can gather information from the network. In addition, we optimize the bandwidth usage of our proposed solution by minimizing unnecessary communication based on feedback from the local neighborhood.

6.1. INTRODUCTION

The usual paradigm for wireless networks consists of wireless nodes - often laptops or smartphones - connecting to a base station in order to access a resource (for example, a data repository or a local printer). Wireless ad hoc networks break away from this model by focusing on the interaction between nodes to create a network on-the-fly, without relying on a preexisting infrastructure. The ad hoc model shuns the centralized approach in favor of operating in a distributed fashion. The resources and services are therefore provided by the nodes themselves, making cooperation between nodes absolutely necessary.

Nodes in a wireless ad hoc network are often mobile, portable, resource-constrained devices. In this chapter, we focus on resource-constrained embedded devices designed with specific applications in mind (for example, a wireless sensor network that monitors the presence of people in a building). Unlike phones or PDAs with wireless capabilities, these devices typically use low power RF radios that provide limited bandwidth and communicate through broadcast (Section 6.3 describes the target platform for SharedState in more detail). The networks that these devices create do not rely on a fixed infrastructure for services, but they self-organize to provide certain functionality.

One of the main challenges with wireless devices is that they are inherently unreliable, as they might fall out of reach due to mobility or leave the network unexpectedly. As a result, nodes - and the data they carry - are constantly joining and leaving the network. While most of the data that a node stores locally may only be relevant to the node itself, some nodes might have information that could be of interest to the community in general. Such information may include, for example, configuration information, advertisements or general announcements. Viewing
these pieces of information as community knowledge and making them available to the general population would enhance coordination efforts in the network and create a cohesive environment.

The goal of this project is to provide a middleware layer capable of storing data items published by any node in the network and make them available for all nodes. In essence, SharedState acts as a distributed repository of shared data. Unlike a publish/subscribe system, nodes do not subscribe to receive certain information. In our system, all nodes are considered to be possible subscribers. The purpose is then not just to deliver a data item to the interested parties, but to store the data item in the network so that any interested node could retrieve the item presently or in the future. In other words, the ultimate goal of SharedState is to ensure the availability and persistence of data items of interest to all (or most) nodes.

SharedState acts as a loosely coupled communication platform, such that producers and consumers of data are decoupled in time and space. Consumers can recover a data item from the network when they deem it necessary: producers and consumers do not need to be present at the same time in order to share data. Likewise, they can do so without being within communication range of each other. SharedState takes care of delivering the data items to consumers, who might be located anywhere in the network. Producers are oblivious to any consumer’s location.

**Contribution** Working under the assumption that the information contained in the network greatly surpasses the storage capacity at each node, this chapter presents SharedState, a scheme designed to move data items through the network and create replicas of the items at various locations. Specifically, the contributions of this chapter are:

- We introduce a scheme for disseminating and replicating data items through the network. This new protocol is characterized by its low complexity and minimal state needed at each node, making it suitable for a wide range of wireless devices.

- We evaluate the efficiency of collecting data items from the network by testing the worst case scenario: each node discovers new items solely by querying its local store. Additionally, we use static topologies in our experiments. The lack of mobility means that data items can propagate only through multiple hops, instead of being carried by mobile nodes to different locations. We show that acceptable discovery rates can be achieved even under these conditions, suggesting that mobility and queries involving surrounding nodes would only improve performance.
• We evaluate our protocol through simulations using TOSSIM. We test the effect of node density on the performance of the system, first using uniform topologies of various densities and then with a non-uniform topology where nodes concentrate at a central point. We show that by allowing individual nodes to decide when to communicate based on neighborhood information, we can take advantage of node density to decrease the number of transmissions by individual nodes. Global knowledge of the network topology and its properties is not required. By relying solely on local information, our algorithm remains effective in larger networks.

6.2. RELATED WORK

Like the shuffle protocol, the role SharedState plays in the software stack is similar to coordination mechanisms such as publish/subscribe (see 2.5) and shared tuple spaces (made popular by Linda [Gelernter 1985]). Like these schemes, SharedState provides a flexible model of interaction based on the decoupling in space and time of producers and consumers of data. SharedState has a more narrow focus as it is intended for small, resource-constrained devices. This puts SharedState in the same category as TeenyLIME [Costa et al. 2006], a Linda-like tuple space abstraction for wireless sensor networks and Hood [Whitehouse et al. 2004], a programming abstraction where nodes share their state with selected neighbors. A major difference with these schemes is that they aim to facilitate the development of wireless sensor network applications by providing a neighborhood abstraction and, therefore, share data only within the neighborhood. Abstract Regions [Welsh and Mainland 2004] expands the scope of sharing by enabling communication using tuple spaces within a region. Unlike these approaches, SharedState is not explicitly aimed a wireless sensor network and, therefore, does not assume that data is only relevant within a restricted space.

The problem of improving data access and availability in wireless environments has been approached in various ways. One approach is to encode [Dimakis et al. 2005; Chessa and Maestrini 2003] the data into a number of pieces in such a way that the original item can be reconstructed by collecting a subset of the pieces. By distributing the pieces through the network, ubiquitous access is provided. In our case, we assume that the data items we propagate are small read-only data files and we achieve availability by replicating the items. The number of replicas and their location is determined probabilistically.

Cooperative caching for ad hoc networks is another related area. Its aim is to share cached data among multiple nodes by having some nodes host the data and handle requests from other interested nodes [Sailhan and Issarny 2003; Yin
and Cao 2006]. Popular data items are cached at various locations leading to resources being saved by requesting the item from a nearby node. Additionally, access can be obtained even when the original source is unavailable. While cooperative caching also makes use of data replication, it differs from our work in that its aim is to improve the experience of being connected to the infrastructure (Internet). Conversely, SharedState focuses on sharing lightweight data items created by nodes in the network that can be disseminated in the background using a limited amount of resources. Instead of a request/forward model, nodes using SharedState discover data items by periodically exchanging them.

Projects focused on data dissemination for ad hoc networks are also relevant to our work. 7DS [Papadopouli and Schulzrinne 2001] focuses on allowing access to data available on the Internet, so that when a node’s access fails it can get the data from its peers. The types of networks 7DS addresses are different from ours. The authors consider that the network is rarely connected (sparse) and that nodes do not necessarily cooperate. The nodes themselves are more powerful than the ones we consider. While 7DS does implement policies for power management, storage space is not a major concern, as it is for us. In PeopleNet [Motani et al. 2005], users forward data to pre-defined geographic regions (according to topic) and within each region the system tries to match queries and responses. The nodes in each region become a database for a particular topic, with items being replicated in many nodes. Within each region, data dissemination/replication occurs in a p2p fashion, whenever two devices encounter each other. In RANDI [Wolfson et al. 2007], nodes also communicate when they encounter each other, but also proactively if a certain amount of time has passed since the last broadcast. These two systems rely on p2p communication and neighborhood discovery/awareness. SharedState intends to be as lightweight as possible, relying solely on broadcast. There is no need to keep track of the identities of neighboring nodes.

In the realm of wireless sensor networks, data-centric storage (DCS [Shenker et al. 2003]) addresses the storage problem by storing data by type at designated nodes, making data retrieval more efficient. Replication of data at strategic locations has been proposed to improve scalability and robustness [Ghose et al. 2003]. Unlike our work, these approaches require a routing layer and replication is done in a deterministic fashion.

Closer to our work are probabilistic protocols for data dissemination, where the decision to broadcast a piece of data is made locally based on a probabilistic algorithm. Due to their simplicity, these types of protocols are appealing for small devices lacking in computing power. They are also resilient to failures and mobility, which makes them attractive for wireless environments. Probabilistic protocols have been used as an alternative to flooding [Haas et al. 2002; Drabkin et al. 2007] and for concrete applications like code dissemination [Levis et al.
6.3. SYSTEM MODEL

The system we envision consists of a collection of nodes with wireless communication capabilities. Participating nodes are required to contribute resources to the system in the form of storage space. Every node contributes a limited amount of storage space to maintain a local data store of shared information. Nodes access their local data stores to discover previously unseen or interesting items. We will refer to the local data store as the node’s cache in the remainder of the chapter.

The caches are updated periodically with data items broadcast by nodes in the local neighborhood. Items have unique ids and are time stamped when created, allowing the system to keep the latest version of an item by overwriting older versions. Since all data exchanges occur within one hop, routing is not necessary. By relying purely on broadcast, we intend to make the system suitable for a wide range of wireless platforms, including simple and inexpensive devices that use broadcast at the physical layer. Moreover, nodes do not need to keep track of their neighbors.

