Epidemic-Based Self-Organization in Peer-to-Peer Systems

Spyros Voulgaris

VRIJE UNIVERSITEIT

EPIDEMIC-BASED SELF-ORGANIZATION IN PEER-TO-PEER SYSTEMS

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 3 oktober 2006 om 10.45 uur
in het auditorium van de universiteit,
De Boelelaan 1105

door

SPYRIDON VOULGARIS

geboren te Athene, Griekenland

promotoren: prof.dr.ir. M.R. van Steen
prof.dr. A.S. Tanenbaum

CONTENTS

LIST OF FIGURES

List of Tables

ACKNOWLEDGEMENTS

Accomplishing a PhD is without doubt a significant achievement in one's life. Although it is habitually accredited to a single person, its author, in most cases there is a number of people who have contributed one way or another to this goal. Having completed my thesis, it is time to give credit to some of the people that played a role in my PhD.

Undoubtedly, my principal promotor, Maarten van Steen, deserves the lion's share of the credit. Without any sense of exaggeration, Maarten combined all I needed in an advisor: support, motivation, and excellent tutoring in research. Maarten has always been enthusiastically devoted to research and to his supervising role. Despite his admittedly busy schedule, he always religiously respected our weekly meeting slot. These meetings provided the ground for numerous constructive discussions, fruitful brainstorming, in-depth analyses, and direction planning. Maarten taught me a whole lot of things, from the gory details of authoring, presenting, and various practical issues to a high level vision in research. This was further enhanced by his talent in passing on his enthusiasm for original research and nifty new ideas to people around him. Maarten played a crucial role in shaping my research thinking and mentality, and I am truly grateful to him.

My interaction with my second promotor, Andy Tanenbaum, although more sporadic, was of great value. Andy gave a critical view to peer-to-peer systems, raising interesting issues, and questioning the "decentralization objectives" that Maarten and I were taking almost for granted. I am grateful to him for his interesting feedback, and for giving me the opportunity to work in this highly competent research environment.

Then, I would like to thank the members of my reading committee, Anne-Marie Kermarrec (also in the role of a *referent*), Ken Birman, Márk Jelasity, Herbert Bos, and Guszti Eiben for their effort and time in reviewing this dissertation. I have put their comments to good use.

In the past I had the opportunity to collaborate closely with two of the members of my reading committee, Márk Jelasity and Anne-Marie Kermarrec. Márk Jelasity introduced me to the world of epidemic protocols during his postdoc at the Vrije Universiteit. I later visited him at the University of Bologna, in the context of a joint research effort that resulted in the work presented in Chapter 3. His understanding of the intrinsic details of epidemic protocols often leads to long interesting discussions. Márk has recently been visiting a number of research groups around Europe, spreading the notion of epidemics in an epidemic-like fashion! Anne-Marie Kermarrec has been my supervisor during my internship at Microsoft Research Cambridge, in the summer of 2003. We have continued our collaboration, and I visited her at the IRISA/INRIA research institute in Rennes during the summer of 2005. Anne-Marie helped me in learning to take a step back, and get a high-level view of my work. She also has a very good sense of finding applications for the protocols we have been working on. I would like to thank both Anne-Marie and Márk, and I hope to keep up our collaboration in the future.

Then come my colleagues at the Vrije Universiteit. I would like to thank them all for a very pleasant and friendly working environment, and for many nice discussions during lunch, at the recently established "Tea at three", or in various other occasions. In fear of offending somebody by not mentioning him or her, I will avoid referring to you by name, and will thank you all as a group. I will only make an exception for my two officemates, Michal and Swami. The three of us, coming from diverse cultures, formed an interesting combination and a very pleasant atmosphere in P4.30. I will never forget our numerous discussions on world politics, ethnic culture, history, ehhh . . . and occasionally science!

I would also like to thank some people that played a role in my decision to pursue a PhD. One of them is Ulrika in Lund, who was the "coolest" PhD student I met while doing my Master's at the University of Michigan, and showed me that doing research does not rule out enjoying life! The other two are Yannis in Chicago and Baris in Washington DC. Many thanks to both of you for your valuable contribution in selecting the Vrije Universiteit for my doctoral studies.

At this point, I would like to thank my friends in Amsterdam for providing me with moral support and a great deal of—so much needed—pleasant distractions during my PhD years. Many thanks to Anna, Madelijne, Rodrigo, Nikos ($\times 3$), Takis ("the President"), Emilios, Dafni, Ankie, Eva, Tom, Eduardo, Vicky, Victoria, and many many more. I would also like to thank my "huisgenoten" (Liza, Hetty, Ronny, Rosanne, Rink, Dejan, etc.) for forming an enjoyable socializing environment, and for putting up with my busy schedule.

At another note, I am deeply thankful to the Alexander Onassis Public Benefit Foundation in Greece for sponsoring my studies by means of a student scholarship. Such generous sponsorships prove the public benefit culture of the foundation, and create invaluable opportunities for boosting the creativity of young researchers and fostering innovation.

Last but not least, I would like to say a big THANK YOU to my family. My parents who have always unconditionally supported me in all respects, and my brother Kostas for being always a very close and dependable friend. I am grateful to them not only for their support during the years of my PhD, but also for their guidance, encouragement, and motivation throughout my whole journey in education and my life in general. Of course, I should thank them for their patience as well! I would like to end my acknowledgements by wishing my brother success with his own PhD.

Spyros Voulgaris

Zürich, August 2006

Part I

PROTOCOLS

CHAPTER 1

Introduction

We are arguably living in the communications revolution era. Communications have never been as ubiquitous, massive, fast, and inexpensive as they are today. Networks in general and the Internet in particular play a catalytic role in contemporary societies. The days when the Internet was a mere research tool, or a communication channel exclusively for academic and military institutions are long left behind. Access to the Internet has spread at an astonishing speed, appealing to an increasingly broader public, emerging the Net from an academic elite tool to a global cult that shapes the way of living on the planet.

The Internet offers a wealth of functionalities by means of deployed *services*. The *World Wide Web*, and the *Electronic Mail* (*e-mail*) are without doubt the two principal services that helped the Internet break its strictly academic barriers, and which remain among the most popular services nowadays. Other "traditional" services include *Telnet*, the *File Transfer Protocol* (*ftp*), *newsgroups*, *IRC* and several others. More recently, a number of more complex services have appeared, including e-commerce, online auctions, file sharing systems, streaming services, etc. This tremendous growth of the Internet has naturally caused an evolution of the models of communication, that we will discuss below.

Early Internet services were designed and built around two principal paradigms, namely the client-server model and centralization. The *client-server model* imposes a clear distinction between nodes that provide a service (*servers*) and those that make use of it (*clients*). Clients are passive entities accessing the service provided by dedicated nodes, the servers. *Centralization*, in its turn, refers to the fact that a service is offered by a single dedicated computer.

The dramatic expansion of the Internet proved the traditional centralized architecture inadequate for a number of services in the large scale. As a result, considerable effort has been made in the area of *distributed systems*. The idea was to abolish centralization by distributing a service across multiple cooperating

servers. Such a distribution can offer a multitude of benefits, including fault tolerance, increased aggregate capacity, geographic distribution, etc. However, clients remain passive entities, using the service collectively offered by the servers.

In the more recent years, advances in network speed and successive leaps in computing power and storage capacity, in combination with the vast number of networked low-end computers, resulted in considerable amounts of resources accumulating in the end-nodes. This has naturally created incentives to harness the power of such a huge pool of resources. Consequently, end-nodes were promoted from passive entities to active components in massively distributed environments. This goes well beyond the—once exclusive—paradigm of end-nodes passively using services, described above. In this new model, known as the *peer-to-peer* (abbreviated *P2P*) model, nodes do not only passively *consume* resources, but they also *participate*, *interact*, and *contribute* to the services they make use of. Instead of nodes being divided in servers and clients, in P2P systems nodes are equal peers collaborating to collectively carry out a service.

The P2P model of communication poses a number of challenging issues. Most important is the unprecedented scale of P2P systems. The size of decentralized systems has grown from large to massive, in certain cases involving millions of nodes spread all around the globe. And with massive scale comes massive instability, due to the highly dynamic nature of systems of that size. Typically, nodes in P2P systems provide no guarantees regarding their participation patterns. Instead, they join and leave at any time at will, let alone unpredictable ungraceful failures of nodes and links. Finally, nodes in P2P systems are generally heterogeneous with respect to their hardware architectures and software platforms, as well as regarding their computational, memory, storage, and network resources.

The aforementioned issues place a critical burden on the administration and management of such systems. Massive scale in combination with highly dynamic environments render explicit control of such systems infeasible, or at least very hard to implement. Centralized systems impose severe limitations in keeping track of millions of nodes coming and going. Scale issues can be alleviated by means of a more sophisticated architecture, such as a hierarchical structure. However, this increases complexity and administration costs, and the system depends on the availability of certain key nodes. Solutions introducing redundancy by means of replication exist, however they inflict additional complexity which is nontrivial at such scale. On top of that, all these solutions (i.e., both centralized and hierarchical ones) require an educated guess of the expected system size in order to appropriately allocate adequate resources. Generally, trying to impose explicit control on this class of systems and tracking them by traditional deterministic methods becomes increasingly complex.

The increasing number of connected devices, and the proliferation of net-

works, provide no indication of a slowdown in this tendency. Quite on the contrary, we anticipate new, unimaginable, large-scale distributed applications to emerge. In our research we step away from the deterministic and explicit control of massive-scale systems. In contrast, we explore methods that enable the autonomous management of such systems by means of self-organization.

The goal of this dissertation is to address such challenges in scale and administration, introduce new protocols, and explore their usage in a number of current or future applications. It should be noted that our goal is not to change the Internet communication model as a whole, as a significant number of applications work well the way they are. We aim at providing alternative solutions to communication models where traditional approaches are either too expensive, do not scale well, require excessive administration, or simply are not applicable for.

1.1. DESIRED SOLUTION

The main question we are faced with is *how decentralized systems should be*. First, they should be self-configurable. That is, configuration and administration of such systems should be simple. It should be done by means of high-level commands, such as join this cluster or that application. They should be plug-and-play, in a decentralized sense.

Ideally, large-scale decentralized systems should be self-organized. They should adapt fast and reliably to changes. In fact, they should be flexible enough to withstand continuous changes, in very dynamic environments. In the presence of large-scale failures (such as network partitioning) they should be robust, and they should demonstrate self-healing behavior.

Considering their anticipated size, decentralized systems should be highly scalable. Even more, they should demonstrate *elastic* capacity. By elastic we mean that the capacity of a system should be proportional to its size. This way, the larger a system gets (and therefore the higher the demand), the more capacity it has to serve that demand. On the other hand, when the system shrinks, the capacity decreases accordingly. Such networks can spontaneously adapt to unexpected increase or decrease in demand.

Last, but not least, large-scale decentralized systems should be simple. In systems scaling to million of nodes, things can easily get out of control.

1.2. WHY NOT DHTS?

A significant direction of recent research in P2P systems has been in designing structured overlay networks for routing. Such systems are widely known as *Distributed Hash Tables* (DHTs). A number of DHT systems have been proposed to date, most important representatives being Pastry [Rowstron and Druschel 2001a], Tapestry [Zhao et al. 2001, 2004], Chord [Stoica et al. 2001], and CAN [Ratnasamy et al. 2001a]. Their common property is that they all form a structured overlay connecting a large number of participating nodes, which is used to route packets among them.

DHT systems assign each participating node an ID, and route messages to a node based on that, rather than based on its IP address. Performance (in terms of routing hops) is typically inferior compared to traditional IP routing, but this is not the point. Their significance lies in the fact that they map ID keys to nodes, in a way similar to traditional hash tables mapping numeric keys to table entries. This function is crucial as a building block for numerous other applications, such as distributed network storage (OceanStore [Kubiatowicz et al. 2000] and PAST [Rowstron and Druschel 2001b]), distributed web caching, etc. These and other DHT applications are based on the following observation. Any indexable set of data can be appropriately mapped to the nodes of a DHT overlay and spread accordingly. The data entry having a given ID is placed on the node with the closest ID to that. Subsequently, given the ID of a data entry, the node hosting it can be easily located through DHT-based ID routing.

Although DHTs have reached a substantial level of maturity, they do not form a panacea for the P2P world. DHTs are excellent for ID-based routing, and, consequently, for exact query matching. They generally excel as a P2P framework for ID-centric operations. However, not all applications are ID-centric. For instance, despite being just right for ID-based queries, DHTs are not appropriate queries based on arbitrary predicates. Neither are they appropriate for building a semantic overlay, that is, linking nodes that are semantically related.

In essence, DHTs form *structured* P2P topologies, where links are determined by the IDs of participating nodes. Data is allocated to nodes according to their IDs. Subsequently, this structure is exploited to efficiently perform ID-based operations. In contrast, the protocols we explore in this dissertation form *unstructured* P2P topologies, in which links are placed: (a) uniformly at random, or (b) based on data laying on nodes. The main difference between DHTs and the type of overlays we build in this dissertation can be synopsized in the following phrase:

DHTs place data after the topology. Our protocols build the topology after data.

1.3. THE GOSSIPING MODEL OF COMMUNICATION

In this dissertation we explore *gossiping protocols* for building and managing P2P overlays. Gossiping is a major representative of fully decentralized, self-organizing systems. Let us take a look at what gossiping protocols are, and how we use them for overlay construction.

1.3.1. Traditional Gossiping

Gossiping—also known as *epidemic*—protocols, were first introduced by Demers et al. [Demers et al. 1987] in 1987. Demers et al. employed them in propagating updates in loosely replicated databases to maintain mutual consistency among the replicas. Ever since they have been used for information dissemination [Birman et al. 1999; Kermarrec et al. 2003; Eugster et al. 2004], for disseminating group membership to the whole group [Golding and Taylor 1992], for failure detection [van Renesse et al. 1998], for garbage collection [Guo et al. 1997], for aggregation [Kempe et al. 2003; Jelasity et al. 2004b, 2005; Montresor et al. 2004], for bootstrapping networks [Voulgaris and van Steen 2003; Jelasity et al. 2006] and for load balancing [Jelasity et al. 2004c]. Generally, gossiping protocols have been associated mostly with dissemination of information.

Omitting specific details, gossiping protocols operate based on the following simple basic model. Each node has a complete view of the network, and periodically picks a random node from the whole network to exchange data with. This permits information known by any *one* node to spread to the whole network with very high probability.

1.3.2. Gossip-based Topology Construction

Maintaining complete membership tables on each node is infeasible for large-scale, highly dynamic networks. Such an effort would impose tremendous synchronization problems, especially since some studies of P2P systems measured that a large percentage of nodes join for a few minutes or up to an hour (see [Bhagwan et al. 2003; Saroiu et al. 2003; Sen and Wang 2004]).

In our work we deviate from the traditional gossiping model in two respects. First, we drop the assumption of complete knowledge of the network. Each node maintains only links to a very small number of *neighbors*, constituting its *partial view* of the network (e.g., only a couple of dozen nodes in networks in the order of millions). When gossiping, a node picks a random neighbor from its partial view to exchange data with.

Second, the type of data exchanged between nodes is neighbors from their partial views. That is, nodes exchange membership information itself. After a gossip

exchange nodes update their partial views to incorporate (part of) the received links, if needed. We are, effectively, dealing with membership management.