Communication is limited to periodic updates that are broadcast by each node. Each update message contains a set of data items selected by each node. The frequency with which nodes can broadcast updates is a network-wide parameter, which should be set considering the workload and bandwidth that we desire to allocate for the service.

In addition to the cache, nodes allocate space for an input buffer to receive update messages from neighboring nodes and an output buffer for the items to be broadcast. Each node uses the input buffer (which should be, at most, as big as the cache) to accumulate data items received during a fixed period of time, which we call a round. At the end of a round, the node updates its cache with the items from the input buffer and broadcasts the set of data items in its output buffer to update its neighbors. We say that a node alternates between two modes: active or passive. Each node takes an active role once per round, when it updates its local cache and decides which items to broadcast. After taking care of these tasks, it falls into a passive mode, where it silently awaits for updates from its neighbors.

We abstract a framework to describe the core structure of a replication and storage protocol like SharedState. There are three main operations (see Figure 6.1) that a node needs to execute: a) handle incoming items (passive mode), b) update its cache (active mode) and c) select which items to broadcast (active mode). The specific way in which these three events are implemented has a direct impact on the characteristics of the propagation and replication of items. In Section 6.4
we describe the implementation details of SharedState, which is one particular instance of this framework.

**Target Platform** The experimental platform for which SharedState was specifically developed consists of very simple, inexpensive units that integrate a radio, an antenna and an embedded processor in one module. These nodes are meant to be expendable and, as such, they have modest features. Nodes operate on a fixed duty cycle and communication is based purely on the broadcast of small data packets of a fixed size (in the order of tens of bytes). In other words, nodes wake up periodically to communicate and process information and then go to sleep for the rest of the cycle.

The reason for choosing to operate by broadcasting/processing periodically is to have a predictable use of resources, enabling us to tailor the duty cycle and packet size according to the requirements of our applications and the desired lifetime of the network. We imagine that a sensor application (sending an alarm whenever high temperatures are measured, for example) needs to be long-lived and has very small data packets. For this application, the broadcast interval can be set to a value that allows the batteries to last for the desired period (two years, for example). By operating periodically, we eliminate the risk of having nodes run out of energy prematurely due to being located at busy spots, as can occur in event-triggered systems. Of course, this comes at a price. The tradeoff is that nodes are required to communicate even when there are no new events to report. This is an acceptable compromise, considering that we aim to deploy very large networks where having clear expectations of the lifetime of nodes is important.

**Applications** Application areas for SharedState include dissemination of topological information, membership management and service discovery. An example
application could be asset management, where active tags attached to objects keep track of each other so that logical groups stay together (e.g., a set of boxes in a warehouse or a collection of documents).

6.4. SharedState

The key to ensuring the availability and persistence of items in the network lies in a strategy of massive replication and relocation of replicas. The replication of items is a natural consequence of the probabilistic methods used for the selection of items to be stored and propagated. As items are propagated, they become available to the nodes who stored them locally. The periodic update of caches ensures that nodes can discover items as they flow through. Discovery is gradual, however, as nodes can store only a limited number of items in their caches.

As we will see later on, we evaluate the performance of Shared State by measuring how fast a node can discover the items that are available in the network. We consider the most simple way in which a node can discover items, which is by looking into its own cache (0-hop query). Nevertheless, if improved data access is required, a node can resort to enlisting the help of neighboring nodes. Having its neighbors search into their own caches effectively casts a wider net from which to retrieve items. We can apply the same reasoning we used in 2.4.2 for finding the probability of seeing an item in a node’s cache after \( k \) trials to the case where we look for an item in \( k \) different caches. To illustrate, imagine that a given item can be found in a given node’s cache with a probability of 5%. If a node requests the help of 4 neighbors to find the item, the probability of success increases to \( 1 - (1 - 0.05)^5 = 22.6\% \). Considering that discovery of items takes place over several rounds, it becomes apparent that requesting the help of neighbors in discovering items is highly beneficial. Nevertheless, we demonstrate that a node can discover the items available in the network over time even when searching in a single (i.e., its own) cache.

The issue of data persistence is critical in a wireless ad hoc network, since nodes may come and go on a regular basis. Whenever a node leaves, the data items it carries disappear with it. For this reason, maintaining a set of replicas per item is necessary. We refer to each replica of an item as an entry. While a data item is a piece of information, an entry is the representation of the data item in the network and for each data item several entries may exist. Instead of explicitly trying to maintain a particular number of entries per item, we allow competition between entries to determine the number of entries per item in the network.

Whenever a node broadcasts an entry, there is a chance that it might be replicated if more than one of the node’s neighbors decides to keep it. Likewise, when-
ever a node updates its cache, some entries are discarded due to lack of space. Because of this, the number of entries per item is constantly experiencing variations and, since there is no preference for any particular data item, competition for space in a node’s cache is fair. This ensures that each data item has on average the same number of replicas in the network and that the number of replicas adjusts dynamically according to the number of different items published. That is, when there is a large number of different items in the network, each item has few entries. Conversely, when there are few different items present, each item has several entries.

6.4.1. The Protocol

The way nodes manage their entries depends on the actions they take in their active and passive modes. Figure 6.2 gives a detailed account of the steps involved in the execution of a node’s active and passive thread. Before giving a more thorough explanation of the events that take place in each thread, one distinction between nodes should be noted. Of all the nodes that participate in the system, only a subset acts as a source of data items. That is, at any point in time, only some nodes take the role of producers of information. The only difference between a producer and a consumer is the fact that the producer makes an effort to insert its own item (represented in Figure 6.2 as localEntry) in the network whenever possible. Other than that, they execute the same algorithm. Moreover, at any point a consumer may take the role of producer if it has some information to add to the collective knowledge base.

6.4.2. Active Thread

Each node in the network executes the active thread once per round. The algorithm executed by the active thread is divided in two phases: a) updating the cache and b) selecting which entries to broadcast.

Phase I - Update Cache: In this phase, the node has to decide what to do with the entries accumulated in the input buffer. The result should be an updated cache with as little correlation as possible to the previous one. The reason for this is that applications that access the cache have already seen the entries in the previous version. Showing the same entries again does not provide any value for the application layer. In the majority of cases, the cache will already be full forcing the node to decide which entries from the input buffer should be placed in the cache and which entries from the cache should be removed.

The strategy for updating a node’s cache is the following. First, if an entry has already been seen (it is in the cache) and also appears in the input buffer, it is discarded from both, making space for new entries in the cache. This strategy
/** Active thread **/
// Runs every T time units

1: PHASE I: Update cache
2: for all entry in inputBuffer do
3:   if cache.contains(entry) then
4:     cache.remove(entry)
5:     inputBuffer.remove(entry)
6:   while cache.slotsAvailable() < inputBuffer.size() do
7:     randomEntry = cache.removeRandomEntry()
8:     if outputBuffer.slotsAvailable() then
9:       outputBuffer.add(randomEntry)
10:   cache.addAll(inputBuffer)
11:  inputBuffer.clear()
12:

13: PHASE II: Select entries to broadcast
14:   if outputBuffer.slotsAvailable() then
15:     if !outputBuffer.contains(localEntry) then
16:       outputBuffer.add(localEntry)
17:   while outputBuffer.slotsAvailable() do
18:     randomEntry = cache.copyOfRandomEntry()
19:     if !outputBuffer.contains(randomEntry) then
20:       outputBuffer.add(randomEntry)
21:
22:  broadcast(outputBuffer)
23:  outputBuffer.clear()

/** Passive thread **/
// Runs when receiving a transmission

1: for all received entries do
2:   if inputBuffer.slotsAvailable() then
3:     if inputBuffer.contains(entry) then
4:       inputBuffer.keepMostRecent(entry)
5:     else
6:       inputBuffer.add(entry)

Figure 6.2: SharedState pseudocode.
might seem at odds with the behavior of the shuffle (which we are trying to emu-
late), where a node always keeps the entries received from its gossiping peer. The
difference is that, in a broadcast-based protocol like SharedState, when a node se-
lects an entry to be sent, it is received by all its neighbors. To emulate the shuffle, a
p2p-style protocol, only one of the neighbors should keep the entry in its cache. In
SharedState, we try to achieve a similar effect by allowing nodes to drop a newly
received entry that they have already seen. The assumption being that even if a
node drops the entry, another node in the neighborhood might keep it, resulting in
a shuffle-like behavior.

The second part of the strategy for updating the cache is that all remaining
entries in the input buffer should be placed in the cache. If there are not enough
empty slots available in the cache, random entries from the cache are removed to
make space for all the entries from the input buffer. The entries removed from the
cache are placed in the output buffer until it reaches its maximum capacity. The
ones that do not fit into the output buffer are discarded.

Phase II - Select Entries to Broadcast: In Phase I, some entries were already
placed in the output buffer. These entries, having been removed from the cache
in Phase I, have preference to be broadcast. The motivation for this is to lower
the risk of items disappearing entirely from the network. If there are not enough
entries to fill the output buffer, the local entry (which is available if the node is
a producer) is added. If this is not enough, random entries are selected from the
cache and a copy of each is placed in the output buffer. Once the selection of
entries has finalized, the node broadcasts the chosen entries and clears the output
buffer for the next round.

6.4.3. Passive Thread

Each node executes the passive thread whenever a broadcast is received. There-
fore, the passive thread may execute several times in one round (depending on
the number of neighbors a node has). Whenever a node receives a broadcast, the
entries received are put into the input buffer. No duplicates (entries with the same
id) are allowed. If a duplicate is received, the version with the freshest timestamp
is kept. The input buffer has a limited capacity, therefore, entries have to be dis-
carded once the buffer is full.

6.4.4. Early Strategies

In 6.3, we introduced the framework on which SharedState is built. During the
design phase, several strategies were explored for each of the three main opera-
tions defined in the framework. As a way to motivate the design choices made in
SharedState, we present a summary of the different approaches that were explored on the way to its final implementation.

**Insertion of items**

In the shuffle protocol, an item only needs to be inserted once to start its replication through the network. This is derived from the fact that when two nodes shuffle, items cannot be lost. At most, an item might lose one of its replicas. In a broadcast-based protocol with limited storage space, however, there are no guarantees that an item that is broadcast will be kept by the nodes that receive it. Since nodes receive more items than what they are allowed to broadcast within one round, they are inevitably forced to discard some items. Early on in the development of the protocol, we used the same strategy for inserting items as in the shuffle: insert the new item once and let it replicate. Because of the possibility of the new item being dropped from the network soon after being published, the protocol would sometimes fail to spread the new item. For this reason, we decided to modify the publishing strategy so that a publisher can reinsert its local entry (i.e., the item it wants to disseminate) whenever there is space in the output buffer (see Active Thread, Phase II.)

**Update the cache directly**

In the early stages of design, each node had only a cache (that is, there was no input buffer). Two of the main operations, handling incoming items and updating the cache, were merged into one step where the cache was updated directly with the incoming items whenever a broadcast was received. The update strategy was simple, if the item is not already in the cache, place it in an empty slot in the cache or a randomly chosen slot if the cache is full. The downside of this approach is that items that arrive when the cache is full have to overwrite an existing item. Consequently, items that were just received could be overwritten before getting the chance to be broadcast further, which has a negative effect on dissemination speed. To avoid this situation (or make it less likely to happen), the input buffer was introduced as a place to collect all incoming items received during a round. In this way, the cache can be updated once per round with the items from the input buffer. By having a clear differentiation between new (in the input buffer) and previously seen (in the cache) items, better choices can be made with respect to the new contents of the cache.
Random selection of items to broadcast

The initial strategy for selecting which items to broadcast was to pick them randomly from the cache. Although selecting them randomly from the cache is not a bad strategy, we quickly realized that we could improve performance by taking a cue from the shuffle protocol. A principle of the shuffle is that only items that are sent can possibly be overwritten in the cache. We incorporate this idea into SharedState by favoring the items that have been displaced from the cache when filling the output buffer. By doing so, the items that are broadcast lose their spot in the cache just like in the shuffle, but are essentially given a chance to be replicated and kept alive in the network.

Removal of duplicates

This strategy is inherited from the shuffle protocol and was motivated by the fact that the application running on top of the dissemination layer has access only to the cache and there is no added value in presenting the same item twice. We included this strategy when updating the input buffer, as it has limited space which should be used optimally. Early on, the same idea was applied when updating the cache with the items from the input buffer. To be more precise, if an item was in the cache and also in the input buffer, one copy was kept in the cache, just like it would occur in the shuffle when an item received from a neighbor is already in the cache. However, keeping these items in the cache would result in the cache being almost always full, which in turn caused us to randomly overwrite items in the cache. To avoid this situation, we now remove items from the cache when they appear both in the input buffer and the cache, clearing slots in the cache for other items that were not in the cache in the previous round.

6.5. BASIC PROPERTIES

When executed in a large scale over a period of time, the SharedState protocol presents certain characteristic behavior. In this section, we take a look at this behavior in terms of dissemination speed and replication of items.

6.5.1. Discovery Rate

The ultimate goal of SharedState is to make data available to all nodes by storing it in the network. Since nodes themselves do not have enough storage space to store all of the available data items, they can “discover” data items from the network when required by the application layer. Discovering items can be done by
inspecting the local cache (0-hop query), consulting immediate neighbors (1-hop query) or recruiting the help of neighbors to inspect caches \( n \)-hops away (\( n \)-hop query). In this work, we consider only 0-hop queries.

We evaluate the performance of the protocol by observing the discovery rate of items. The discovery rate is defined as the number of items that a node discovers by examining its cache over a period of time versus the total number of items in the network. The discovery of items is gradual, as nodes update their caches once per round. The discovery rate, therefore, is measured over a number of rounds and with every passing round it increases or stays the same.

The speed at which nodes discover items is directly related to the fraction of all items that they can store locally. In other words, for a collection of nodes with a cache size of \( c \) and \( d \) different items in the network, the fraction \( \frac{c}{d} \) determines the discovery rate. Figure 6.3 presents the discovery rate over time for three different experiments. 900 nodes, each with a cache size of \( c = 18 \), input buffer size of 18, and output buffer size of 9, were arranged in a \( 30 \times 30 \) grid. The nodes can reach only their neighbors to the North, South, East and West. After the network has been running for 300 rounds, a number of test nodes start measuring their discovery rates. The graphs show the average discovery rate and standard deviation. For each experiment, a different number of items in the network \( d \) was used \( (d = 180, 360, 720) \). A higher value of \( d \) means that a node can store a smaller percentage of the \( d \) items locally. As Figure 6.3 illustrates, the discovery rate slows down when the fraction of items that a node can store in its cache decreases.

Figure 6.3: Discovery rate for different rates of cache size \( c \) versus number of items \( d \).
Figure 6.4: Histogram of the number of replicas per item in a network of 900 nodes with 180 items (cache size = 18).

Figure 6.5: Number of replicas for one item published at round 300 in a network of 900 nodes with 180 items (cache size = 18).
6.5.2. Fairness in Replication

Given that our network has a fixed storage capacity (the sum of the space available at each node), the number of replicas that an item can have is limited. Items have to compete for the limited space in a node’s cache. When there is a large number of items in the network, this competition is intense. When there are few items, the competition is less fierce. Nevertheless, the protocol does not favor any particular item, resulting in every item having the same chance of creating or losing a replica.

Figure 6.4 presents a histogram of the number of replicas per item averaged over a period of 50 rounds (after a start-up period of 300 rounds where items are published and nodes populate their caches). The results correspond to the experiment in Figure 6.3 where $d = 180$. Our 900-node network, with $c = 18$, has $900 \times 18$ available slots. With 180 different items in the network, if the storage space is divided evenly, each item should have 90 replicas. Figure 6.4 shows how many items are replicated a given number of times. From the distribution, we see that there is a tendency for items to have 90 replicas (the mean value of the distribution).

Due to the constant competition for space, the number of replicas for a particular item is always fluctuating. Figure 6.5 shows this fluctuation with a new experiment where an item is published after a start-up period of 300 rounds. By the time the new item is published, the nodes in the network have been broadcasting items for 300 rounds already and their caches are full. The new item has to compete for space with the 180 items that were published earlier. As can be seen, at every round the new item experiences gains and losses in its total number of replicas, but manages to eventually create just as many replicas as the other items.

6.6. COMPARISON TO THE SHUFFLE PROTOCOL

The original motivation for the development of SharedState was to have a broadcast-based protocol capable of replicating and disseminating items in the same way as the shuffle protocol, but without the need for point-to-point communication. With this in mind, a comparison of the behavior of both protocols is in order. Having just discussed the basic properties of SharedState, in this section we compare the performance of the shuffle protocol under the same conditions and discuss the similarities in the results.