By exchanging links and continuously refreshing their partial views, nodes can self-organize into topologies that better suit certain applications. In particular, depending on the target application our protocols exchange links in two main manners:

Randomly Exchanging links at random leads to randomized overlays. That is, the partial view of each node consists of links to randomly selected neighbors. Note that selecting a random node from a randomized partial view is essentially equivalent to selecting a random node out of the whole network, which is the assumption made by traditional gossiping protocols. In other words, randomized overlays can serve as a substrate that allows the operation of traditional gossiping protocols in very large and dynamic networks, bypassing the need for full knowledge of the network. In addition, randomized overlays demonstrate remarkable robustness with respect to large-scale failures, and highly dynamic conditions with nodes joining and leaving frequently. As we will see, they are ideal for keeping large sets of nodes in a single connected overlay.

Methodically By exchanging links based on certain criteria, nodes can self-organize in structured overlays. A multitude of P2P applications can be served by forming the appropriate structures. As we will see, the framework for forming structured overlays is very simple and generic.

1.4. WHY GOSSIPING?

An interesting question is *what drove us in pursuing desirable solutions in the domain of gossiping protocols*. Let us motivate our decision by presenting a brief list of the main advantages of gossiping protocols.

Scalability Gossiping protocols are inherently scalable. This scalability stems from the fact that each node performs a fixed set of operations, at a fixed rate, irrespectively of the network size. With respect to the load per node, gossiping protocols are, therefore, virtually infinitely scalable.

Fault-tolerance and Robustness As we will see in this dissertation, appropriately designed gossiping protocols turn out to be extremely robust with respect to large-scale failures and node churn. Fault-tolerance is achieved by means of redundancy, and this proves to be very robust.

Symmetry Gossiping protocols form symmetric overlays. That is, nodes are roughly equal, with no node playing a specific, critical role. As a consequence, no single node failure can have a significant effect on the correct operation and state of a gossiping protocol.

Graceful degradation Although individual node failures generally do not disturb the operation of a gossip-based system, large numbers of failures can naturally affect it. However, as we shall see throughout this dissertation, performance, functionality, and reliability of gossip-based systems do not drop rapidly as the number of failures increases.

Adaptability Gossiping protocols appear to form overlays that are highly adaptable to network changes and dynamic environments. We will repeatedly see this throughout this dissertation.

Elasticity Although it is generally feasible to support large-scale services by means of one or more dedicated servers, the appropriate resources have to be provisioned for certain load levels in advance. When the load exceeds a certain threshold, such systems fail to fulfil their duties. When the load is lower, dedicated servers are underutilized, their resources being wasted. This issue becomes of higher concern for systems with unpredictable load or high load fluctuation. In contrast, gossip-based—and generally P2P—systems do not face this problem, as their resources are inherently proportional to the system load. Since peers use *and* contribute resources at the same time, both load and resources grow aside, proportionally to the number of participating peers. We say that P2P systems are inherently *elastic* with respect to load.

An additional, important advantage of gossiping protocols is their *simplicity*. Simplicity is an essential advantage for systems of massive scale, which tend to be inherently complex. The interesting combination between simplicity and the remarkable properties of gossiping algorithms, motivated us in exploring them for building self-organizing systems.

1.5. RESEARCH METHODOLOGY

The research laid out in this dissertation is validated experimentally. Gossiping protocols generally exhibit chaotic behavior that is too complex to be formally analyzed. Theoretical analysis of our protocols has been out of the scope of this dissertation.

Our experiments were carried out in the following two ways: simulation and/or emulation. For the former, we simulated up to 100K peers in PeerSim, an event-driven simulator running on a single physical machine [PeerSim]. For emulated experiments we implemented the respective protocols or applications, and deployed them on the DAS-2 wide-area cluster, consisting of 200 physical machines [DAS-2]. By mapping a number of peers on each physical host, we were able to emulate up to 65K peers. Although the majority of the experiments were carried out by simulation, our emulations provide strong evidence that what we see in simulations constitutes a very accurate prediction of protocol behavior in realistic systems.

1.6. OUTLINE AND CONTRIBUTIONS

This dissertation is split in two parts. Part I presents the gossiping protocols we have developed. Part II lays out a number of applications making use of these protocols. Let us now list the chapters that constitute this thesis, along with a short description of the thesis contributions per chapter.

PART I

Chapter 2 — Building Random Overlays: CYCLON Chapter 2 introduces CYCLON, a gossiping protocol for inexpensive membership management in very large P2P overlays. CYCLON constitutes a fundamental protocol serving a number of applications presented in subsequent chapters. It is highly scalable, very robust, and completely decentralized. Most important is that the resulting communication graphs share important properties with random graphs.

Chapter 3 — Random Overlays: Exploring the Design Space Chapter 3 expands on our study of gossiping protocols that form randomized overlays by extending the gossiping framework introduced in [Jelasity et al. 2004a] and exploring the design space. CYCLON constitutes a specific instance in this framework. This chapter aims at providing an insight into the factors that affect certain aspects of the behavior of gossiping protocols.

Chapter 4 — From Randomness to Structure: VICINITY In Chapter 4 we take a shift and investigate how gossiping and random overlays can be harnessed to create structure. We present the VICINITY protocol, which, in conjunction with CYCLON builds arbitrary topologies. VICINITY is a completely decentralized protocol, and nodes self-organize to form the desired structure.

PART II

Chapter 5 — Routing Table Management: Building Pastry Chapter 5 presents our first attempt to devise structure from randomness. It presents how a gossiping protocol that builds randomized overlays can be used to build the routing tables of the Pastry DHT [Rowstron and Druschel 2001a]. This work has preceded the VICINITY protocol, which is more powerful in building and maintaining topologies. Nevertheless, this research deserves a place in this dissertation, as an illustration of our first step in harnessing randomness to create structure.

Chapter 6 — Information Dissemination Chapter 6 presents a protocol for information dissemination, that combines gossip-based probabilistic dissemination (through CYCLON) with deterministic dissemination (through VICINITY). In fail-free and static environments our protocol guarantees dissemination to every single node. In the presence of failures or node churn, its performance degrades gracefully, however still managing to deliver messages to a large proportion of the network.

Chapter 7 — Semantic Overlay Networks Chapter 7 presents how the CYCLON and VICINITY protocols can be used to significantly enhance content-based searching. This is done by establishing links among semantically related nodes, that is, among nodes of similar content. In essence, the links established form a semantic overlay network.

Chapter 8 — SUB-2-SUB: Purely P2P Publish/Subscribe Chapter 8 presents SUB-2-SUB, a fully decentralized, collaborative, content-based publish/subscribe system. SUB-2-SUB is a self-organizing system based on the CYCLON and VICINITY protocols.

Finally, Chapter 9 summarizes our most significant observations and conclusions, and discusses future directions for the research presented throughout this dissertation.

CHAPTER 2

Building Random Overlays: CYCLON

The unprecedented growth of the Internet in size and speed has triggered the conception of massive scale applications involving a very large number of nodes in the order of thousands or millions. The client-server model of communication is often not adequate for applications of such scale. This has encouraged the emergence of an alternative communication model, namely *peer-to-peer* (*P2P*) systems. The main philosophy behind these systems is communal collaboration among peers: sharing both duties and benefits. By distributing responsibilities across all participating peers, they can collectively carry out large-scale tasks in a simple and generally scalable way, that would otherwise depend on expensive, dedicated, and hard to administer centralized servers.

A distinguishing feature of P2P systems is that the peers jointly maintain an *overlay network*. Unlike traditional layer-3 networks, the structure of these overlay networks is not dictated by the (often fairly static) physical presence and connectivity of hosts, but by *logical relationships* between peers. In particular, P2P overlay networks are generally required to handle a much higher rate concerning the joining and leaving of nodes, while at the same time they assume that membership behavior is roughly the same for all nodes. In other words, they are designed to handle highly dynamic, symmetric networks. Overlay management is therefore a key issue in designing P2P systems.

There are currently two main categories of P2P systems in terms of overlay management. Structured P2P systems impose a specific linkage structure between nodes. Distributed Hash Tables [Balakrishnan et al. 2003] are typical examples in this category. They link peers based on their IDs in a way to enable efficient ID-based routing among them. In contrast, in unstructured P2P systems peers are not linked according to a predefined deterministic scheme. Instead, links are created

either randomly (as is typically done in Gnutella), or are probabilistically based on some proximity metric between nodes (i.e., semantic proximity, leading to what are known as semantic overlay networks). Unstructured P2P systems are primarily designed to support rapid information dissemination and content-based searching in highly dynamic distributed environments, but their utility extends beyond these applications.

In this chapter we concentrate on gossip-based unstructured P2P overlays. In such systems nodes self-organize in unstructured overlays by gossiping to each other to discover new links. Two key issues govern these systems. First, defining the gossiping protocol specifics that result in overlays with certain desirable properties. Second, ensuring that these properties are maintained or are swiftly recoverable in the event of major disasters or enduring highly dynamic conditions, a property often referred to as *self-healing*. Above all, these requirements should be met without maintaining any global information or requiring any sort of central administration.

It has been observed that a number of overlays appearing in social, ecological, neural, science collaboration, and phone call networks, as well as the World Wide Web itself, exhibit properties of small-world and scale-free networks [Albert and Barabási 2002; Newman 2002]. Certain types of small-world networks are very appealing, notably when decentralized search is at stake [Kleinberg 2000a, b, 2001, 2004]. However, in many cases it is better to construct overlays that are close to random graphs [Bollobas 2001]. More specifically, random graphs exhibit higher robustness to large-scale failures than small-world networks, while they distribute the load evenly across all nodes in the system. In this chapter we introduce CYCLON, a gossiping protocol that organizes nodes in overlays similar to random graphs.

Gossip-based unstructured overlays can be used as a building block in a variety of network management applications, especially when highly dynamic environments are anticipated. A good example is self-monitoring in large-scale networks, where each node is charged with monitoring a few random others, sharing the monitoring cost. An appropriately chosen overlay management algorithm can ensure load fairness, and can guarantee that no node is left unattended, resulting in a robust self-monitoring system. Another example is our work on managing routing tables in very large P2P networks [Voulgaris and van Steen 2003], presented in Chapter 5 in this dissertation. Stavrou et al. suggest management of flash crowd crises by disseminating information over gossip-based unstructured overlays [Stavrou et al. 2004]. In Chapter 4 we build on top of the research presented in this chapter to provide a framework that organizes nodes in desirable structured topologies. That framework is further utilized in a number of applications illustrated in Part II of this dissertation.

The problem we address is that of building and maintaining overlays with properties suitable for diverse applications like the aforementioned ones. More specifically, we are looking into inexpensively building and maintaining very large, connected overlay networks that exhibit important properties of random graphs. These properties should be maintained even in highly dynamic environments. In essence, we are interested in inexpensive membership management, in the sense that any disruption of the overlay's properties resulting from joining or leaving nodes should be quickly and efficiently corrected. We assume that nodes maintain a small, partial view of the entire network. Our starting point is the view exchange protocol described in [Stavrou et al. 2004]. That protocol ensures that connectivity of the overlay is maintained as long as membership does not change.

We make two contributions. First, we provide an experimental analysis of the basic swapping protocol for large networks, examining properties such as clustering and node degree distribution. These experiments have not been conducted before, and, in particular, not on large networks. We demonstrate that swapping neighbors is indeed a promising exchange protocol.

Second, and most importantly, we describe CYCLON[1], a complete framework for inexpensive membership management. CYCLON introduces an enhanced version of swapping, which results in node-degree distributions that exhibit better properties than those found in overlays resulting from basic swapping, or even in random graphs. Moreover, it includes better, in terms of efficiency and quality, management of node additions and removals, which allows us to establish truly inexpensive membership management that does not disrupt the randomness of the overlay network.

2.1. THE PROTOCOL

CYCLON is based on the *swapping* operation, by which two nodes gossiping with each other *swap* some randomly chosen links of theirs. The main intuition behind this operation is to mix links, resulting in overlays resembling random graphs. The experimental analysis following in Section 2.2 confirms this intuition.

2.1.1. Basic Swapping

The basic swapping algorithm, introduced in [Stavrou et al. 2004], is a simple peer-to-peer communication model. It forms an overlay and keeps it connected by means of an epidemic algorithm. The protocol is extremely simple: each peer

[1]The name CYCLON was inspired by the protocol's power in mixing nodes in the network, sort of like a tornado. It is also inspired by the Greek origin of the word, *kyklos* (=*circle*), due to the uniformity it imposes on the overlay.

knows a small, continuously changing set of other peers, called its *neighbors*, and occasionally contacts a random one of them to exchange some of their neighbors.

More formally, knowledge regarding neighbors is stored and exchanged by means of *node descriptors*. A node descriptor referring to peer P contains P's contact information, (i.e., network address and port). Each peer maintains a small, fixed number of ℓ node descriptors, which constitute its *view* of the network. Typical values for ℓ are 20, 50, or 100. When the view of peer P contains a descriptor of peer Q, we say that Q is a neighbor of P, or that P has a *link* (or *pointer*) to Q. These expressions will be used interchangeably.

Each peer P repeatedly initiates a gossip exchange (also referred to as view exchange), known as *swapping*, by executing the following six steps:

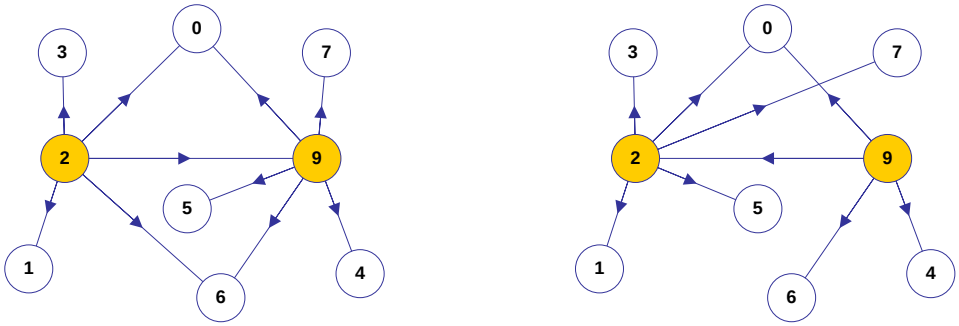
1. Select a random subset of g neighbors ($1 \leq g \leq \ell$) from P's own view, and a random peer, Q, within this subset, where g is a system parameter, called *gossip length*.
2. Replace the descriptor of Q with that of P.
3. Send the updated subset to peer Q.
4. Receive from Q a subset of no more than g of Q's descriptors.
5. Discard descriptors of P (if any), and descriptors that are already in P's view.
6. Update P's view to include *all* remaining descriptors, by *firstly* using empty view slots (if any), *secondly* replacing the descriptor of Q, and *finally* replace descriptors among the ones sent to Q.

Upon reception of a gossip request, peer Q randomly selects a subset of its own neighbors, of size no more than g, sends it to the initiating node, and executes steps 5 and 6 to update its own view accordingly.

Figure 2.1 presents a schematic example of the basic swapping operation.

2.1.2. Enhanced Swapping

CYCLON employs an enhanced version of swapping, that, as we shall show in subsequent sections, among other things improves the quality of the overlay in terms of randomness. Enhanced swapping follows the same model as basic swapping. The key difference is that nodes do not *randomly* choose which neighbor to swap views with. Instead, they select the neighbor whose information was the earliest one to have been injected in the network.



(a) Before swapping

$2 \rightarrow 9 : \{2, 0, 6\}$
$2 \leftarrow 9 : \{0, 5, 7\}$

(b) After swapping

Figure 2.1: An example of swapping between nodes 2 and 9. Note that, among other changes, the link between 2 and 9 reverses direction.

The first motivation behind this enhancement is to limit the time a node descriptor can be passed around until it is chosen by some node for a view exchange. As we shall see in Section 2.4, this results in a more up-to-date overlay at any given moment, as it prevents links to dead nodes from lingering around indefinitely.

The second—and far less obvious—motivation is to impose a predictable lifetime on each descriptor, in order to control the number of existing links to a given node at any time. In Section 2.2.3 we will show that as a consequence of that, pointers are distributed in a remarkably even way across all nodes.

To accommodate the aforementioned issues, node descriptors in CYCLON contain an extra field called *descriptor age*, or simply *age*. That is, a CYCLON node descriptor referring to node P is a tuple containing the following two fields:

1. P's contact information (i.e., network address and port)

2. A numeric *age* field

The age field denotes roughly the age of the descriptor expressed in ΔT intervals since the moment it was created by the node it points at. We emphasize that it is an indication of the lifetime of a particular descriptor, not the lifetime of the node itself.

In enhanced swapping nodes initiate neighbor exchanges periodically, yet not synchronized, at a fixed period ΔT. The enhanced swapping operation is performed by letting the initiating peer P execute the following seven steps:

1. Increase by one the age of all descriptors.

2. Select neighbor Q with the highest age among all neighbors, and $g-1$ other random neighbors.
3. Replace the descriptor of Q with that of P (with age 0).
4. Send the updated subset to peer Q.
5. Receive from Q a subset of no more that g of its own neighbors.
6. Discard descriptors of P (if any), and descriptors that are already in P's view.
7. Update P's view to include *all* remaining descriptors, by *firstly* using empty view slots (if any), *secondly* replacing the descriptor of Q, and *finally* replace descriptors among the ones sent to Q.

Like in basic swapping, the receiving node Q replies by sending back a random subset of at most g of its neighbors, and updates its own view to accommodate all received ones. It does not increase, though, any descriptor's age until its own turn comes to initiate a gossip exchange.

Note that after node P has initiated a gossip operation with its neighbor Q, P becomes Q's neighbor, while Q is no longer a neighbor of P. That is, the neighbor relation between P and Q reverses direction. As a consequence, the indegree of P increases by one, while the indegree of Q decreases by one.

Figure 2.2 shows the pseudocode of a generic gossiping skeleton modeling CYCLON. The functions appearing in boldface, namely `selectPeer()`, `selectToSend()`, and `selectToKeep()`, form the three *hooks* of this skeleton. Different protocols can be instantiated from this skeleton by implementing specific policies for these three functions, in turn, leading to different emergent behaviors. The policies implemented by CYCLON are presented in Figure 2.3.

As a final note, given the asynchronous nature of the protocol, the gossiping procedure of a node may be interrupted by a swap request by another node. No measures are taken to explicitly control concurrency issues. Practice showed that occasional intervening swaps do not alter the macroscopic behavior of the system (see Section 3.6).

From now on, any references to *swapping* apply to *both* basic and enhanced swapping. Otherwise, the swap type will be explicitly specified.

2.2. BASIC PROPERTIES

As it turns out, swapping has some desirable statistical properties. In order to observe the characteristics of the overlay, we need to consider the connectivity

```
do forever
{
  wait(T time units)
  incr. all descriptors' age by 1
  Q ← selectPeer()
  remove Q from view
  buf_send ← selectToSend()
  send buf_send to Q
  receive buf_recv from Q
  view ← selectToKeep()
}
```

Active thread

```
do forever
{
  receive buf_recv from Q
  buf_send ← selectToSend()
  send buf_send to Q
  view ← selectToKeep()
}
```

Passive thread

Figure 2.2: The generic gossiping skeleton for CYCLON.

Hook	**Action taken**
`selectPeer()`	Select descriptor with the oldest age
`selectToSend()`	
active thread	Select $g-1$ random descriptors. Add own descriptor with own profile and age 0.
passive thread	Select g random descriptors.
`selectToKeep()`	
active thread	Keep all g received descriptors, replacing (if needed) the descriptor selected by `selectPeer()` and then the $g-1$ ones selected by `selectToSend()`.
passive thread	Keep all g received descriptors, replacing (if needed) the g ones selected by `selectToSend()`. In case of multiple descriptors from the same node, keep the one with the lowest age.

Figure 2.3: Implementation of the generic gossiping skeleton hooks, for the CYCLON protocol.

graph, that is, the graph having the peers as vertices, and the links between them as (directed) edges. In this section we consider—unless otherwise stated—the undirected version of the connectivity graph, which is taken by simply dropping the direction of the edges. The motivation behind this is that we are interested rather in the "can-communicate" than the "knows-about" version of the graph. A node has the same potential to communicate with another node either if the first is a neighbor of the second or vice versa.

In this section we show and experimentally analyze the basic properties of the basic and enhanced swapping protocols. All experiments presented here have been carried out with an event-driven simulator we developed in C++.

2.2.1. Connectivity

The fundamental property of swapping is that, given a fail-free environment, connectivity of the overlay is guaranteed.

Considering the trivial case of *individual* nodes' connectivity, it should be clear that no node becomes disconnected as a result of a swap operation. It simply moves from being the neighbor of one node to being the neighbor of another. This provides an intuitive indication that connectivity is generally preserved.

To provide a complete proof, we now show that an overlay cannot be split in two *disjoint* subsets as a result of a swap operation. Assume two subsets A and B connected by at least one link. For instance, some node in A has a pointer to some node in B. Swaps within subset A may pass this pointer around, but it will always point at the same node in B, keeping the two subsets connected. Swaps within subset B do not interfere with this link. Finally, a swap between the node in A currently holding the pointer, and the node in B being pointed at, will simply reverse the direction of the pointer, thus maintaining the link between the two subsets. Therefore, no swap operation can result in A and B (or generally any two subsets of the overlay) becoming disconnected.

2.2.2. Convergence

In this section we study the convergence of specific statistical properties of CYCLON, namely the shortest path length, and the clustering coefficient, outlined below.

The *shortest path length* between nodes P and Q is the minimum number of edges needed to traverse to reach Q from P. The *average path length* is the average of the shortest path lengths between any two nodes. The average path length is a metric of the number of hops (and hence, communication costs and time) to reach nodes from a given source. A small average path length is therefore essential for broadcasting or, generally, information dissemination applications.

The *clustering coefficient* of a node is defined as the ratio of the existing links among the node's neighbors over the total number of possible links among them. It basically shows to what percentage the neighbors of a node are also neighbors among themselves. The *average clustering coefficient* is the clustering coefficient averaged across all nodes in the network. It is generally undesirable for an overlay to have a high average clustering coefficient for two reasons. First, it weakens the connectivity of a cluster to the rest of the network, therefore increasing the chances of partitioning. Second, it is not optimal for information dissemination applications due to the high number of redundant message deliveries within highly clustered node communities.

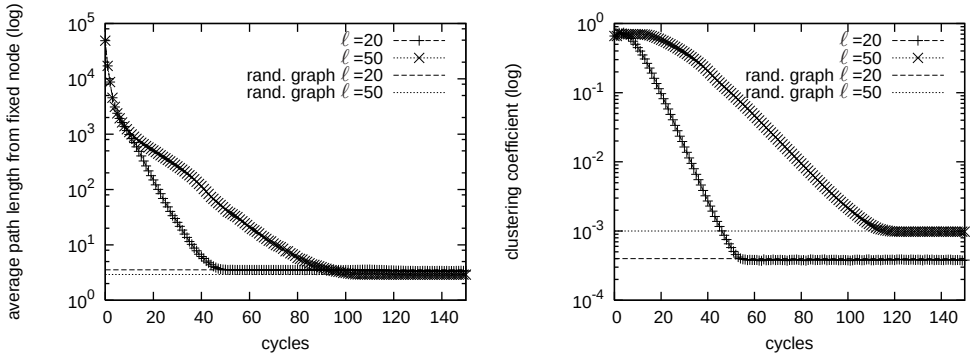
To study the emergent behavior of CYCLON, we define a *cycle* to be the time period during which a number of gossip exchanges equal to the number of nodes have been made. Since nodes initiate gossip exchanges periodically, at the same rate, a cycle coincides with the gossip period ΔT. Note that during a cycle, each node has initiated a gossip exchange exactly once. We studied the protocol's emergent behavior by observing the aforementioned statistical properties at times 0, ΔT, $2\Delta T$, etc.

Note that the selection of the period ΔT effectively regulates the speed at which an experiment runs in real time. However, it does not affect the protocol's emergent behavior or its convergence speed with respect to the number of cycles elapsed. Nevertheless, ΔT should not be comparably short to twice the typical latencies in the underlying network, as network delays would unpredictably affect the order in which events are taking place. Typical values of $\Delta T = 10sec$ or higher are recommended for experiments running over a wide-area network.

We conducted a series of experiments involving networks containing up to 100,000 nodes. In fact, we initialized our experiments in various different ways, including random, star, lattice, and gradually growing topologies. We observed that the statistical properties of the emerging overlay always converged to the exact same values, irrespectively of the bootstrapping method employed each time. The experiments presented in this section (Fig. 2.4) were intentionally bootstrapped with the worst imaginable setting. Nodes were set to form a "chain", each node having a single neighbor, namely its previous one. For instance, the average path length was initially the longest possible: 99,999 hops were needed to reach the first from the last node.

Figure 2.4(a) demonstrates a significant aspect of the emergent behavior of swapping. It clearly shows that the average path length converges to a very small value, which coincides with the average path length of a random graph with the same number of edges.

Figure 2.4(b) shows that swapping exhibits convergent behavior for the clustering coefficient. In our experiments, the clustering coefficient always converged



(a) (b)

Figure 2.4: (a) Average shortest path length between two nodes for different view lengths. (b) Average clustering coefficient taken over all nodes.

to values practically equal to the clustering coefficient of random graphs. Moreover, both the average path length and the clustering coefficient converge almost exponentially (linearly in the log-linear graph).

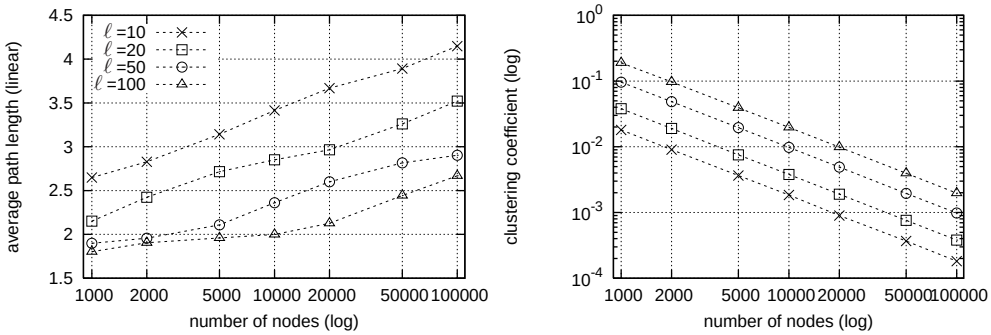
Note that these experiments ran for several thousand cycles. However, only the initial cycles are depicted, as the values remained stable for all subsequent cycles, indicating convergent behavior.

Figure 2.5 shows the converged values for the average path length and the average clustering coefficient, for overlays of different numbers of nodes and view lengths. Two initial observations can be made. First, the average path length increases logarithmically as a function of the number of nodes in the overlay. Second, the clustering coefficient drops exponentially (linearly in the log-log graph) as a function of the number of nodes.

What is more interesting is that both the average path length and the clustering coefficient converge to the exact values expected in a random graph of the same number of nodes and links. A random graph is a graph where an edge between two random nodes exists with a probability p. Consequently, p is equal to the ratio of existing links among nodes, over the total number of node pairs (total number of *possible* links). Also, in random graphs, the average clustering coefficient, C_{rand}, is equal to p. Therefore, it follows that

$$C_{rand} = p = \frac{\#links}{total\ \#\ of\ possible\ links} = \frac{\#links}{\frac{N\times(N-1)}{2}} \tag{2.1}$$

where N is the number of nodes. For the sake of comparison, we consider random graphs with an equal number of links as in our graphs. In our overlay, the number



(a) (b)

Figure 2.5: Converged state of CYCLON. (a) Average shortest path length between two nodes. (b) Average clustering coefficient taken over all nodes.

of links is $N \times \ell$, where ℓ is the view length. By substituting that in 2.1, we get:

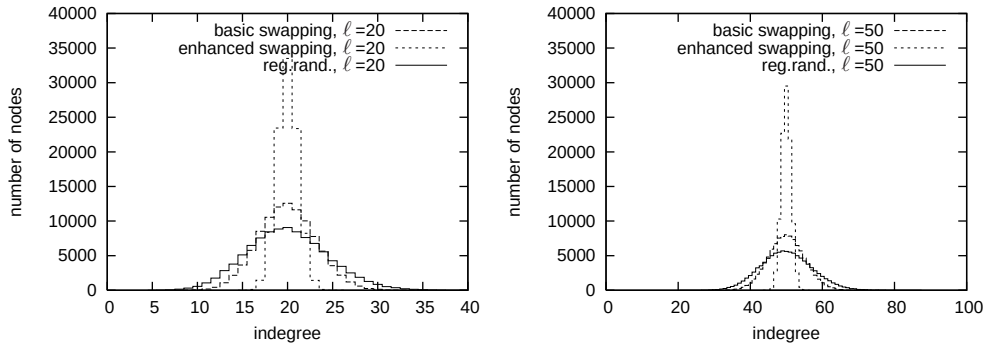
$$C_{rand} = \frac{2 \times \ell}{N - 1} \tag{2.2}$$

One can observe that formula 2.2, which gives the clustering coefficient for random graphs of the same number of nodes and edges as ours, also provides the exact values retrieved through experimentation in 2.5(b). This proves that the overlays formed by swapping have the same clustering coefficient as equivalent random graphs.

2.2.3. Degree Distribution

The *degree* of a node is the number of links it has to other nodes, in the undirected connection graph. The interest in the degree distribution stems from three reasons. First, the degree distribution is highly related to the robustness of the overlay in the presence of failures as it shows the existence of weakly connected nodes and massively connected hubs. Second, it is an indication of the way epidemics are spread. Third, it provides an indication of how fairly links are distributed among nodes, and, as a consequence, an indication of the distribution of resource usage (processing, bandwidth) across nodes. For the sake of robustness, efficient information dissemination, and load balancing, it is desirable to have a balanced, uniform distribution of links across all nodes of the overlay. In other words, it is desirable to have a degree distribution with low standard deviation.

In the directed version of the connection graph, we distinguish between the *outdegree*, and the *indegree* of a node, which are the number of edges leaving



(a) (b)

Figure 2.6: Indegree distribution in converged 100,000 node overlay, for basic swapping, enhanced swapping, and an overlay where each node has ℓ randomly chosen outgoing links. (a) View length $\ell = 20$. (b) View length $\ell = 50$.

from and ending at the node respectively. In our case, the outdegree of every node is fixed, and equal to the view length. Therefore, we concentrate on observing the *indegree distribution* of our overlays.