In order to compare SharedState and the shuffle, we need to find a common ground in the execution of both protocols. Each execution of the shuffle protocol requires that a node contact a randomly chosen neighbor and that the neighbor send its reply. If we count each of these interactions as one broadcast, we find that in each round of the shuffle protocol the number of broadcasts executed is twice
the number of nodes. Therefore, as a way of having a fair comparison between protocols in terms of executed broadcasts, we take one round of shuffling to be the equivalent of two rounds of the SharedState protocol.

Figure 6.6.b) reproduces the discovery rate experiments from Section 6.5 with nodes executing the shuffle protocol. To facilitate the comparison, Figure 6.3 is repeated as Figure 6.6.a) next to the shuffle results. Note that Figure 6.6.b) shows only 200 rounds, half the number of rounds presented in Figure 6.6.a). The first observation that we can make is that both figures show essentially the same behavior: the discovery rate slows down as the number of different items in the network increases. The fundamental differences between the protocols are reflected in the slight variations in their performance. SharedState is more sensitive than the shuffle to the increase in number of different data items, as can be clearly seen in the experiment with the most different items in the network \((c/d = 0.025)\). It is also evident from comparing both figures that the discovery rate curves, which are the average from the discovery rate measurements of 50 test nodes, show a smaller standard deviation for the shuffle.

For the purpose of understanding the reason for the observed differences, we reproduce the replication experiment from the previous section using the shuffle protocol. Figure 6.7 shows the replication of an item published at round 300 in a network where 180 previously published items are already occupying the network. Figure 6.5 is repeated as Figure 6.7.a) for comparison with the shuffle results displayed in Figure 6.7.b). While both figures eventually converge to a somewhat stable number of replicas (around 90), the major difference between the figures is the change in the number of replicas from one round to the next.

When two nodes shuffle items cannot be lost, as demonstrated in the modelling analysis in Chapter 3. At most, a node will drop an item that has been sent to a neighbor. However, the neighbor will store it, which ensures that the item is not
lost after the exchange. On the other hand, losing items is a very real possibility in SharedState. Even though when a node broadcasts an item it is received by all of its neighbors, there is a possibility that none of the neighbors will keep the item for the next round. Conversely, broadcasting an item to several neighbors can result in the item being replicated more quickly that in the shuffle. As a result, we see more dramatic variations in the number of replicas for the SharedState protocol as opposed to the shuffle. We speculate that the fluctuating behavior with regards to replication of SharedState affects the discovery rate at the test nodes in the Figure 6.6.b) resulting in the larger standard deviation observed. Nevertheless, we can conclude that SharedState does retain the main properties of the shuffle protocol.

6.7. DENSITY AWARENESS

An important characteristic of wireless networks is that neighborhoods are defined by the proximity between nodes. If a given area is densely populated, one node’s broadcast will be overheard by a large number of nodes. On the flipside, if the area is sparsely populated, the broadcast will be received just by a few nodes in the sender’s range.

The protocol introduced in Section 6.4 does not take density information into account. Nodes blindly broadcast update messages every round, regardless of whether their neighbors would be able to handle the traffic or not. While this may not be a problem in sparse networks, in densely populated areas excessive communication could be detrimental due to collisions. It should also be noted that the size of the input buffer limits the number of updates that a node can effectively make use of in one round. Once a node’s input buffer is full, the subsequent...
updates have no effect on the outcome. With this in mind, we propose a slight modification of the original SharedState protocol to optimize the use of bandwidth by reducing the amount of ineffective communication.

The key to reducing the number of ineffective broadcasts is identifying when a node becomes “overloaded” by transmissions from its neighbors. If that is the case, the node cannot derive any benefit from receiving more broadcasts. It would be desirable, then, to decrease the chances that its neighbors send more broadcasts in the remainder of the round. This can be accomplished if nodes inform their neighbors of their “overload level,” defined as the fraction of ineffective broadcasts received in one round. Nodes can use the overload levels of their neighbors to decide if they should broadcast in the next round or not.

We say that a node is overloaded if its input buffer is already full when it receives a broadcast. Since several broadcasts are often received in one round, each node can calculate its own overload level by keeping track of the number of broadcasts received in a round and of how many of those were received when the input buffer was already full. Let \( R_i(x) \) represent the sources of broadcasts received by node \( x \) during round \( i \) and let \( U_i(x) \) represent the number of ineffective broadcasts. The calculation of the overload level is based on the observations made during the previous round and takes place in the active thread, once per round, in the following way:

\[
O_i(x) = \frac{U_{i-1}(x)}{|R_{i-1}(x)|}
\]

Ideally, a node’s overload level should be close to 0, indicating that the node rarely receives ineffective broadcasts. However, the node itself does not have direct control over its own overload level. The local overload level is determined by the behavior of the node’s neighbors. For this reason, we propose an improved version of the protocol where nodes are required to append their own overload level when broadcasting a message. Each node can then accumulate these reported overload levels to get a sense of the overall overload level in its neighborhood. With this information, each node can determine if it should skip a broadcast based on whether the broadcast would benefit its neighbors or not. Let \( ProbSkip_i(x) \) be the probability used by node \( x \) to decide whether to skip a broadcast or not at round \( i \). \( ProbSkip \) is calculated in the active thread as follows:

\[
ProbSkip_i(x) = \left[ \frac{\sum_{y \in R_{i-1}(x)} O_{i-1}(y)}{|R_{i-1}(x)|} + O_i(x) \right] + 1
\]

The probability of skipping a broadcast, \( ProbSkip \), is an estimate of the overload level in node \( x \)’s neighborhood calculated based on the overload levels reported by \( x \)’s neighbors (\( y \)) that communicated in the previous round and \( x \)’s own
overload level. It is important to note that this estimate may not always be very precise. The reason for this is that if a node decides not to broadcast, its neighbors will not be updated on the node’s overload level. Therefore, the calculation of the overload level in the neighborhood is done with only partial information. The inclusion of the node’s own overload level helps make up for the missing reports of some neighbors. In any case, as will be shown later on, an estimate - even if it is not very precise - is good enough to result in considerable resource savings.

Before moving on, it should be noted that since each node can calculate its overload level locally, an alternative optimization could be proposed where \( \text{ProbSkip} \) is simply calculated based on the local overload level (under the assumption that it accurately reflects the overload levels in the neighborhood). While this strategy would work in homogeneous topologies where nodes have roughly the same number of neighbors, it does not perform as well in more complex scenarios. For example, take a situation where a node is surrounded by obstacles and as a result only has one neighbor. The neighbor, however, is surrounded by many other nodes and is often overloaded. In this case, the first node will never be overloaded and will always broadcast, contributing to the overload of its only neighbor. The second node, meanwhile, experiences higher overload levels, meaning that it will skip some rounds. The results are detrimental to both nodes: the first receives broadcasts only sporadically, as its only neighbor tends to skip broadcasts, while the second node becomes even more overloaded. Under the scheme that we proposed earlier, the first node would be aware that its neighbor is often overloaded and would skip some rounds to relieve its neighbor’s load. At the same time, the second node measures less overload in its neighborhood and would tend to broadcast more often, benefitting the first node. Through experimentation, we observed that the method proposed earlier performs better overall. We compare its performance against the original SharedState in the remainder of the chapter.

6.8. PERFORMANCE EVALUATION

We evaluate the effectiveness of the SharedState system by observing i) the discovery rate of items (as defined in Section 6.5) and ii) the number of broadcasts generated. Section 6.7 introduced a modified version of the original algorithm aimed at reducing resource consumption by letting nodes skip broadcasts according to the overload levels in their neighborhoods. In order to quantify the improvements introduced by the modified algorithm, we gathered statistics (over a test interval of 50 rounds) on: a) \( \text{Broadcasts sent} \) in the whole network per round and b) \( \text{Broadcasts received} \) by a node per round. These statistics give us some insight into the usage of the communication medium and the workload of the nodes,
both of which we aim to reduce.

We tested our storage system by implementing it as a TinyOS application and using TOSSIM to run experiments under different scenarios. In order to observe the effect of varying node densities in the performance of the protocol, various topologies with different node densities were used. The topologies were generated with the *LinkLayerModel* tool that comes bundled with TinyOS. This tool generates network topologies using a theoretical propagation model that takes parameters to describe the channel (which affects how the signal propagates), the radio (which determines link asymmetry due to noise) and the topology (how nodes are physically positioned). Based on these parameters, the tool generates the link gain and the noise between any pair of nodes in the network. The TOSSIM radio model is signal-strength based and takes these link gain and noise values to determine the connectivity of the network. Within a 100×100 meter terrain, we explored the following simulation scenarios:

**Uniform Node Distribution** For the “uniform” setting, the physical terrain is divided into a number of cells (based on the number of nodes) and a node is randomly placed within each cell. Since the tool requires that the number of nodes be a square, we generated topologies with the following numbers of nodes: 100, 144, 196, …900, or \((10 + 2n)^2\) for \(n = 0…10\).