Figure 2.6 shows the indegree distribution for both basic and enhanced swapping for two different view lengths. The indegree distribution of an overlay with ℓ randomly chosen outgoing links per node is shown as well for comparison. In both protocols, the indegree distributions demonstrate a peak at the indegree equal to the view length, while the number of nodes having larger or smaller indegrees drops symmetrically according to the shift from the view length.

It is, however, clear that enhanced swapping does a significantly better job with respect to spreading out the links extremely evenly across all nodes. For the experiment with view length 20, 80.31% of the nodes have an indegree of $20 \pm 5\%$. For the experiment with view length 50, 93.95% of the nodes have an indegree of $50 \pm 5\%$. The respective percentages for basic swapping are 36.22% and 38.47%.

To understand CYCLON's enhancement with respect to the indegree distribution, we should take a closer look at the life cycle of pointers. A pointer to node *P* is *born* when *P* initiates a swap with one of its neighbors, say *Q*. In the absence of failures this pointer will remain in the network, possibly hopping from node to node as a result of subsequent swaps. It will *die* only when it is selected by the node currently holding it, say *R*, to perform a swap with *P*, in which case it will be replaced by a fresh pointer from *P* to *R*.

In both basic and enhanced swapping exactly *one* new pointer to each node is born in each cycle, since each node initiates exactly one swap operation. However,

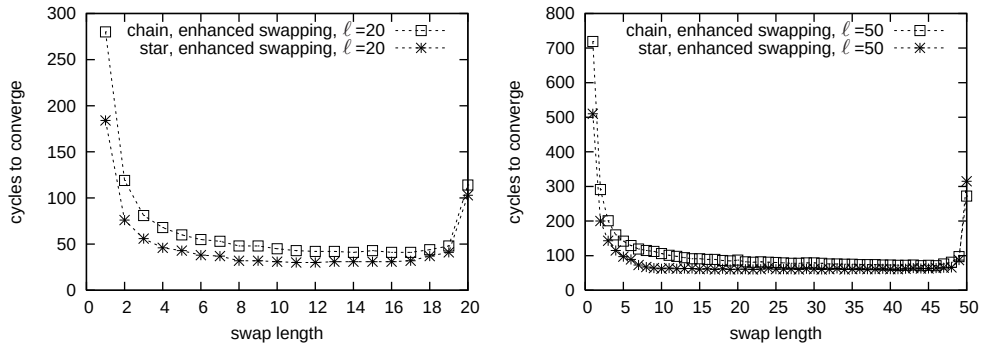


Figure 2.7: Effect of gossip length on convergence speed. N=100,000.

the two protocols differ in the *death rate* of pointers.

In basic swapping, any number of existing pointers to a given node can die in a single cycle, as they are selected for swapping *at random*. As the death rate does not follow the birth rate closely, the distribution of the *population* of pointers to individual nodes has a high standard deviation, resulting in the wide distribution of Figure 2.6.

In enhanced swapping, in each cycle a node P is contacted by one other node on average to do swapping, thus inserting one new pointer of age 0 in its view, and pushing out of its view the pointer of maximum age. In the subsequent cycle, that pointer of age 0 is upgraded to age 1, and a new pointer with age 0 replaces the currently oldest pointer. As a result, a node's view contains on average one pointer of each age, from 0 to $\ell-1$. This means that replaced pointers are typically of age around $\ell-1$. In other words, a pointer has a lifetime of about ℓ cycles. Taking this observation one step further, in each cycle *one* pointer to a given node will die, the one injected by that node ℓ cycles ago. So, the death rate of pointers to a given node is very close to one per cycle, that is, very close to the birth rate, keeping the population of pointers to that node almost static. This intuitive result can be clearly seen in the distributions of Figure 2.6.

2.2.4. Dependency on Gossip Length

We conducted a series of experiments to examine the effect of the gossip length on convergence. Interestingly, for all overlays we tried, the converged state with respect to the average path length, average clustering coefficient, and indegree distribution, proved to be *independent* of the gossip length used. The only effect of the gossip length was in the number of cycles it took to reach the converged state.

We used the indegree distribution as a metric to identify at which cycle an overlay converges. As the indegree distribution does not converge to *exact* numbers, but to a certain graph shape whose points keep slightly fluctuating, we had to use an approximation algorithm to identify the convergence point. In particular, we recorded the indegree distributions for the first 1000 cycles of each experiment. In all cases, by looking at the distribution series we could tell that they were certainly converged well before the 900th cycle. We computed the average indegree distribution of the last hundred cycles, namely cycles 900-999. Subsequently, for each cycle $i = 900 \ldots 999$ we calculated the *sum of squared errors* E_i, between this cycle's indegree distribution and the average one. The sum of squared errors for cycle i is defined as follows:

$$E_i = \sum_{k=0}^{\infty} (x_{ik} - \bar{x}_k)$$

where $X_i = \{x_{i0}, x_{i1}, x_{i2}, \ldots\}$ is the i-cycle distribution, and $\overline{X}_i = \{\bar{x}_0, \bar{x}_1, \bar{x}_2, \ldots\}$ is the average distribution of cycles 900-999. We, then, figured out the maximum of these values,

$$E_{max} = max\{E_{900}, \ldots, E_{999}\}$$

This was used as the maximum threshold of the sum of squared errors to consider a distribution converged. Finally, starting from cycle 0, we checked one distribution at a time to find the first cycle i whose E_i was below that threshold. This cycle was logged as the cycle for which this experiment converged.

To demonstrate the gossip length's effect independently of the initial condition, two different bootstrapping methods were used. The first one, *chain*, is the one described in Section 2.2.2: considering nodes in a line, each node has a single link to its previous one, forming a chain topology. In the second bootstrapping method, *star*, all nodes initially have a single neighbor, the same one for all of them, essentially forming a star topology.

Figure 2.7 presents the number of cycles an experiment took to converge as a function of the gossip length, for 100,000 nodes, and view lengths 20 and 50. In all cases, swapping just one neighbor at a time took clearly the longest. Swapping two, three, or more neighbors, gradually sped up the process. However, no significant improvement was noticed beyond gossip lengths of about eight or ten. Counter intuitively, convergence speed degraded suddenly when swapping the whole view, or almost all of it. The reason behind slow convergence either when swapping too *few* or too *many* neighbors is that in either case views are not mixed up too much by each swap operation. Therefore, a view is only minimally (if at all) enriched with new links, even if it has moved as a whole to a new node in the case of full view swapping.

This result has more significance when bandwidth is an issue, since the gossip length linearly affects the amount of bandwidth used by CYCLON. Choosing a gossip length as low as around six or eight results in nearly optimal convergence speed, while keeping bandwidth utilization to relatively low levels. We return to this issue in Section 2.6.

2.3. ADDING NODES

CYCLON introduces a new method for nodes to join the overlay efficiently, without disrupting randomness. To join, a new node simply needs to know any single node that is already part of the overlay, called its *introducer*. Such a node can be discovered in various ways, including broadcasting in the local network, making use of a designated multicast group, or even contacting a well-known server, etc. Finding such an introducer is out of the scope of this chapter. Here, we are interested in how to join once an introducer is known, without affecting the basic properties of the system.

To this end, a key observation is that due to the randomness of the connectivity graph, a random walk of length at least equal to the average path length is guaranteed to end at a *random* node of the overlay, irrespectively of the starting node.

Based on this observation, a new node P can join in a fairly straightforward manner. P's introducer initiates ℓ (view length) random walks, setting their TTL (time to live) to a small value close to the expected average path length, such as four or five. A node Q where a random walk ends, replaces one of its neighbors with a new link to P. Q then forwards the replaced neighbor to P, who, in turn, has to include it in its view. In effect, this operation is equivalent to P initiating a swap of gossip length 1 with node Q. The join operation ends when all ℓ random walks have ended and the corresponding neighbor exchanges have been accomplished.

We claim that this join operation makes it impossible for an external observer to distinguish P as being different from the rest of the nodes, or to discover any randomness disruption in the overlay. First, P's view is filled up with ℓ *randomly chosen* neighbors, which renders the values of P's average path length to reach any other node as well as its clustering coefficient, indistinguishable from the respective values of other, older nodes. Second, since there are ℓ random nodes in the overlay that know P, P's indegree is equal to the view length. Third, no other node's indegree has been modified.

In the presence of node failures or an unreliable network, some of the random walks may fail. Note, however, that a node's join does not depend on the complete execution of this join procedure. We ran experiments (not presented here) that

showed that a node can join by simply being involved in a swap with a single participating node. In that case, though, it will take a few cycles until the new node's properties become indistinguishable from the respective properties of other nodes. The join procedure described above is meant as a means of efficient node joining at a cost of a constant number of messages.

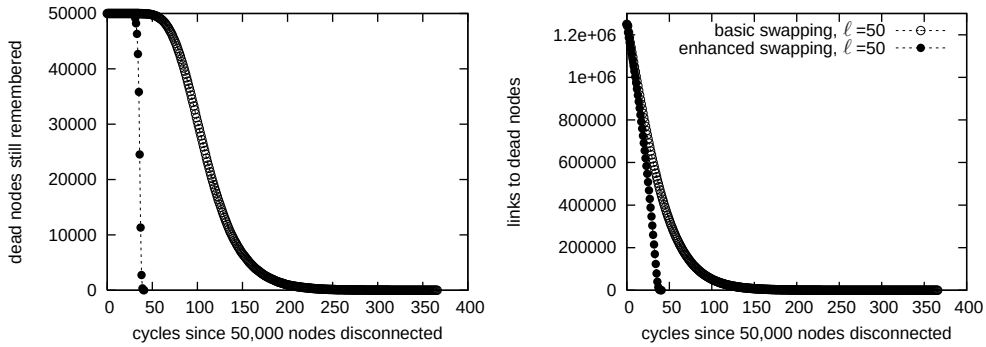
2.4. REMOVING NODES

In a dynamically changing overlay, nodes may leave for various reasons and in various ways. We make no distinction between nodes disconnecting gracefully or abruptly. What we are interested in is that, once a node disconnects, other nodes should detect it and remove any pointers to it in a timely manner. We consider pointers to disconnected nodes to constitute a sort of view *pollution*, as they take up slots that could be otherwise holding valid, useful links. Particularly in highly dynamic environments, timely elimination of dead links is crucial for the robustness of the overlay.

In order to keep the protocol simple and inexpensive, we do not add any explicit messages (such as frequent pings) to detect disconnected nodes. Instead, CYCLON uses a transparent dead-link detection mechanism, based on the default swap operation. We do, however, employ an effective strategy (by means of the *age* field introduced in enhanced swapping) to improve timely detection of disconnected nodes "for free", as a natural consequence of the emergent behavior of enhanced swapping.

When a node tries to initiate a swap with a neighbor and gets no reply within a predefined timeout, it simply assumes that neighbor to be disconnected and removes the corresponding neighbor from its view. This way, dead links are gradually being removed. Note that the timeout should be at least twice the maximum typical latency in the underlying network. For simplicity, a timeout equal to the gossip period ΔT would suffice.

In the case of basic swapping, the detection of dead links relies on chance and takes unbounded time. In CYCLON's enhanced swapping, though, the age field defines a key priority in which neighbors are contacted. A young descriptor (i.e., with low age), whose node is quite probably still alive, is less likely to be chosen for swapping. In contrast, a descriptor that has been injected in the network several cycles ago (and has since been hopping around between nodes due to gossip exchanges), is more likely to be the oldest one in the view currently hosting it, and therefore more likely to be selected to gossip with. In general, the longer a node descriptor stays in the network, the higher the chances it is selected for gossiping. When P initiates a gossip and selects a descriptor referring to peer Q, that descrip-



(a) (b)

Figure 2.8: (a) Time until dead nodes are forgotten. (b) Number of dead links.

tor is then replaced by a fresh (i.e., with age 0) descriptor of peer *P*. This process naturally recycles the node descriptors, maintaining an equilibrium with respect to their ages, consequently limiting the lifetime of a link. This way, it is no longer possible for old links to disconnected nodes to linger around indefinitely.

To demonstrate the advantages of CYCLON's enhanced swapping with respect to node removal, we ran experiments where we suddenly killed half of the nodes after the overlay had converged. As we shall show in the next section, such a drastic change poses no threat to the (remaining) overlay's connectivity. We observed how long it took for the remaining nodes to "forget" the dead ones. Figure 2.8 shows the respective graphs for the experiment with 100,000 nodes, out of which 50,000 were killed at once. Figure 2.8(a) shows how long dead nodes are still referenced, while Figure 2.8(b) shows the number of dead links that are maintained since the nodes were killed. It is clear that enhanced swapping limits the detection of dead nodes to a number of cycles equal to (in fact less than) the view length, while basic swapping takes almost an order of magnitude more cycles to decontaminate the surviving nodes' views.

2.5. ROBUSTNESS - SELF HEALING BEHAVIOR

In the previous section, we dealt with node disconnections, and we mentioned that killing half of the nodes at once does not threaten the connectivity of the remaining ones. In this section, we explore CYCLON's limits in terms of robustness to node disconnections. Swapping proves to be a very strong and robust epidemic protocol with respect to keeping an overlay connected. Moreover, it appears to ex-

hibit robustness to node disconnections similar to the one found in random graphs.

We conducted experiments as follows. We used CYCLON to create overlays (until they converged), and subsequently examined how the connectivity is affected by node removals. Figure 2.9 presents the results for networks with initially 100,000 nodes, and view lengths 20, 50, and 100, respectively. For the sake of comparison, it also presents the same graphs for overlays of view length 20 and 50, where the neighbors were randomly chosen among the whole node set. Note that basic swapping has the same behavior as enhanced swapping with respect to robustness, and therefore the corresponding graphs are not shown. Figure 2.9(a) shows the number of disjoint clusters as a function of the percentage of nodes removed. Note that the number of clusters decreases as we approach 100% node removal because the total number of surviving nodes becomes too small. Figure 2.9(b) shows the number of nodes not belonging to the largest cluster, in log scale.

These graphs show considerable robustness to node failures, especially considering the fact that in the early stages of clustering very few nodes are out of the largest cluster, which indicates that most nodes are still connected in a single large cluster. Moreover, swapping appears to share the same robustness properties with overlays where each node's neighbors are a random sample of the nodes in the network.

Note that the graph for the experiment with view length 100 is practically a flat line. That is, for 100,000 nodes and view length 100, the overlay created is so robust, that no matter how many nodes are removed, the remaining ones remain connected in a *single* cluster.

The effect of the view length on the overlay's robustness is shown in Figure 2.10. We carried out 100 experiments, with view lengths $1, 2, \ldots, 100$, and for each of them we determined the percentage of random nodes needed to be removed in order to partition the overlay. It can be seen that there is a critical value of the view length around eleven. Overlays with smaller view lengths exhibit significantly worse behavior with respect to robustness. On the other hand, overlays with view length over 85 or 90, are almost impossible to partition, no matter how many nodes are removed.

It is important to point out that the results presented in this and the previous section, suggest that CYCLON is capable of repairing an overlay after a serious disaster, a property often referred to as *self-healing* behavior. This comes as a consequence of the following two facts. First, the overlay has proven to be highly resilient to large-scale node failures. Second, once such a massive failure has occurred, the surviving nodes quickly strengthen the connectivity among themselves by replacing links to dead nodes with valid links in a timely manner.

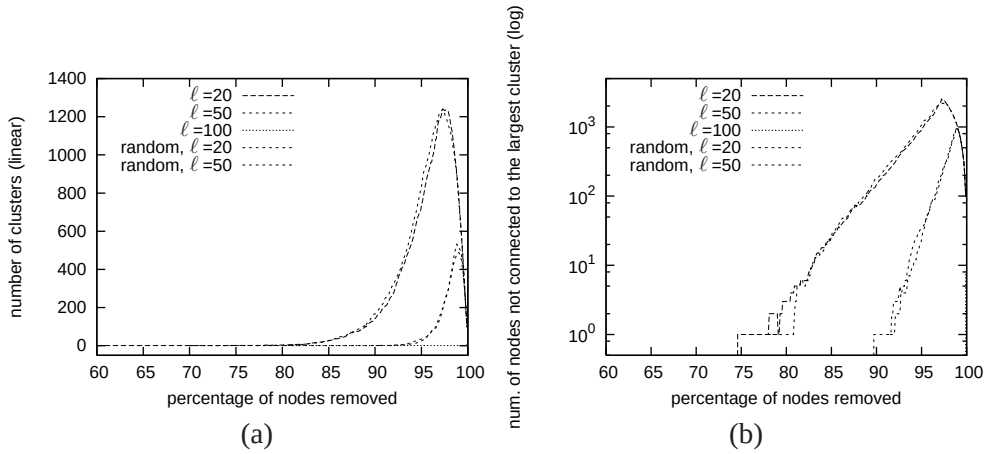


Figure 2.9: (a) Number of disjoint clusters, as a result of removing a large percentage of nodes. Shows that the overlay does not break into two or more disjoint clusters, unless a major percentage of the nodes are removed. (b) Number of nodes not belonging to the largest cluster. Shows that in the first steps of clustering only a few nodes are separated from the main cluster, which still connects the grand majority of the nodes.

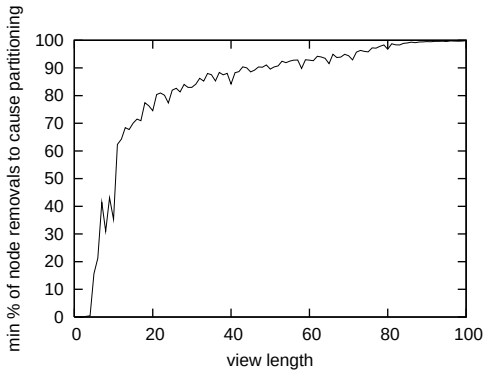


Figure 2.10: Tolerance to node removal, as a function of the view length. Network size is 100K nodes.

2.6. BANDWIDTH CONSIDERATIONS

Due to the periodic behavior of gossiping, the price of maintaining a robust overlay may inhibit a high usage of network resources (i.e., bandwidth). In the case of CYCLON, the per-node network load depends on two factors: the amount of data transferred per cycle, and the cycle duration.

In each cycle, a node gossips on average twice: exactly once as an initiator and on average once as a responder. It, therefore, on average, sends two gossip messages and receives another two in each cycle. If g is the gossip length, a gossip message consists of g node descriptors. A node descriptor is assumed to be 8 bytes long (IP address: 4 bytes, port: 2 bytes, age: 2 bytes), so, a gossip message is $8 \cdot g$ bytes long. Therefore, in each cycle a node experiences a total traffic of $32 \cdot g$ bytes.

Choosing a gossip length in the range of 3 to 8, we achieve nearly optimal convergence speed with respect to the number of cycles (see Figure 2.7). The choice of the period ΔT depends on the underlying network's bandwidth availability, as well as on the network's expected churn rate and the application's need for quick convergence. For relatively fast convergence, say, within a few minutes, we could set $g = 8$ and $\Delta T = 10sec$. In this case, a node would transfer 256 bytes every 10 seconds, that is, 25.6B/sec ($\sim$200bps). This is very low bandwidth that could be sustained even by modem connections. For less demanding environments that experience limited churn and failure rate, we could set $g = 1$ and $\Delta T = 1min$, resulting in a negligible bandwidth of less than 5bps per node.

This analysis backs our claim about CYCLON being inexpensive. Finally, note that all communication is carried out in a connectionless fashion, sending UDP packets.

2.7. APPLICATIONS

CYCLON is a core technology that can be employed as a building block in P2P applications, particularly in highly dynamic environments. The applications briefly described in this section demonstrate the utility of CYCLON in a range of different contexts.

In our work on Chapter 4, we use CYCLON as a component in a composite gossiping protocol that establishes links between nodes that demonstrate some relationship, building (almost) arbitrary topologies. In this case, CYCLON serves as a lightweight means of learning *random* nodes of the network.

Another potential use of CYCLON is self-monitoring for very large clusters of nodes, possibly in the wide-area, avoiding centralized or hierarchical archi-

tectures. In such a scheme, each node of a cluster is monitored by some other random nodes of the cluster. The bounded indegree of nodes ensures that at any given moment, each and every node will be pointed at (and therefore monitored) by a number of other nodes more or less equal to the view length. This ensures that no node will be left unattended at all.

In [Stavrou et al. 2004], the basic swapping algorithm is employed to construct an overlay that handles flash crowds. Upon detection of a flash crowd condition with respect to a document request, a client node abandons the respective server, and attempts to retrieve the document in question from peers that have already obtained it. In order to locate the document, the client node performs a series of randomized, scoped searches over the links of the overlay created by basic swapping.

In our work in Chapter 5 (also appearing in [Voulgaris and van Steen 2003]), we apply a different gossiping protocol, NEWSCAST [Jelasity et al. 2003; Voulgaris et al. 2003] (described below), in managing P2P routing tables. In particular, we exploit the randomized overlay of NEWSCAST to pick nodes that satisfy certain criteria, to fill in Pastry-like [Rowstron and Druschel 2001a] routing tables. CYCLON is, in fact, a better candidate than NEWSCAST in that application, as it exhibits a superior randomness property and lower bandwidth requirements compared to NEWSCAST.

Finally, in Newscast EM [Kowalczyk and Vlassis 2004], Kowalczyk and Vlassis use CYCLON for data aggregation, and more specifically as a component of a distributed Expectation-Maximization (EM) algorithm for probabilistic clustering of a set of (geographically) dispersed data. In particular, they conclude that aggregation using CYCLON is performed *faster* than in a fully connected graph. This is due to the bounded indegree of nodes, which results in each node having an equal influence on the aggregation.

2.8. THE NEWSCAST PROTOCOL

For the sake of completeness, we now turn our attention to NEWSCAST, a protocol that preexisted CYCLON, and which greatly contributed in developing a mindset that led to the design of the latter.

NEWSCAST, invented and introduced by Márk Jelasity et al. in [Jelasity et al. 2003], is a protocol for membership management and information dissemination in large-scale, distributed systems. Much like CYCLON, it employs a simple epidemic protocol to form an overlay network and to keep it connected.

NEWSCAST and CYCLON operate along very similar gossiping paradigms. In short, their differences lie in the following three key points. First, with respect to

peer selection, in NEWSCAST nodes select a *random* neighbor from their views to gossip with, rather than the oldest one in CYCLON. Second, when gossiping, nodes always send each other their *whole* view. Third, views are not swapped when gossiping. Instead, they are *merged*, and each node updates its view by keeping the youngest pointers out of this merged set.

In addition to these three key differences, NEWSCAST employs *timestamps* to track a pointer's age, in place of CYCLON's *age* field. However, this is merely a matter of choice that does not significantly affect either protocol's behavior. Therefore, it does not constitute a key difference between the two protocols.

We subsequently provide a brief overview of NEWSCAST's operation, and then explore some properties of its emergent behavior.

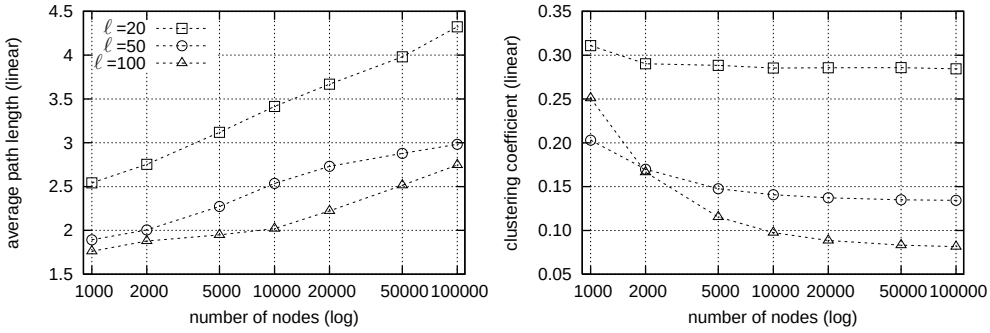
2.8.1. Principal Operation

In NEWSCAST each node maintains a small, fixed-sized view of ℓ peers. A node descriptor contains the network address of another peer, and a *timestamp* denoting when this descriptor was created. For simplicity we currently assume globally synchronized time, but we relax this assumption below.

The basic idea is that each node periodically picks a random peer from its view to gossip with and subsequently both nodes replace their view entries with the ℓ freshest descriptors of the union of their original views. More formally, each node P executes the following five steps once every ΔT time units (ΔT is referred to as the *refresh interval*):

1. Randomly select a peer Q from P's own view.
2. Add a fresh descriptor (i.e., timestamped with the current time) with P's address to P's own view.
3. Send P's view to Q, and, in return, receive Q's view (augmented by fresh descriptor of Q).
4. Discard descriptors of P (if any), and descriptors that are already in P's view.
5. Keep the ℓ newest descriptors and discard the rest.

Peer Q executes the last four steps as well, so that after the exchange both peers have the same merged view set, except for a pointer to each other. Note, however, that as soon as any of them executes the protocol again, their respective views will most likely be different again.



(a) (b)

Figure 2.11: NEWSCAST converged state. (a) Average shortest path length between two nodes. (b) Average clustering coefficient taken over all nodes.

This algorithm resembles the traditional push-pull epidemic protocol [Demers et al. 1987]. A critical difference, however, is that no correspondent knows the complete member list, but only a small, random fraction of it.

Finally, regarding the synchronized time assumption, the protocol does not really require that clocks be synchronized, but only that the timestamps of all descriptors in a single view be mutually consistent. We assume that the communication time between two peers is much shorter than ΔT (which is generally in the order of tens of seconds or minutes). When a peer P passes its view to Q, it also sends along its current local time, T_P. When Q receives P's view, it adjusts the timestamp of every descriptor with a value $T_P - T_Q$, effectively normalizing them to its local time.

2.8.2. Properties of NEWSCAST

In this section we provide a brief presentation of NEWSCAST's basic properties, under the prism of CYCLON. It is particularly interesting to see how the two protocols' different design choices reflect on their behavior.

Average path length First, with respect to the average path length between nodes, the two protocols' difference is rather insignificant. Figure 2.11(a) shows the average path length in NEWSCAST overlays for a multitude of view lengths and network sizes. Although CYCLON overlays exhibit slightly lower average path lengths, as seen in Figure 2.5(b), we do not consider this a noteworthy difference.

Clustering A more substantial difference is observed with respect to clustering. Figure 2.11(b) shows the average clustering coefficient for NEWSCAST overlays of various view lengths and network sizes. Clustering is significantly higher compared to the respective CYCLON overlays shown in Figure 2.5(b).

High clustering is a direct consequence of the NEWSCAST protocol operation. After a view exchange, two nodes share exactly the same view, apart from a pointer to each other. In essence, by exchanging views two nodes maximize their clustering with respect to each other. Exactly the opposite happens in CYCLON. By *swapping* neighbors, two nodes maintain their views' diversity. In fact, two views' diversity may even *increase* in case a common neighbor is shipped over to the other peer, and subsequently discarded while eliminating duplicate pointers. The effect of keeping common neighbors as opposed to distinct ones after a view exchange, is further studied and analyzed in Chapter 3.

Note that the combination of high clustering with low average path length suggests that the induced NEWSCAST overlays have many properties in common with what are known as *small-world networks* [Albert and Barabási 2002]. In contrast, CYCLON overlays have shown to resemble random graphs.

Degree distribution Figure 2.12 shows the indegree distribution of NEWSCAST and CYCLON overlays, as well as for a regular random graph. We consider graphs with 100,000 nodes, and view lengths 20 and 50, respectively. The vertical axis is in logarithmic scale. There is a clear difference between the two protocols. NEWSCAST exhibits a widely skewed distribution of incoming links centered around small values, with a tail reaching out to almost an order of magnitude higher values, unlike CYCLON, which imposes a bounded indegree in a self-regulated manner.

Self healing Finally, it is interesting to see how the two protocols perform at removing dangling links to dead nodes. We repeated the catastrophic failure experiment presented in Section 2.4, this time for NEWSCAST. In that experiment, we abruptly killed half of the nodes of an already converged overlay, and observed how long it takes the remaining nodes to "forget" the dead ones.

Figure 2.13 shows the number of dead nodes still remembered and the number of links to them as a function of the cycles elapsed since the catastrophic failure. CYCLON graphs (enhanced swapping) shown in Figure 2.8 are reproduced here for comparison. NEWSCAST's priority at keeping the *freshest* pointers to neighbors becomes evident by the speed at which dead nodes are forgotten.

The above illustrate the high dependence of protocol behavior on specific design choices. A comprehensive exploration of the design space, including CY-

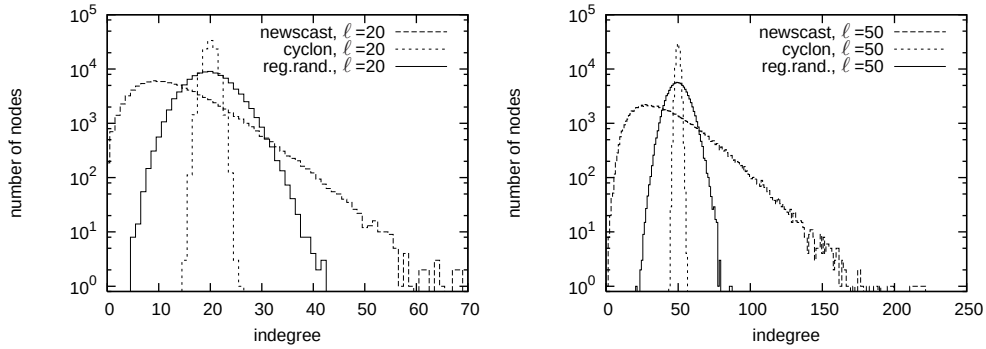
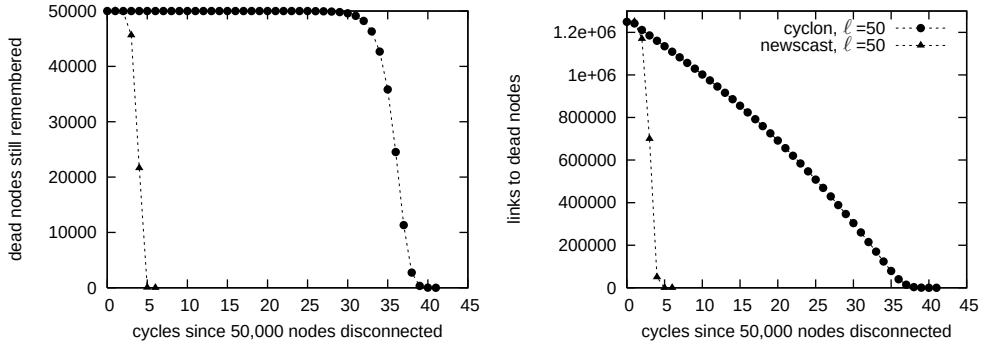


Figure 2.12: Indegree distribution in converged 100,000 node overlay, for NEWSCAST, CYCLON (enhanced swapping), and a regular random graph of outdegree ℓ.