**“Center of attraction” Distribution** This scenario emulates a more realistic situation where nodes gather around a point of interest. We crafted a topology with 576 nodes where the highest concentration of nodes occurs at the center of the terrain (50,50). The position of the nodes was determined by selecting a random angle between 0 and 360 degrees and a distance from the center according to a normal distribution (with a mean of 0 and a standard deviation of 30; we use the absolute value). The minimum distance between nodes was set to 1 meter.

Besides the distribution of nodes, system-wide parameters and the role of nodes had to be defined. For all experiments, the following settings were used:

- Of all nodes in the network, 80 nodes were chosen at random to be publishers. These 80 nodes produce 80 items that are disseminated through the network.
- 20 nodes selected at random are chosen as “test nodes.” These nodes measure their discovery rates and numbers of broadcasts sent and received.
- For all nodes, the cache and the input buffer can hold only 8 entries. The output buffer can hold 4 entries.
- All nodes execute the active thread once per second.
Figure 6.8: Discovery rate over time for networks of different sizes: a) sparse topologies, b) dense topologies.

6.9. UNIFORM NODE DISTRIBUTION

In this section, we will focus on the behavior of the system when the collection of participating nodes is spread uniformly over an area of 100×100 meters.

6.9.1. Discovery Rate and Node Density

Figure 6.8 shows the discovery rate measured over time for the different topologies, starting from a sparse 100-node network and increasing in density up to 900 nodes in the same 100×100 meter terrain. For clarity, we present the sparse topologies (from 100 nodes to 400 nodes) on left and the dense topologies (484 nodes to 900 nodes) on the right. Notice how the increase in density leads to higher discovery rates, most notably for the sparse topologies in Figure 6.8.a). It can be clearly observed that for a sparse network of 100 nodes the discovery rate after 50 rounds is quite low (about 11%) and does not improve substantially over time. This is due to the low connectivity between nodes. With only 100 nodes in the 100×100 meter terrain, the network is too sparse. However, even a slight increase in density (144 nodes) already yields much better results.

To better understand the discovery rate results, it is necessary to have more insight into the connectivity of the different topologies. Figure 6.9 shows the average number of transmissions received by a node in one round for the different topologies used in Figure 6.8. A linear relationship between the number of nodes in the network and the number of received transmissions can be distinctly observed. Taking a closer look at Figure 6.9, we can see why the discovery rate for the 100-node network was so low. With an average number of receptions per node of 1.6, it is not uncommon for nodes to be unreachable or for the network to become partitioned at times. Due to the nodes being sparsely located, the LinkLay-
erModel tool generates a topology where the link gain between neighboring nodes is low. As a result, links in the 100-node network are weak and nodes may not always receive messages from their neighbors. At the other end of the spectrum, we have the 900-node network with an average of 19.23 transmissions received per node in one round, indicating stronger links between neighbors.

The linear increase in receptions with the number of nodes does not translate to a linear increase in performance (measured by the discovery rate), as shown in Figure 6.8. After a considerable improvement in discovery rate when going from the sparse 100-node network to the more populated 144, 196 and 256-node networks, further increases in density fail to have a substantial effect in the discovery rate (as shown in Figure 6.8.b)). The reason for this can be traced back to the limited size of the input buffer. With the input buffer being twice the size of the output buffer for our experiments, the first transmission received fills up half of the input buffer. The subsequent transmissions gradually fill up the rest. Since the input buffer becomes full after receiving a few transmissions, the higher number of transmissions received in the denser topologies do not provide much additional benefit.

We can model the way the input buffer fills up under the assumptions that: a) each entry received is selected randomly from the collection of \( d \) different items in the network, b) each transmission from a neighbor consists of \( s \) randomly selected entries and c) the input buffer is infinite. Let \( n_k \) represent the number of entries in the input buffer after \( k \) transmissions.

For \( k = 1 \), all entries are kept, resulting in \( n_1 = s \). For the second transmission,
some of the $s$ received entries might already exist in the input buffer. Therefore, $n_2$ increases only by $s \cdot \left(1 - \frac{n_2}{d}\right)$ entries, where $\frac{n_2}{d}$ is the probability that an entry is already in the input buffer. Likewise, $n_3$ increases by $s \cdot \left(1 - \frac{n_2}{d}\right)$ entries over $n_2$.

A general expression for $n_k$ can be derived in terms of $s$, $d$ and $n_{k-1}$:

$$n_k = n_{k-1} + s \cdot \left(1 - \frac{n_{k-1}}{d}\right), n_0 = 0$$

The second term of the equation represents the increase in entries for each additional transmission. Since the probability of receiving an entry that is already in the input buffer, $\frac{n_2}{d}$, increases as the input buffer accumulates more entries, the second term becomes smaller with every new transmission.

Using the same parameters as our TOSSIM simulations (number of items $d = 80$ and output buffer size $s = 4$), $n_k$ is plotted in Figure 6.10. The graph illustrates how later transmissions have less impact on filling up the input buffer by comparing $n_k$ with a curve depicting a linear increase in number of entries with each transmission received. We can also observe from this graph that, under this model, our input buffer (which can hold only 8 entries) would be full by the third transmission. This helps to explain why the 100-node network underperforms: the input buffers are rarely used to their full capacity.

On the other hand, the 256-node network, where the average number of broadcasts received is 4.8 with a standard deviation of 1.5, makes full use of the input buffers and performs considerably better. Notice, however, how the experiments with more than 256 nodes show slightly better discovery rates. We speculate that
this is due to the fact that when nodes have more neighbors, the $s$ entries broadcast by each node are more likely to be truly random selections from the $d$ possible items in the network. For more sparse networks where nodes have only a few neighbors, the entries at neighboring nodes are more likely to be correlated, slowing down the discovery of new items.

### 6.9.2. Taking Advantage of Node Density

The SharedState protocol works as expected, but suffers from a common problem in wireless networks: unnecessary transmissions. Section 6.7 introduced a modified version of the protocol aimed at optimizing the use of bandwidth by taking node density into account. Knowing that the input buffers have a limited capacity, it is evident that at some point additional transmissions are not effective anymore and resources are wasted on them. In this section, we show that our modified algorithm can reduce the waste of resources while still delivering good performance.

We start by comparing the discovery rate over time for the original algorithm and the density-aware version. Figure 6.11 shows the results for two selected topologies: a sparse (256 nodes) and a dense (784 nodes) one. The performance is virtually the same in the sparse topology. For the dense topology, the discovery rate is slower during the initial phase of the experiment. Nevertheless, it recovers and matches the original algorithm in the later stage of the experiment.

Given that performance has not been compromised by the changes introduced to the original algorithm, we proceed to study the effect the changes have on the workload of the network. First, we measure the average number of broadcasts received per node during one round and compare the results to the original ones. Figure 6.12 presents the new measurements alongside the results shown in Figure 6.9, clearly showing the reduction in the number of broadcasts received per
node. As a result, the nodes have fewer transmissions to handle in each round. This is not surprising, given that in the density-aware version of the algorithm nodes refrain from broadcasting based on the overload levels measured in their neighborhoods. It can be expected, then, that nodes in denser topologies would experience higher overload levels and, therefore be more likely to skip broadcasts.

Figure 6.13 showcases the reduction in the number of broadcasts sent per round in the whole network. Figure 6.13.a) highlights the impact of density in the decrease of broadcasts being sent. As the topologies become more dense, and the number of neighbors per node increases, nodes are more likely to be overloaded increasing the probability of broadcasts being skipped. The companion table to the right gives a more detailed account of the reduction in broadcasts,
with the column titled “Broadcast Ratio” referring to the ratio of the number of broadcasts sent in one round using the density-aware protocol versus the number of broadcasts using the original version. While the difference is minimal in the sparse 100-node network, as the networks become more dense, it is clear that the new algorithm allows the nodes to save transmissions by reducing the number of ineffective broadcasts.

6.10. CENTER OF ATTRACTION

In this section, we observe the behavior of our system using a more realistic node distribution where nodes (576 in total) are arranged around a central point. While the uniform node distributions used previously may approximate the layout of a sensor network (in a field, for example), we think that this scenario resembles more closely a social event, such as an outdoor barbecue, where people gather around a central location (a bonfire, for example). The physical distribution of the nodes is shown in Figure 6.14.

We start by comparing the original algorithm and the modified version using the new centralized topology. In terms of discovery rate, the results vary slightly (see Figure 6.15), with the original version outperforming the modified version in the initial rounds of the experiment. After that initial period, both versions perform similarly, with the modified version gaining an edge over the original protocol towards the end of the experiment.
Figure 6.15: Comparing the original algorithm and the improved version in terms of discovery rate.

Figure 6.16: Distribution of nodes according to their probability of skipping a round.
Given that the density-aware version of the protocol behaves as expected, we proceed to analyze the resource usage in the network. For this experiment, values of $\text{ProbSkip}$ for every node were collected over a 400-round run. The histogram in Figure 6.16 shows the percentage of nodes using a value of $\text{ProbSkip}$ that falls within a certain range. It is clear that the graph is skewed towards high values of $\text{ProbSkip}$, indicating that most nodes skip some rounds. In fact, a large majority skips more than 50% of broadcasts. It should be noted, though, that a few nodes (roughly 3%) do not skip any broadcasts. These are the nodes in the outer regions of the terrain, which have only a few neighbors and need to take advantage of every broadcast.

The relationship between the distance from the center and the probability of skipping a round becomes evident in Figure 6.17. During a period of 50 rounds, every node reported its $\text{ProbSkip}$ and distance from the center. This graph plots each pair $(\text{distance}, \text{ProbSkip})$ as one point and clearly shows a trend where nodes closer to the center report lower values for $\text{ProbSkip}$.

### 6.11. CONCLUSIONS

In this chapter, we have demonstrated that it is possible to build an effective shared-storage solution based purely on probabilistic methods. Moreover, we have shown that by taking into account only local neighborhood information the use of resources, namely bandwidth and energy, can be drastically reduced with-
out having a major impact on performance. We achieve this by allowing nodes to regulate their output to prevent their neighbors from being overloaded, which in turn benefits them by saving transmission costs. This results in resource savings according to the density of the area, without the need to explicitly disseminate topology information. We conclude that the combination of a probabilistic approach and local-only decision making is key to the scalability of systems such as SharedState.
CHAPTER 7

Conclusions

In this dissertation, we studied the use of epidemic techniques for information dissemination in the context of a wireless environment. While pursuing our goal of building protocols for wireless networks based on epidemic techniques and evaluating their properties, it became evident that modifying the steps of a protocol to obtain the desired properties is not a trivial task. In this chapter, we elaborate on the problems we encountered while trying to fine-tune our protocols to achieve certain behavior and the lessons we learned along the way. We conclude by outlining future research directions.

7.1. DISCUSSION

Epidemic techniques have successfully been applied in dynamic environments, specifically peer-to-peer networks, triggering our interest in studying their behavior in wireless networks. Wireless communication presents particular challenges in comparison to a wired network caused by the fact that placement of nodes is a determining factor for connectivity. Therefore, interactions within a neighborhood are crucial for achieving the desired large-scale behavior. Given that epidemic protocols rely solely on local interactions (that is, the execution of the protocol can be fully realized without contacting any nodes from outside the neighborhood), we find them ideally suited for a wireless environment. In this section, we discuss the challenges we have faced in the development of gossip-based protocols and what these challenges taught us about the nature of gossiping.

7.1.1. Local Interactions vs. Large-scale Behavior

Gossip protocols achieve their goals through repeated execution of simple routines. As long as there are enough gossiping nodes to maintain the network con-
NOTATION

A network of gossiping nodes displays certain large-scale behavior. While changes to the routines that describe the local interactions often result in changes in the large-scale behavior, the mapping between changes in small-scale interactions and their large-scale implications is not obvious. A small modification in a routine may result in very noticeable changes in the emergent behavior of the protocol when executed in a large network. For example, let’s take a look at the shuffle protocol. Figure 7.1.a) shows the discovery rate for 900 nodes \( c = 18 \) arranged in a grid and three experiments where the number of different items is \( d = 180, 360 \) and 720, respectively. This figure was presented earlier in Section 6.6. To show the impact of a small change in the large-scale behavior, Figure 7.1.b) reproduces the same set of experiments with a version of the shuffle, in which the method \texttt{selectItemsToKeep()} has been modified slightly. Like before, the first step is to remove repeated items (to avoid duplicates in the cache). The difference is that now, if there is not enough space in the cache to store the received entries, random entries from the cache are dropped. In the shuffle, a node is allowed to drop only those entries that have already been sent to a neighbor. This seemingly unimportant change has a big impact in the discovery rate, as can be seen in Figure 7.1. Looking only at the local interactions is not enough to predict the effect a modification to the protocol may have in the large-scale behavior of the network.

Not all changes to local interactions have a noticeable effect in the protocol’s performance. Chapter 6 offers a good example of this. In Section 6.7, we propose a modification to the original SharedState protocol such that nodes can decide to skip broadcasts based on neighborhood information. The number of broadcasts is reduced dramatically (in dense neighborhoods), yet performance is minimally affected. In this case, the fact that large-scale behavior is barely affected works to our advantage, as we have reduced the number of transmissions.
To summarize, the fact that gossip-based protocols execute simple routines does not necessarily mean that their large-scale behavior can be easily understood or manipulated. The mapping between the local interactions and the resulting large-scale behavior is not obvious. Throughout our work we have discovered that minimal changes in a protocol can have unexpected consequences for its large-scale behavior. For this reason, the modification of local interactions should be accompanied by large-scale experiments to verify that the desired emergent properties are retained or changed as expected.

### 7.1.2. The parameter space for gossip-based protocols

What makes a protocol “gossip” has been the subject of some debate. The definition of gossip itself when applied to networking is somewhat fuzzy. The literature provides several examples of protocols that define themselves as gossip, many of which are vastly different. Some are broadcast-based while others adhere to peer-to-peer interactions. Some execute periodically, while others are event-based. In any case, a defining characteristic of gossip-based protocols is an element of randomness in their routines.

We have explored two basic types of gossip protocols: one based on peer-to-peer interactions (the shuffle) and one based on broadcast communication (Shared-State). While the style of communication (request/reply vs. broadcast) differs between both protocols, they share many characteristics: periodic execution, limited storage space and fixed amount of data exchanged per round. With these constraints, we have reduced the number of possible parameters to tune in our protocols.

Identifying the parameters that affect system behavior is crucial, yet not trivial, as protocols may be sensitive to changes in some parameters more than others. Let’s look, for example, at the size of the local storage space (i.e., the cache). While our protocols benefit from having a larger cache, we can intuitively predict that at some point we would stop seeing the benefit of increasing the cache size. Once the cache size surpasses the number of different items being propagated, it is of no use to have a larger cache (as the cache can now store all available items).

The effects of tuning a parameter cannot always be easily predicted. Chapter 3 provides a good example of a parameter which, when adjusted, has a counterintuitive effect on performance. We are referring to the exchange buffer $s$ in the shuffle protocol. At first, we presumed that increasing the size of the exchange buffer would always improve performance. This is true for small sizes of the exchange buffer (in relation to the size of the cache). However, at some point further increases in the size of the exchange buffer fail to yield better performance. In fact, to our surprise, larger sizes of $s$ are counterproductive for the dissemination.

We find that, even when dealing with a reduced parameter space, the setting
of the parameters still remains an open question. Not only is the effect of a given parameter setting not easily predictable, there may be non-obvious dependencies between parameters. For the case of the shuffle protocol, we have discovered through probabilistic analysis that the exchange buffer has an optimal value, which depends on other system parameters. Probabilistic analysis has proven to be extremely helpful and enlightening for the understanding of parameter relationships for the shuffle protocol and, we surmise, that taking a similar approach for other gossip-based protocols could be fruitful.

7.1.3. Determinism and Probabilistic Protocols

Gossip protocols are characterized by having a component of randomness in their behavior. Be it in the selection of a peer to gossip with or in the handling of the information being gossiped, randomness plays an integral role in gossiping. However, pure random behavior tends to be wasteful, as it fails to take advantage of opportunities to achieve the goal at hand. What we have observed is that through a controlled use of randomness the beneficial properties of random behavior can be preserved, while performance is improved. In other words, it is important to take the time to pinpoint the appropriate places where randomness can be beneficial.