(a) (b)

Figure 2.13: (a) Time until dead nodes are forgotten. (b) Number of dead links.

CLON and NEWSCAST among others, and its effect on overlay properties follows in Chapter 3.

2.9. AN APPLICATION: AGGREGATION

To illustrate the utilization of CYCLON overlays, let us now take a look at an example gossip-based application. *Aggregation*, that is, the collective estimation of system-wide properties, is a key functionality to a number of large-scale distributed systems. Such properties may refer to the network size, average system load, node with the lowest or highest load, aggregate capacity in a distributed storage system, etc. Aggregation should be carried out collectively by all participating nodes in a purely distributed fashion, and the results should become known to all nodes.

Aggregation is not the focus of this dissertation. For more information the reader is directed to work such as [Gupta et al. 2001], [Kempe et al. 2001], [Kempe et al. 2003], [Renesse et al. 2003], [Jelasity et al. 2005], [Jelasity and Montresor 2004], [Montresor et al. 2004], and [Kowalczyk and Vlassis 2004]. In this section we adopt the basic aggregation protocol followed in all aforementioned publications, and we show how it can benefit from CYCLON's properties.

We consider a basic aggregation protocol that follows the push-pull gossip-based paradigm, appearing in [Jelasity et al. 2005]. Each node is assumed to have a local *estimate* of the property being aggregated and a set of *neighbors*. At random times, but exactly once every δ time units, a node picks a random neighbor and they exchange their estimates. Subsequently, each node updates its local estimate based on its previous value and the estimate received.

Averaging constitutes a fundamental aggregation operation, in which each node is equipped with a numeric value, and the goal is to estimate the average of all nodes' values. Jelasity et al. [Jelasity et al. 2005] show how averaging can be used as the basis for the computation of other aggregates, including generalized mean, variance, counting of nodes, sum, and product.

In averaging, a node updates its estimate to the average between its previous local estimate and the estimate received. That is, when nodes p and q with estimates s_p and s_q gossip, their estimates are updated as follows:

$$s_p = s_q = \frac{s_p + s_q}{2}$$

Note that the sum of the two nodes' estimates does not change, Therefore neither does the global average. The variance, however, over the set of all nodes' estimates decreases, unless s_p and s_q were already equal, in which case it remains unaltered. Experiments and theoretical analysis in [Jelasity et al. 2005; Jelasity

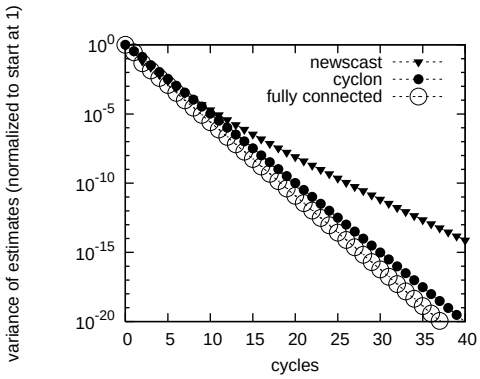


Figure 2.14: Aggregation in static overlays.

and Montresor 2004; Montresor et al. 2004; Kowalczyk and Vlassis 2004] show that the variance converges to zero. Moreover, it converges at an exponential rate, whose exponent depends on the communication graph defining the nodes' neighbors. The rule of thumb is that the higher the link randomization in an overlay, the faster the aggregation convergence.

Here we evaluate averaging over CYCLON and NEWSCAST overlays, in one *static* and two *dynamic* settings. All presented experiments involve 100,000 nodes, and a view length of 20.

Aggregation over static overlays In our first experiment setting, we applied averaging over converged, static CYCLON and NEWSCAST overlays. By *static* we mean that, after an overlay had converged, we ceased gossiping and studied aggregation over its—static—links. Figure 2.14 shows the evolution of the variance as a function of the aggregation cycles (δ time units) elapsed. To have a point of reference, we also plot the variance evolution for averaging over a fully connected graph, in which a node exchanges estimates with a node picked randomly out of the whole network. Fully connected graphs demonstrated the fastest convergence among all overlays tested in [Jelasity et al. 2005].

First, we observe that in all cases the variance converges to zero at an exponential rate. Second, we record a clear difference between the aggregation efficiency of static CYCLON and NEWSCAST overlays, the former converging significantly faster. This is a direct consequence of CYCLON's narrow indegree distribution and very low clustering. Each node has roughly the same number of incoming links, and therefore participates in roughly the same number of estimation exchanges as all other nodes. Moreover, the very low clustering ensures that each node's initial value is uniformly spread across "all directions" of the network, not being con-

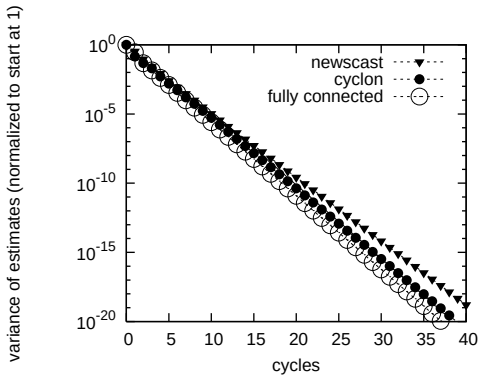


Figure 2.15: Aggregation in dynamic overlays.

fined to any highly clustered subset. This leads to faster convergence of nodes' estimates to the global average. On the other hand, NEWSCAST's skewed indegree distribution results in an uneven distribution of estimation exchanges across nodes. Also, due to high clustering, local estimates spread quickly within highly clustered communities, but take longer to spread globally.

Aggregation over dynamic overlays Although sufficiently random static topologies are good enough for aggregation, they are not adequate for dynamic environments. In real world systems, a dynamic overlay management protocol —such as CYCLON or NEWSCAST— is needed to deal with network changes. Consequently, in our second experiment we evaluated averaging over *dynamic* overlays. More specifically, each node was running an overlay management protocol (CYCLON or NEWSCAST) *and* the averaging protocol. Both overlay management and averaging protocols were running at the same frequency, that is, each one executed at a random time (not necessarily together) but exactly once every δ time units. The node to exchange averaging estimates with, was picked randomly out of the overlay management protocol's current view.

Figure 2.15 presents the variance evolution for CYCLON, NEWSCAST, and a fully connected graph of 100,000 nodes. Clearly, both protocols perform better compared to their static counterparts. The gain is larger for NEWSCAST, as its dynamic execution significantly increases "randomness" in nodes' views. CYCLON, whose static views already resembled regular random graph edges, does not gain as much, nevertheless it still outperforms NEWSCAST.

In further experiments (not shown here) we observed that when the overlay management protocol runs *faster* than the averaging one, even more randomness is introduced in both protocols, gradually approaching the fully connected graph

behavior. In fact, when the overlay management protocol runs at an order of 10 times faster than averaging, both protocols effectively converge like a fully connected graph. The tradeoff for this—better—randomness and faster aggregation is the higher gossiping frequency for overlay management, resulting in more network traffic.

Aggregation piggybacked on overlay management Our third set of experiments is based on a scheme combining the aggregation algorithm with the overlay management protocol. The idea is to *piggyback* estimation exchanges on the view exchange messages used for overlay management. The evident benefit of this combination is that aggregation comes for free: aggregation messages are replaced by a few extra bytes in overlay management messages.

In the case of CYCLON a more significant—albeit less obvious—benefit applies. CYCLON has proved to trigger highly even communication among nodes. That is, in each cycle each node initiates gossiping *exactly once* with a *random* node, and is contacted for gossiping by another *random* node *almost exactly once*. When piggybacked on CYCLON, aggregation adopts this highly *even* and *randomized* communication pattern, further improving its convergence rate.

On the contrary, piggybacking aggregation on NEWSCAST has a negative effect. In NEWSCAST, two nodes that gossip end up with the same view (see Section 2.8.1). If these two nodes also engage in aggregation exchange, the (negative) effect of clustering is maximized: The two fresh estimates are confined in two nodes belonging to the exact same cluster, hindering their fast diffusion to distant nodes in the network.

The results of the piggybacked setting are shown in Figure 2.16. Note that averaging piggybacked on CYCLON converges as fast as on a fully connected graph. On the other hand, we can clearly see the negative effect of piggybacking on NEWSCAST, as explained above.

To sum up our findings, we conclude that CYCLON constitutes a very attractive overlay management protocol for aggregation, either for static or dynamic environments.

Although aggregation is not one of the core applications presented in this dissertation, it allowed us briefly illustrate how applications can benefit from CYCLON's particular properties. A number of additional applications based on CYCLON are extensively presented and evaluated in Part II.

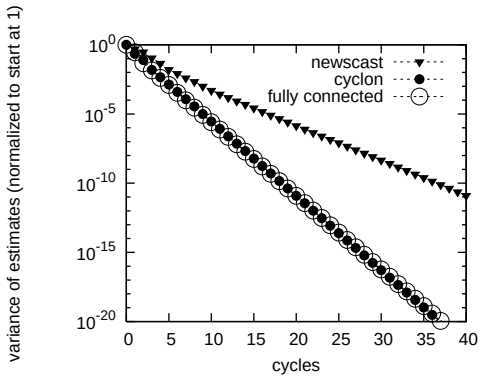


Figure 2.16: Aggregation in piggy-backed version.

2.10. RELATED WORK

There are currently several efforts in constructing unstructured overlays that share properties with random graphs. We have already described basic swapping, introduced in [Stavrou et al. 2004], forming the starting point for our own work described in this chapter.

Another example is the Scamp protocol [Ganesh et al. 2003], which is explicitly designed to construct overlays with evenly distributed links, achieving view lengths of $O(\log N)$ without prior knowledge of N. Scamp is reactive, in the sense that view exchanges take place only when nodes join, leave, or a failure is detected. As it turns out, the protocol exhibits similar properties in comparison to random graphs when considering its capabilities for information dissemination and recovering from massive node failures. However, no thorough analysis has been undertaken to compare the communication graph with random graphs as we did, but it is known that there are important differences. A proactive extension of Scamp is suggested in [Massoulie et al. 2003], improving the protocol's behavior with respect to load balancing and resilience to failures in dynamic settings.

In many unstructured overlays, such as CYCLON, scalability of the network is achieved by maintaining a partial view on the entire network. The construction of the network itself, that is, membership management, is crucial as we have argued in this chapter. It is interesting to see that the assumption is sometimes made that the communication graph resulting from a specific membership protocol is random. However, as we will see in Chapter 3, there is a large family of membership protocols for unstructured overlays for which this assumption is false. This includes the work on Lightweight Probabilistic Multicast [Eugster et al. 2003b], as well as the NEWSCAST protocol [Jelasity et al. 2003; Voulgaris et al. 2003].

As it turns out, such membership protocols generally lead to small-world graphs, which distinguish themselves from random graphs by a high clustering coefficient. In contrast, CYCLON overlays appear to fall outside the category of small-world networks.

Of course, random graphs may not be the best structure for communication networks. Alternative schemes are described in [Pandurangan et al. 2003; Law and Sui 2003]. In these cases, the requirements of low diameter, resilience to massive node failures, and inexpensive membership management lead to specific graph-construction protocols. We argue that with enhanced swapping these requirements are all met, and that the resulting communication graphs allow us to adopt the rigorous analysis of random graphs. Moreover, in contrast to, for example [Pandurangan et al. 2003], there is no need to use a central server.

In this light, it is also interesting to mention the recent work on Phenix for the decentralized construction of low-diameter, scale-free networks [Wouhaybi and Campbell 2004]. In Phenix, an additional goal is to construct networks that are resilient to massive malicious attacks. We have not yet examined this feature for CYCLON, but suspect its good randomness properties will help in also making it attack resilient.

Finally, it is interesting to discuss Kelips [Gupta et al. 2003], [Linga et al.], a DHT based on gossiping. Kelips imposes a fixed structure on a network of N nodes, organizing them in $\sqrt{N}$ groups of $\sqrt{N}$ nodes each. Each node maintains a complete view of its group ($O(\sqrt{N})links$), and at least one link to each other group ($O(\sqrt{N})$ links). Routing is, therefore, performed at constant time, namely in two steps: in the first step a message is routed to the appropriate group, in the second it reaches the target node within that group. Gossiping is applied to spread membership information within and across groups. Old links are discarded based on their age.

The main philosophy behind Kelips is trading per-node memory requirements for constant time lookups. Indeed, increased memory requirements pose no serious constraints on contemporary memory-abundant computer systems. However, another facet of this design decision has been overlooked. The larger a view is, the harder it becomes to keep it up-to-date in the face of node churn and network changes, leading to incomplete or inaccurate views. It is therefore unclear whether the constant time lookup premise of Kelips is maintained when the scale of the system increases and membership becomes highly dynamic. However, the concept of discarding links solely based on their age, rather than due to a predefined, fixed view length (as in CYCLON), seems highly promising and deserves further research. It can lead to the design of a family of gossiping protocols (of which Kelips is an instance), different than the protocols dealt with in this dissertation.

2.11. CONCLUSIONS AND FUTURE WORK

In this chapter we presented CYCLON, a complete framework for inexpensive membership management in very large P2P overlays. CYCLON is highly scalable, very robust, and completely decentralized. Most important is that the resulting communication graphs share important properties with random graphs. Besides the fact that desirable features such as low diameter and robustness are supported, this similarity justifiably opens the possibility to rigorously analyze the networks.

We also conclude that CYCLON is an improvement of the basic swapping protocol developed by Stavrou et al. [Stavrou et al. 2004]. We offer a scalable and inexpensive membership protocol, achieve better node-degree distributions, and significantly lower the pollution of views concerning stale references to previous members.

In addition to our continued pursuit for new applications of CYCLON, we will also explore potential improvements of the protocol itself. An important next step in our research will be the replacement of the periodic view exchanges with a reactive exchange protocol, as in Scamp [Ganesh et al. 2003]. We envisage that this replacement will lead to a better utilization of network resources, and incur only minimal costs for detecting failed nodes. Another important subject that we will address is taking network proximity into account. The latter will be largely based on our scalable latency estimation service, described in [Szymaniak et al. 2004].

CHAPTER 3

Random Overlays: Exploring the Design Space

This chapter is a paper under submission (extending [Jelasity et al. 2004a]):
Mark Jelasity, Spyros Voulgaris, Anne-Marie Kermarrec, Rachid Guerraoui, Maarten van Steen
"Gossip-based Peer Sampling"

Gossip-based communication protocols are appealing in large-scale distributed applications such as information dissemination, aggregation, and overlay topology management. In this chapter we factor out a fundamental mechanism at the heart of all these protocols: the PEER SAMPLING SERVICE. In short, this service provides every node with peers to gossip with. We promote this service at the level of a first class abstraction of a large-scale distributed system, pretty much like a name service is a first class abstraction of a local-area system. We present a generic framework to implement a PEER SAMPLING SERVICE in a decentralized manner by constructing and maintaining *dynamic unstructured* overlays through gossiping membership information itself. Our framework generalizes existing approaches and makes it easy to discover new ones. We use this framework to empirically explore and compare several implementations of the PEER SAMPLING SERVICE including existing gossiping approaches. Through extensive simulation experiments we show that—although all protocols provide a good quality uniform random stream of peers to each node locally—traditional theoretical assumptions about the randomness of the unstructured overlays as a whole do not hold in any of the instances. We also show that different design decisions result in severe differences from the point of view of two crucial aspects: load balancing and fault

tolerance. Our simulations are validated through a real implementation over a wide-area cluster.