Although the protocols for information dissemination we consider are simple, there are several points at which decisions have to be taken. To rely on randomness at every step of the decision-making process would produce less than optimal results. Figure 7.1, presented earlier, gives a clear example of the negative effect of applying randomness in a thoughtless manner. The graphs illustrate how the extra care in selecting which items to keep in the shuffle protocol pays off significantly in terms of performance. In contrast, the modified version of the shuffle, which randomly drops items from the cache to make space for the items received, performs poorly. We should stress that the superior performance of the shuffle is not derived from increased complexity in the algorithm. In fact, the shuffle still uses randomness to select which items to drop. The only difference is that the items are randomly selected from a subset of the items in the cache, namely the ones that were selected to be sent to the gossiping neighbor.

Throughout the work presented in this dissertation, we have strived to find the balance between retaining the random component of gossipping and improving performance. For example, in the SharedState protocol, we introduced the input buffer to collect incoming items during one round before updating the cache. Early on, we tried updating the cache directly obtaining much poorer results. Having the input buffer in place gives us more control, as having more complete knowledge of the messages received in the previous round allows us to make better decisions when updating the cache. This comes at the cost of using extra storage space for the input buffer.
We find that introducing ways to impose control over the random behavior of the network can allow us to improve performance, yet often times there is a tradeoff. Chapters 4 and 5 help to illustrate this point. As described in Chapter 5, we can reduce the amount of spam in a gossiping network more effectively if we keep track of the amount of spam sent by each neighbor individually. While this is an improvement over checking a fixed amount of the incoming traffic (see Chapter 4), the need to maintain historic information about the behavior of neighbors in order to dynamically adjust the probability of executing checks opens the door for malicious nodes to try elaborate strategies to continue spamming without raising suspicion. For instance, they may choose irregular spamming patterns or constantly change neighborhoods. What we observe is that our attempt to improve performance (decrease spam) by exerting more control over the traffic in the neighborhood comes with the tradeoff of our solution being more fragile. That is, malicious nodes, knowing our anti-spam strategy, may find ways to get around it. On the other hand, the strategy of indiscriminately checking a certain portion of the messages (Chapter 4), while inefficient when there are no spammers, would have a guaranteed level of success.

7.2. CONCLUSIONS

Based on the preceding discussion, we give a summary of the conclusions derived from the work presented in this dissertation:

- When designing gossip-based protocols, the design choices are not obvious since there is no clear mapping between the local interactions (described by the protocol) and the behavior of the protocol observed at a large scale.

- By restricting our protocols to a particular framework, we have reduced the parameter space. Nevertheless, how to set the remaining parameters is still a challenge since the effect of parameter setting is not easily predictable.

- While random behavior is a key ingredient of gossip protocols, it is important to identify how and when to best apply it.

While we realize that these conclusions seem to suggest that the design of gossip protocols does not follow any strict methodology, we would like to outline the strategies that can be followed in the design of gossip protocols:

The incremental design (trial-and-error) approach - This strategy requires the evaluation of the performance of the protocol whenever changes are introduced to determine if the changes are an improvement. The same strategy
conclusions

The modelling approach - Modelling of the local interactions (as done for the Shuffle protocol in Chapter 3) requires a deep understanding of the mechanics of the protocol and, as a result, can help identify ways to improve the protocol. More importantly, modelling can clarify the relationship between local interactions and large-scale behavior and even help reduce the parameter space (by uncovering relationships between parameters).

We find that a good approach to developing gossip-based protocols falls somewhere between these two strategies. While modelling forces us to understand, evaluate and even reconsider certain design decisions, we are still required to look at the large-scale behavior of the protocol because that is where emergent behavior manifests itself. Standard modelling techniques, such as the use of model checkers, fail to help us when considering large networks, as they do not scale well. Traditional models of epidemics concentrate on characterizing the speed of infection, often using very simple gossip protocols or abstracting away the details of more complex protocols. While providing useful insight into the generic behavior of epidemics, these models are not detailed enough to capture subtle changes that we may introduce in our protocols.

In order to serve as tools for the understanding and development of gossip protocols, the models we develop should: a) capture the interactions between gossiping nodes at a level of abstraction that can allow us to evaluate the effect of system parameters on the behavior of the protocol and b) be able to handle large networks, in order to observe the emergent behavior of the protocol.

7.3. Future Directions

In this section, we outline possible research directions based on the work presented in this dissertation.

7.3.1. Further exploration of the parameter space

The constraints we have defined for our gossip-based protocols (i.e., periodic execution, limited storage space and fixed amount of data sent per round) have allowed us to concentrate on a limited set of parameters and their influence on the characteristics of the data dissemination through the network. It follows that upon the removal or modification of these constraints we would be faced with a richer
parameter space. Moreover, customizing our protocols with specific applications in mind will undoubtedly introduce new parameters.

The expansion of the parameter space calls for a reevaluation of the performance of our protocols, with an emphasis on discovering how the new parameters may impact the behavior we have come to expect from our protocols.

7.3.2. Incremental Design of Gossip Protocols aided by Models

We have shown that we can model a gossip exchange between two nodes and use this model to reproduce the dissemination of an item through the network and observe its properties. The development of the model gave us valuable insight into the mechanics of gossiping and let us see the relationship between system parameters. This, in turn, allowed us to determine the optimal value for a parameter (the exchange buffer $s$) as a function of the other system parameters.

In the case of the shuffle protocol, the theoretical analysis came after the protocol had been developed and its behavior studied through simulations. While developing the analytical model for the shuffle protocol, it became clear to us that modelling the local node interactions could help us understand the large-scale behavior we observed in earlier experiments. For this reason, a logical next step for our research could be to incorporate this theoretical analysis into the design phase of future gossip protocols. The aim of this design approach would be to use modelling to validate the inclusion of new policies into a protocol. In other words, we would start with a simple gossip-based protocol (and its model) and refine its behavior through the inclusion of policies (that should also be modelled). Policies would then be incorporated into the final version of the protocol depending on whether the model suggests that their large-scale behavior would be desirable.

7.3.3. Focus on Applications

The protocols we have presented are essentially application independent low-level services for information dissemination. By studying the properties of these protocols, unencumbered by application-specific optimizations, we have established a baseline of performance and expected behavior. We foresee that tailoring our protocols for particular application scenarios can only improve their performance.

Being probabilistic protocols, our gossip-based protocols are best suited for non-critical applications, where delays or occasional loss of messages can be tolerated. In that category, we find a variety of monitoring applications, notification services and asset management. Our current work in the area of ambient assisted living is intended to enhance elderly care by monitoring patients in non-intrusive ways using embedded technology [ALwEN 2008]. In this context, gossip protocols are used for robust communication within a network of sensor nodes and
monitoring nodes.

7.3.4. Field Experiments

Even though not included in this dissertation, we have conducted practical experiments with real wireless gossiping nodes to validate some of our observations from simulations [Mandemaker 2008]. To be more specific, we deployed networks of up to 100 nodes executing a simplified version of the SharedState protocol. Being our first experience with a real deployment, we used these experiments to measure the connectivity of the network and the coverage achieved by a newly-published data item. We observed that the pattern of dissemination of a new item shows similar characteristics to the dissemination observed in simulations.

The main challenges we faced with our field experiments were related to the connectivity of the network and the collection of performance data. Unlike simulations, where we are able to define which nodes can communicate, real deployments use unreliable radio links to connect the nodes. As a result, we only had a vague idea of our topology as we deployed the network. Case in point, while we arranged our nodes in a grid, the number of neighbors that each node received messages from varied greatly from node to node, with some nodes having more than eight neighbors while others had less than one on average. The reasons for this variance are hard to pinpoint. The condition of the ground the nodes were laid on (grass, in this case) or the quality of the air (very humid, most days) may be factors. External interference, such as cell phone usage or intruders (people and even cats) wandering through our deployment certainly affected connectivity. In any case, instead of being discouraging, the experiments have solidified our conviction that wireless networks are far from reliable or predictable and that gossiping as a communication paradigm provides the robustness necessary to overcome these issues. After the experiments finished, the time-consuming task of gathering the data from memory began. Due to the large number of nodes, having a wired backbone for data collection was not an option and since we did not want to disturb the experiment by having additional wireless communication, we opted for logging data in local memory (EEPROM) and manually recovering the data after the conclusion of the experiment.

Next steps in practical experimentation with nodes include gathering information about the connectivity of the network and deploying simple applications to run on top of our gossip protocols. We realize that most of our simulations use unrealistic topologies. For this reason, we are interested in collecting data about node connectivity in real deployments (e.g., asymmetric links, number of neighbors). We expect to use this information to generate more realistic simulations of our protocols and compare the results with actual performance measurements from a deployment. Regarding applications, the ongoing ALwEN project [AL-
wEN 2008] focuses on practical uses of embedded technology for assisted living. As part of this project, a number of simple monitoring applications running on top of a gossiping network are being developed.
SAMENVATTING

Epidemische Informatiedisseminatie in Grootschalige Draadloze Netwerken

Het voortschrijden van de technologie heeft het de afgelopen jaren de grootte en kosten van computers dramatisch doen afnemen, waardoor ze nu alomtegenwoordig zijn in de thuisomgeving en op kantoor. Bovendien heeft de voortdurende miniaturisatie van verwerkingseenheden geleid tot ingebedde systemen, die net zo krachtig zijn als personal computers van een aantal jaar geleden. Deze computersystemen voor specifieke doeleinden treft men nu overal aan, van mobiele telefoons tot keukenapparaten, en het is niet onredelijk om aan te nemen dat hun aantal de komende jaren zal stijgen.

Terwijl personal computers vaak met elkaar verbonden worden door middel van een vaste bedrade infrastructuur, gebruiken kleine rekenapparaten normaliter de ether als verbinding naar zo’n infrastructuur. Deze apparaten werken als een draadloze extensie van het bedrade netwerk. Om deze draadloze apparaten hun eigen autonome draadloze netwerk te laten vormen, moeten vele problemen opgelost worden. In deze dissertatie heb ik me gericht op hoe men effectief en efficiënt informatie kan verspreiden in volledig draadloze netwerken.

De karakteristieken van het draadloze medium (beperkt bereik van de radio’s, onbetrouwbare communicatie, dynamische topologieën) maken het gebruik van gecentraliseerde oplossingen doorgaans complex, tenzij men zich beperkt tot relatief kleine systemen. Hoewel gecentraliseerde oplossingen goed zouden kunnen werken op kleine schaal (zoals voor een groep gebruikers met laptops in een café)
mag verwacht worden dat de wijdverspreidheid van draadloze ingebouwde syste-
men kan leiden tot grootschalige draadloze netwerken met duizenden knopen.
Voor deze netwerken is ge-centraliseerd beheer al zel bal ve triv iaal. Grootschalige
draadloze netwerken vereisen algoritmen die volledig gedistribueerd zijn, zich anpassen aan veranderende omstandigheden, bestand zijn tegen het falen van
individuele elementen en gebaseerd zijn op lokale interacties tussen elementen.
Epidemische (of roddel) protocollen voldoen aan deze eisen.

De termen roddel en epidemisch worden over het algemeen door elkaar ge-
bruikt. Formeel zijn roddelprotocollen een subgroep van epidemische protocollen.
Analoog aan de verspreiding van geruchten in het echte leven geeft de term roddel
aan dat een gerucht wordt verspreid wanneer entiteiten interacteren. Deze
interactie gebeurt willekeurig en iedere keer dat het gerucht doorgegeven wordt
zal de ontvangende entiteit het met een bepaalde waarschijnlijkheid weer verder
verspreiden. Dit heeft als resultaat dat het gerucht snel verspreid wordt, maar zon-
der harde garanties dat het alle entiteiten zal bereiken. In de informatica refereert
een roddelprotocol over het algemeen aan een protocol met de volgende karakter-
istieken: willekeurige selectie van entiteiten om mee te interacteren, periodieke
uitvoering en symmetrie (waarmee bedoeld wordt dat alle knopen hetzelfde algo-
ritme uitvoeren).

We hebben twee basale typen van roddelprotocollen verkend: één gebaseerd
op peer-to-peer interactie (genaamd het Shuffle type) en één gebaseerd op broadcast communicatie (genaamd het SharedState type). Hoewel de stijl van commun-
icatie (verzoek/antwoord vs. stuur-naar-allen) verschilt tussen beide protocollen
delen zij vele karakteristieken: periodieke uitvoering, beperkte opslagcapaciteit en
de uitwisseling van een vaste hoeveelheid data. Voor beide protocollen hebben wij
een uitgebreide studie uitgevoerd naar de karakteristieken van de informatiever-
spreiding in grootschalige netwerken en hoe deze karakteristieken beïnvloed wor-
den door verschillende parameterwaarden en ontwerpkeuzes. Na uitgebreide sim-
ulaties en analyses van de resultaten kunnen we de hieruit getrokken lessen als
volgt samenvatten:

- Een netwerk van roddelende knopen vertoont na het herhaaldelijk uitvoeren
  van de roddelroutines een bepaald globaal gedrag. Hoewel veranderingen
  in de routines die de lokale interactie tussen knopen beschrijven vaak resul-
teren in veranderingen in het globale gedrag van het netwerk is de afbeeld-
ing van de lokale veranderingen naar hun globale implicaties niet voor de
  hand liggend.

- Het identificeren van de parameters die het systeemgedrag beïnvloeden is
  cruciaal, maar niet triviaal, omdat protocollen gevoeliger kunnen zijn voor
  veranderingen in bepaalde parameters dan in andere. Door onze proto-
collen te beperken tot een bepaald raamwerk hebben wij de parameterruimte beperkt. Maar zelfs met een gereduceerde parameterruimte is onze bevinding dat de keuze van de parameters nog steeds een open vraag blijft. Niet alleen is het effect van een gegeven parameterkeuze niet gemakkelijk te voorspellen, er kunnen ook niet voor de hand liggende afhankelijkheden bestaan tussen parameters.

- Alhoewel willekeur een cruciaal bestanddeel is van roddelprotollen is het belangrijk om te identificeren hoe en wanneer probabilistische keuzes het beste toegepast kan worden. In ons werk hebben we ernaar gestreefd om de balans te vinden tussen het behoud van willekeur bij roddelen en het verbeteren van de prestaties. Al zijn de protocollen voor informatiedisseminatie die wij beschouwen simpel, toch zijn er verscheidene punten waarop er beslissingen moeten worden genomen. Het gebruik van willekeur in elke stap van het beslissingsproces zou tot minder dan optimale resultaten leiden.

Op basis van deze observaties concluderen we dat hoewel roddelprotocollen heel eenvoudig te implementeren kunnen zijn, het bereiken van het gewenste globale gedrag is verre van simpel. Om de mechanismen van roddel waarlijk te begrijpen hebben we twee benaderingen gevolgd: (a) we hebben een variëteit aan parameter- en ontwerpkeuzes verkend, en (b) de interactie tussen roddelende knopen gemodellerd. De eerste benadering hield in dat wij de prestaties van het protocol moesten evalueren wanneer er wijzigingen gemaakt werden om te bepalen of deze wijzigingen tot een verbetering leiden. De tweede benadering dwong ons om roddelinteracties op een dieper niveau te begrijpen, en om bepaalde ontwerpbeslissingen te evalueren en zelfs te herzien. Een combinatie van beide benaderingen is cruciaal om te begrijpen hoe roddelprotocollen te ontwerpen en af te stellen zodat het gewenste globale gedrag bereikt wordt.
BIBLIOGRAPHY


ABOUT THE AUTHOR

Daniela Gavidia was born on May 6, 1978 in Lima, Peru. She attended the Peruvian University of Applied Sciences for her undergraduate studies and earned her bachelor’s degree in Electronic Engineering in 2000. After graduation, she started working as a web developer. After some months on the job, Daniela realized that she missed the intellectual challenge of being in an academic environment and decided to go to graduate school. While considering her options, Daniela came across an advertisement for NUFFIC, the Netherlands organization for international cooperation in higher education. Without missing a beat, she applied and a few months later was awarded a NUFFIC fellowship to pursue a Master’s degree in The Netherlands. She decided to try something different from her undergraduate studies. After two years, she earned a Master’s degree in Computational Science from the University of Amsterdam.

In 2003, Daniela joined the Distributed Systems group at the Vrije Universiteit as a Ph.D. candidate. Again, her research topic was a departure from what she had previously studied. In her third year, she took a break to go to Mountain View, California for an internship at Google. The experience was a refreshing change of pace and, after three months, Daniela returned to Amsterdam to continue with her research work with renewed energy.

Daniela is currently taking part in the ALwEN project, which aims to apply a combination of sensor, embedded and wireless technology to enhance healthcare for the growing elderly population. The practical nature of the project provides a unique opportunity for Daniela to experiment in the field with state-of-the-art technology and test many of the ideas presented in this dissertation.
ABOUT THE AUTHOR
List of Publications