3.1. INTRODUCTION

Gossip-based protocols, also called epidemic protocols, are appealing in large-scale distributed applications. The popularity of these protocols stems from their ability to reliably pass information among a large set of interconnected nodes, even if the nodes regularly join and leave the system (either purposefully or on account of failures), and the underlying network suffers from broken or slow links.

The common interaction pattern underlying gossip-based protocols consists in each node in the system periodically exchanging information with a subset of its peers. The choice of this subset is crucial to the wide dissemination of the gossip. Ideally, any given node should exchange information with peers that are selected following a uniform random sample of *all nodes* currently in the system [Demers et al. 1987; van Renesse et al. 1998; Birman et al. 1999; Karp et al. 2000; Sun and Sturman 2000; Kowalczyk and Vlassis 2004]. This assumption has led to rigorously establish many desirable features of gossip-based protocols like scalability, reliability, and efficiency (see, e.g., [Pittel 1987] in the case of information dissemination, or [Kempe et al. 2003; Jelasity et al. 2005] for aggregation).

In practice, enforcing this assumption would require to develop applications where each node may be assumed to know of every other node in the system [Birman et al. 1999; Gupta et al. 2002; Kermarrec et al. 2003]. However, providing each node with a complete membership table from which a random sample can be drawn, is unrealistic in a large-scale dynamic system, for maintaining such tables in the presence of joining and leaving nodes (referred to as *churn*) incurs considerable synchronization costs. In particular, measurement studies on various peer-to-peer networks indicate that an individual node may often be connected in the order of only a few minutes to an hour (see, e.g. [Bhagwan et al. 2003; Saroiu et al. 2003; Sen and Wang 2004]).

Clearly, decentralized schemes to maintain membership information are crucial to the deployment of gossip-based protocols. This chapter factors out the abstraction of a PEER SAMPLING SERVICE and presents a generic, yet simple, gossip-based framework to implement it.

The PEER SAMPLING SERVICE is singled out from the application using it and, abstractly speaking, the same service can be used in different settings: information dissemination [Demers et al. 1987; Eugster et al. 2004], aggregation [Kempe et al. 2003; Jelasity et al. 2005, 2004b; Montresor et al. 2004], load balancing [Jelasity et al. 2004c], membership management [Voulgaris et al. 2005],

and overlay bootstrapping [Jelasity et al. 2006; Voulgaris and van Steen 2003]. The service is promoted as a first class abstraction of a large-scale distributed system. In a sense, it plays the role of a naming service in a traditional LAN-oriented distributed system as it provides each node with other nodes to interact with.

The basic general principle underlying the framework we propose to implement the PEER SAMPLING SERVICE is itself based on a gossip paradigm. In short, every node (1) maintains a relatively small local membership table that provides a *partial view* on the complete set of nodes and (2) periodically refreshes the table using a gossiping procedure. The framework is generic and can be used to instantiate known [Eugster et al. 2003b; Jelasity et al. 2003; Voulgaris et al. 2005] and novel gossip-based membership implementations. In fact, our framework captures many possible variants of gossip-based membership dissemination. These variants mainly differ in the way the membership table is updated at a given peer after gossiping with communicating peers. We use this framework to experimentally evaluate various implementations and identify key design parameters in practical settings. Our experimentation covers both extensive simulations and emulations on a wide-area cluster.

We consider many dimensions when identifying qualitative differences between the variants we examine. These dimensions include the randomness of selecting a peer as perceived by a single node, the accuracy of the current membership view, the distribution of the load incurred on each node, as well as the robustness in the presence of failures and churn.

As it turns out, we find that two peers should exchange elements from their respective tables. In other words, communication should be bidirectional. Adhering to a push-only or pull-only approach can easily lead to (irrecoverable) partitioning of the set of nodes. Another finding is that robustness against failing nodes or churn can be easily accomplished if old table entries are dropped when exchanging membership information.

However, as we shall also see, no single implementation outperforms the others along all dimensions. In this study we identify these tradeoffs when selecting an implementation of the PEER SAMPLING SERVICE for a given application. For example, to achieve good load balancing, table entries should rather be *swapped* between two peers. However, this strategy is less robust against failures and churn than non-swapping ones.

3.2. THE PEER SAMPLING SERVICE

The PEER SAMPLING SERVICE is implemented over a set of nodes that form the domain of the gossip-based protocols that make use of the service. The same

sampling service can be utilized by multiple gossip protocols simultaneously, provided they have a common target group. The task of the service is to provide a participating node of a gossiping application with a subset of peers from the group to send gossip messages to.

3.2.1. API

The API of the PEER SAMPLING SERVICE simply consists of two methods: `init` and `getPeer`. It would be technically straightforward to provide a framework for a multiple-application interface and architecture. For a better focus and simplicity of notations we assume however that there is only one application. The specification of these methods is as follows.

`init()` Initializes the service on a given node if this has not been done before. The actual initialization procedure is implementation dependent.

`getPeer()` Returns a peer address if the group contains more than one node. The returned address is a sample drawn from the group. Ideally, this sample should be an independent unbiased random sample. The exact characteristics of this sample (e.g., its randomness or correlation in time and with other peers) is affected by the implementation.

The focus of the research presented in this chapter is to give accurate information about the behavior of the `getPeer` method in the case of a class of gossip-based implementations. Applications requiring more than one peer call this method repeatedly.

Note that we do not define a `stop` method. The reason is to ease the burden on applications by propagating the responsibility of automatically removing nonactive nodes to the service layer.

3.2.2. Generic Protocol Description

We consider a set of nodes connected in a network. A node has an address that is needed for sending a message to that node. Each node maintains a membership table representing its (partial) knowledge of the global membership. Traditionally, if this knowledge is complete, the table is called the *view*. However, in our case each node knows only a limited subset of the system, so the table is consequently called a *partial view*. The partial view is a list of ℓ *node descriptors*. Parameter ℓ is called the *view length* and is the same for all nodes.

A node descriptor contains a network address (such as an IP address) and an *age* that represents the freshness of the given node descriptor. The partial view is a list data structure, and accordingly, the usual list operations are defined on it.

```
do forever
  wait(T time units)
  p ← view.selectPeer()
  if push then
    // 0 is the initial age
    buffer ← ((MyAddress,0))
    view.permute()
    move oldest H items to end of view
    buffer.append(view.head(ℓ/2))
    send buffer to p
  else // empty view to trigger response
    send (null) to p
  if pull then
    receive buffer_p from p
  view.select(ℓ,H,S,buffer_p)
  view.increaseAge()
```

(a) active thread

```
do forever
  receive buffer_p from p
  if pull then
    // 0 is the initial age
    buffer ← ((MyAddress,0))
    view.permute()
    move oldest H items to end of view
    buffer.append(view.head(ℓ/2))
    send buffer to p
  view.select(ℓ,H,S,buffer_p)
  view.increaseAge()
```

(b) passive thread

```
method view.select(ℓ,H,S,buffer_p)
  view.append(buffer_p)
  view.removeDuplicates()
  view.removeOldItems( min(H,view.size-ℓ) )
  view.removeHead( min(S,view.size-ℓ) )
  view.removeAtRandom(view.size-ℓ)
```

(c) method view.select(ℓ,H,S,buffer$_p$)

Figure 3.1: The skeleton of a gossip-based implementation of the PEER SAMPLING SERVICE.

Most importantly, this means that the order of elements in the view is not changed unless some specific method (for example, `permute`, which randomly reorders the list elements) explicitly changes it. The protocol also ensures that there is at most one descriptor for the same address in every view.

The purpose of the gossiping algorithm, executed periodically on each node and resulting in two peers exchanging their membership information, is to make sure that the partial views contain descriptors of a continuously changing random subset of the nodes and (in the presence of failure and joining and leaving nodes) to make sure the partial views reflect the dynamics of the system. We assume that each node executes the same protocol of which the skeleton is shown in Figure 3.1. The protocol consists of two threads: an active (client) thread initiating communication with other nodes, and a passive (server) thread waiting for and answering these requests.

We now describe the behavior of the active thread. The passive thread just mirrors the same steps. The active thread gets activated in each T time units exactly once. Three globally known system-wide parameters are used in this algorithm:

parameters ℓ (the partial view length on each node), H and S. For the sake of clarity, we leave the details of the meaning and impact of H and S until the end of this section.

First, a peer node is selected to exchange membership information with. This selection is implemented by the method `selectPeer` that returns the address of a *live* node. This method is a parameter of the generic protocol. We discuss the implementations of `selectPeer` in Section 3.2.3.

Subsequently, if the information has to be pushed (boolean parameter `push` is true), then a buffer is initialized with a fresh descriptor of the node running the thread. Then, $\ell/2$ elements are appended to the buffer. The implementation ensures that these elements are selected randomly from the view without replacement, ignoring the oldest H elements (as defined by the age stored in the descriptors). If there are not enough elements in the view, then the oldest H elements are also sampled to fill in any remaining slots. In addition, the view will have exactly the selected elements as first items (i.e., in the list head). This fact will be important later. Parameter H is guaranteed to be less than or equal to $\ell/2$. The buffer created this way is sent to the selected peer.

If a reply is expected (boolean parameter `pull` is true) then the received buffer is passed to method `select(`ℓ`,H,S,buffer)`, which creates the new view based on the listed parameters, and the current view, making sure the size of the new view does not decrease and is at most ℓ.

Finally, method `select(`ℓ`,H,S,buffer)` creates the new view based on the listed parameters, and the current view as follows: After appending the received buffer to the view, it keeps only the freshest entry for each address. After this operation, there is at most one descriptor for each address. At this point, the size of the view is guaranteed to be at least the original size, since in the original view each address was included also at most once. Subsequently, the method performs a number of removal steps to decrease the size of the view to ℓ. The parameters to the removal methods are calculated in such a way that the view size never decreases under ℓ. First, the oldest items are removed, as defined by their age, and parameter H. The name H comes from *healing*, that is, this parameter defines how aggressive the protocol should be when it comes to removing links that potentially point to faulty nodes (dead links). Note, that in this way self-healing is implemented without actually checking if a node is alive or not. If a node is not alive, then its descriptors will never get refreshed (and thus become old), and therefore sooner or later get removed. The larger H is, the sooner older items will be removed from views.

After removing the oldest items, the S first items are removed from the view. Recall that it is exactly these items that were sent to the peer previously. As a result, parameter S controls the priority that is given to the addresses received

from the peer. If S is high, then the received items will have a higher probability to be included in the new view. Since the same algorithm is run on the receiver side, this mechanism in fact controls the number of items that are *swapped* between the two peers, hence the name S for the parameter. This parameter controls the diversity of the union of the two new views (on the passive and active side). If S is low then both parties will keep many of their exchanged elements, effectively increasing the similarity between the two respective views. As a result, more unique addresses will be removed from the system. In contrast, if S is high, then the number of unique addresses that are lost from both views is lower. The last step removes random items to reduce the size of the view to ℓ.

This framework captures the essential behavior of many existing gossip membership protocols (although exact matches often require small changes). As such, the framework serves two purposes: (1) we can use it to compare and evaluate a wide range of different gossip membership protocols by changing parameter values, and (2) it can serve as a unifying implementation for a large class of protocols. As a next step, we will explore the design space of our framework, forming the basis for an extensive protocol comparison.

3.2.3. Design Space

In this section we describe a set of specific instances of our generic protocol by specifying the values of the key parameters. These instances will be analyzed in the rest of the chapter.

Peer Selection As described before, peer selection is implemented by `selectPeer()` that returns the address of a *live* node as found in the caller's current view. In this study, we consider the following *peer selection* policies:

rand	Uniform randomly select an available node from the view
tail	Select the node with the *highest* age

Note that the third logical possibility of selecting the node with the *lowest* age is not included since this choice is not relevant. It is immediately clear from simply considering the protocol scheme that node descriptors with a low age refer to neighbors that have a view that is strongly correlated with the node's own view. More specifically, the node descriptor with the lowest age always refers exactly to the last neighbor the node communicated with. As a result, contacting this node offers little possibility to update the view with unknown entries, so the resulting overlay will be very static. Our preliminary experiments fully confirm this simple reasoning. Since the goal of peer sampling is to provide uncorrelated random peers continuously, it makes no sense to consider any policies with a bias towards low age, and thus protocols that follow such a policy.

View propagation Once a peer has been chosen, the peers may exchange information in various ways. We consider the following two *view propagation* policies:

push	The node sends descriptors to the selected peer
pushpull	The node and selected peer exchange descriptors

Like in the case of the view selection policies, one logical possibility: the *pull* strategy, is omitted. It is easy to see that the pull strategy cannot possibly provide satisfactory service. The most important flaw of the pull strategy is that a node cannot inject information about itself, except only when explicitly asked by another node. This means that if a node loses all its incoming connections (which might happen spontaneously even without any failures, and which is rather common as we shall see) there is no possibility to reconnect to the network.

View selection The parameters that determine how view selection is performed are H, the self-healing parameter, and S, the swap parameter. Let us first note some properties of these parameters. First, assuming that ℓ is even, all values of H for which $H > \ell/2$ are equivalent to $H = \ell/2$, because the protocol never decreases the view size under ℓ. For the same reason, all values of S for which $S > \ell/2 - H$ are equivalent to $S = \ell/2 - H$. Furthermore, the last, random removal step of the view selection algorithm is executed only if $S < \ell/2 - H$. Keeping this in mind, we have a "triangle" of protocols with H ranging from 0 to $\ell/2$, and with S ranging from 0 to $\ell/2 - H$. In our analysis we will look at this triangle at different resolutions, depending on the scenarios in question. As a minimum, we will consider the three vertices of the triangle defined as follows.

blind	$H = 0, S = 0$	Select blindly a random subset
healer	$H = \ell/2$	Select the freshest entries
swapper	$H = 0, S = \ell/2$	Minimize loss of information

3.2.4. Implementation

We now describe a possible implementation of the PEER SAMPLING SERVICE API based on the framework presented in Section 3.2.2. We assume that the service forms a layer between the application and the unstructured overlay network.

Initialization The `init` method will cause the service to register itself with the gossiping protocol instance that maintains the overlay network. From that point, the service will be notified by this instance whenever the actual view is updated.

Sampling As an answer to the `getPeer` call, the service returns an element from the *current* view. To maximize the diversity of the returned peers, the service makes a best effort not to return the same element twice during the period while the given element is in the view: this would introduce an obvious bias and would damage randomness. To achieve this, the service maintains a queue of elements that are currently in the view but have not been returned yet. Method `getPeer` returns the first element from the queue and subsequently it removes this element from the queue. When the service receives a notification on a view update, it removes those elements from the queue that are no longer in the current view, and appends the new elements that were not included in the previous view. If the queue becomes empty, the service falls back on returning random samples from the current view. In this case the service can set a warning flag that can be read by applications to indicate that the quality of the returned samples is no longer reliable.

In the following sections, we analyze the behavior of our framework in order to gradually come to various optimal settings of the parameters. Anticipating our discussion in Section 3.7, we will show that there are some parameter values that never lead to good results (such as selecting a peer from a fresh node descriptor). However, we will also show that no single combination of parameter values is always best and that, instead, tradeoffs need to be made.

3.3. LOCAL RANDOMNESS

Ideally, a PEER SAMPLING SERVICE should return a series of unbiased independent random samples from the current group of peers. The assumption of such randomness has indeed led to rigorously establish many desirable features of gossip-based protocols like scalability, reliability, and efficiency [Pittel 1987].

When evaluating the quality of a particular implementation of the service, one faces the methodological problem of characterizing randomness. In this section we consider a fixed node and analyze the series of samples generated at that particular node.

There are essentially two ways of capturing randomness. The first approach is based on the notion of Kolmogorov complexity [Li and Vitányi 1997]. Roughly speaking, this approach considers as random any series that cannot be compressed. Pseudo random number generators are automatically excluded by this definition, since any generator, along with a random seed, is a compressed representation of a series of any length. Sometimes it can be proven that a series *can* be compressed, but in the general case, the approach is not practical to test randomness due to the difficulty of proving that a series *cannot* be compressed.

The second, more practical approach assumes that a series is random if any statistic computed over the series matches the theoretical value of the same statistic under the assumption of randomness. The theoretical value is computed in the framework of probability theory. This approach is essentially empirical, because it can never be mathematically proven that a given series is random. In fact, good pseudo random number generators pass most of the randomness tests that belong to this category.

Following the statistical approach, we view the PEER SAMPLING SERVICE (as seen by a fixed node) as a random number generator, and we apply the same traditional methodology that is used for testing random number generators. We test our implementations with the "diehard battery of randomness tests" [Marsaglia 1995], the *de facto* standard in the field.

3.3.1. Experimental Settings

We have experimented our protocols using the PeerSim simulator [PeerSim]. All the simulation results were obtained using this implementation.

The `diehard` test suite requires as input a considerable number of 32-bit integers: the most expensive test needs $6 \cdot 10^7$ of them. To be able to generate this input, we assume that all nodes in the network are numbered from 0 to N. Node N executes the PEER SAMPLING SERVICE, obtaining one number between 0 and $N-1$ each time it calls the service, thereby generating a sequence of integers. If N is of the form $N = 2^n + 1$, then the bits of the generated numbers form an unbiased random bit stream, provided the PEER SAMPLING SERVICE returns random samples.

Due to the enormous cost of producing a large number of samples, we restricted the set of implementations of the view construction procedure to the three extreme points: `blind`, `healer` and `swapper`. Peer selection was fixed to be `tail` and `pushpull` was fixed as the communication model. Furthermore, the network size was fixed to be $2^{10} + 1 = 1025$, and the view length was $\ell = 20$. These settings allowed us to complete $2 \cdot 10^7$ cycles for all the three protocol implementations. In each case, node N generated four samples in each cycle, thereby generating four 10-bit numbers. Ignoring two bits out of these ten, we generated one 32-bit integer for each cycle.

Experiments convey the following facts. No matter which two bits are ignored, it does not affect the results, so we consider this as a non-critical decision. Note that we could have generated 40 bits per cycle as well. However, since many tests in the `diehard` suit do respect the 32-bit boundaries of the integers, we did not want to artificially diminish any potential periodic behavior in terms of the cycles.

3.3.2. Test Results

A detailed description of the tests in the `diehard` benchmark is out of the scope of this dissertation. In Table 3.1 we summarize the basic ideas behind each class of tests. In general, the three random number sequences pass all the tests, including the most difficult ones [Marsaglia and Tsang 2002], with one exception. Before discussing the one exception in more detail, note that for two tests we did not have enough 32-bit integers, yet we could still apply them. The first case is the permutation test, which is concerned with the frequencies of the possible orderings of 5-tuples of subsequent random numbers. The test requires $5 \cdot 10^7$ 32-bit integers. However, we applied the test using the original 10-bit integers returned by the sampling service, and the random sequences passed. The reason is that ordering is not sensitive to the actual range of the values, as long as the range is not extremely small. The second case is the so called "gorilla" test, which is a strong instance of the class of the monkey tests [Marsaglia and Tsang 2002]. It requires $6.7 \cdot 10^7$ 32-bit integers. In this case we concatenated the output of the three protocols and executed the test on this sequence, with a positive result. The intuitive reasoning behind this approach is that if any of the protocols produces a non-random pattern, then the entire sequence is supposed to fail the test, especially given that this test is claimed to be extremely difficult to pass.

Consider now the test that proved to be difficult to pass. This test was an instance of the class of binary matrix rank tests. In this instance, we take 6 consecutive 32-bit integers, and select the same (consecutive) 8 bits from each of the 6 integers forming a 6×8 binary matrix whose rank is determined. That rank can be from 0 to 6. Ranks are found for 100,000 random matrices, and a chi-square test is performed on counts for ranks smaller or equal to 4, and for ranks 5 and 6.

When the selected byte coincides with the byte contributed by one call to the PEER SAMPLING SERVICE (bits 0-7, 8-15, etc), protocols `blind` and `swapper` fail the test. To better see why, consider the basic functioning of the rank test. In most of the cases, the rank of the matrix is 5 or 6. If it is 5, it typically means that the same 8-bit entry is copied twice into the matrix. Our implementation of the PEER SAMPLING SERVICE explicitly ensures that the diversity of the returned elements is maximized in the short run (see Section 3.2.4). As a consequence, rank 6 occurs relatively more often than in the case of a true random sequence. Note that for many applications this property is actually an advantage. However, `healer` passes the test. The reason of this will become clearer in the remaining parts of this chapter. As we will see, in the case of `healer` the view of a node changes faster and therefore the queue of the samples to be returned is frequently flushed, so the diversity-maximizing effect is less significant.

The picture changes if we consider only every 4th sample in the random sequence generated by the protocols. In that case, `blind` and `swapper` pass the

Birthday Spacings	The k-bit random numbers are interpreted as "birthdays" in a "year" of 2^k days. We take m birthdays and list the spacings between the consecutive birthdays. The statistic is the number of values that occur more than once in that list.
Greatest Common Divisor	We run Euclid's algorithm on consecutive pairs of random integers. The number of steps Euclid's algorithm needs to find the greatest common divisor (GCD) of these consecutive integers in the random series, and the GCD itself are the statistics used to test randomness.
Permutation	Tests the frequencies of the $5! = 120$ possible orderings of consecutive integers in the random stream.
Binary Matrix Rank	Tests the rank of binary matrices built from consecutive integers, interpreted as bit vectors.
Monkey	A set of tests for verifying the frequency of the occurrences of "words" interpreting the random series as the output of a monkey typing on a typewriter. The random number series is interpreted as a bit stream. The "letters" that form the words are given by consecutive groups of bits (i.e., for 2 bits there are 4 letters, etc).
Count the 1-s	A set of tests for verifying the number of 1-s in the bit stream.
Parking Lot	Numbers define locations for "cars." We continuously "park cars" and test the number of successful and unsuccessful attempts to place a car at the next location defined by the random stream. An attempt is unsuccessful if the location is already occupied (the two cars would overlap).
Minimum Distance	Integers are mapped to two or three dimensional coordinates and the minimal distance among thousands of consecutive points is used as a statistic.
Squeeze	After mapping the random integers to the interval $[0,1)$, we test how many consecutive values have to be multiplied to get a value smaller than a given threshold. This number is used as a statistic.
Overlapping Sums	The sum of 100 consecutive values is used as a statistic.
Runs Up and Down	The frequencies of the lengths of monotonously decreasing or increasing sequences are tested.
Craps	200,000 games of craps are played and the number of throws and wins are counted. The random integers are mapped to the integers $1,\ldots,6$ to model the dice.

Table 3.1: Summary of the basic idea behind the classes of tests in the `diehard` test suite for random number generators. In all cases tests are run with several parameter settings. For a complete description we refer to [Marsaglia 1995].

test, but `healer` fails. In this case, the reason of the failure of `healer` is exactly the opposite: there are relatively too many repetitions in the sequence. Taking only every 8th sample, all protocols pass the test.

Finally, note that even in the case of "failures," the numeric deviation from random behavior is rather small. The expected occurrences of ranks of $\leq$4, 5 and 6 are 0.94%, 21.74% and 77.31%, respectively. In the first type of failure, when there are too many occurrences of rank 6, a typical failed test gives percentages 0.88%, 21.36% and 77.68%. When ranks are too small, a typical failure is, for example, 1.05%, 21.89% and 77.06%.

3.3.3. Conclusions

The results of the randomness tests suggest that the stream of nodes returned by the PEER SAMPLING SERVICE is close to random for all the protocol instances examined. Given that some widely used pseudo random number generators fail at least some of these tests, this is a highly encouraging result regarding the quality of the randomness provided by this class of sampling protocols.

Based on these experiments, we cannot, however, conclude on global randomness of the resulting graphs. Local randomness, evaluated from a peer's point of view is important, however, in a complex large-scale distributed system, where the stream of random nodes returned by the nodes might have complicated correlations, merely looking at local behavior does not reveal some key characteristics such as load balancing (existence of bottlenecks) and fault tolerance. In Section 3.4 we present a detailed analysis of the global properties of our protocols.

3.4. GLOBAL RANDOMNESS

In Section 3.3 we have seen that from a local point of view all implementations produce good quality random samples. However, statistical tests for randomness and independence tend to hide important *structural* properties of the system *as a whole*. To capture these global correlations, in this section we switch to a graph theoretical framework. To translate the problem into a graph theoretical language, we consider the *communication topology* or *overlay topology* defined by the set of nodes and their views (recall that `getPeer()` returns samples from the view). In this framework the directed edges of the communication graph are defined as follows. If node a stores the descriptor of node b in its view then there is a directed edge (a,b) from a to b. In the language of graphs, the question is how similar this overlay topology is to a random graph in which the descriptors in each view represent a uniform independent random sample of the whole node set.

In this section we consider graph-theoretic properties of the overlay graphs.

An important example of such properties is the *degree distribution*. The indegree of node *i* is defined as the number of nodes that have *i* in their views. The outdegree is constant and equal to the view size ℓ. Degree distribution has many significant effects. Most importantly, degree distribution determines whether there are hot spots and bottlenecks from the point of view of communication costs. In other words, *load balancing* is determined by the degree distribution. It also has a direct relationship with reliability to different patterns of node failures [Albert et al. 2000], and has an effect on the exact way epidemics are spread [Pastor-Satorras and Vespignani 2001]. Apart from the degree distribution we also analyze the clustering coefficient and average path length, as described and motivated in Section 3.4.2.

The main goal of the work presented in this chapter is to explore the different design choices in the protocol space described in Section 3.2.2. More specifically, we want to assess the impact of the peer selection, view selection, and view propagation parameters. Accordingly, we chose to fix the network size to $N = 10^4$ and the maximal view size to $\ell = 30$. The results presented in this section were obtained using the PeerSim simulation environment [PeerSim].

3.4.1. Properties of Degree Distribution

The first and most fundamental question is whether, for a particular protocol implementation, the communication graph has some stable properties, which it maintains during the execution of the protocol. In other words, we are interested in the *convergence behavior* of the protocols. We can expect several sorts of dynamics which include chaotic behavior, oscillations, and convergence. In case of convergence the resulting state may or may not depend on the initial configuration of the system. In the case of overlay networks we obviously prefer to have convergence towards a state that is independent of the initial configuration. This property is called *self-organization*. In our case it is essential that in a wide range of scenarios the protocol instances should automatically produce consistent and predictable behavior. Section 3.4.1 examines this question.

A related question is whether there is convergence and what kind of communication graph a protocol instance converges to. In particular, as mentioned earlier, we are interested in what sense overlay topologies deviate from certain random graph models. We discuss this issue in Section 3.4.1.

Finally, we are interested in looking at *local dynamic* properties along with *globally stable* degree distributions. That is, it is possible that while the overall degree distribution and its global properties such as maximum, variance, average, etc., do not change, the degree of the individual nodes does. This is preferable because in this case even if there are always bottlenecks in the network, the bottleneck will not be the same node all the time which greatly increases robustness

and improves load balancing. Section 3.4.1 is concerned with these questions.

Convergence

We now present experimental results that illustrate the convergence properties of the protocols in three different bootstrapping scenarios:

Growing In this scenario, the overlay network initially contains only one node. At the beginning of each cycle, 500 new nodes are added to the network until the maximal size is reached in cycle 20. The view of these nodes is initialized with only a single node descriptor, which belongs to the oldest, initial node. This scenario is the most pessimistic one for bootstrapping the overlays. It would be straightforward to improve it by using more contact nodes, which can come from a fixed list or which can be obtained using inexpensive local random walks on the existing overlay. However, in our discussion we intentionally avoid such optimizations to allow a better focus on the core protocols and their differences.

Lattice In this scenario, the initial topology of the overlay is a ring lattice, a structured topology. We build the ring lattice as follows. The nodes are first connected into a ring in which each node has a descriptor in its view that belongs to its two neighbors in the ring. Subsequently, for each node, additional descriptors of the nearest nodes are added in the ring until the view is filled.

Random In this scenario the initial topology is defined as a random graph, in which the views of the nodes were initialized by a uniform random sample of the peer nodes.

As we focus on the dynamic properties of the protocols, we did not wish to average out interesting patterns, so in all cases the result of a single run is shown in the plots. Nevertheless, we ran all the scenarios 100 times to gain data on the stability of the protocols with respect to the connectivity of the overlay. Connectivity is a crucial feature, a minimal requirement for all applications. The results of these runs show that in all scenarios, every protocol under examination creates a connected overlay network in 100% of the runs (as observed in cycle 300). The only exceptions were detected during the growing overlay scenario. Table 3.2 shows the push protocols. With the push-pull scheme we have not observed partitioning.

The push versions of the protocols perform very poorly in the growing scenario in general. Figure 3.2 illustrates the evolution of the maximal indegree. The maximal indegree belongs to the central contact node that is used to bootstrap the

protocol	partitioned runs	average number of clusters	average largest cluster
(rand,healer,push)	100%	22.28	9124.48
(rand,swapper,push)	0%	n.a.	n.a.
(rand,blind,push)	18%	2.06	9851.11
(tail,healer,push)	29%	2.17	9945.21
(tail,swapper,push)	97%	4.07	9808.04
(tail,blind,push)	10%	2.00	9936.20

Table 3.2: Partitioning of the push protocols in the growing overlay scenario. Data corresponds to cycle 300. Cluster statistics are over the partitioned runs only.

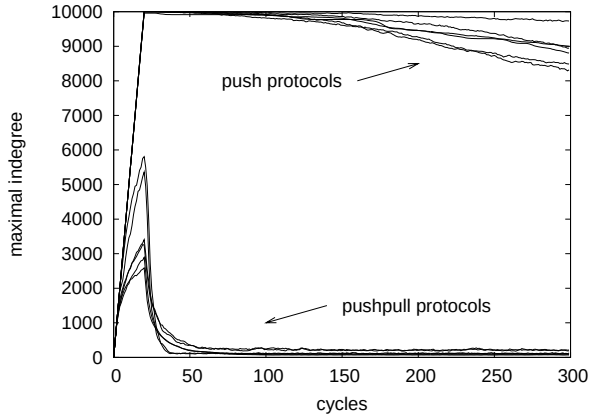


Figure 3.2: Evolution of maximal indegree in the growing scenario (recall that growing stops in cycle 20). The runs of the following protocols are shown: peer selection is either rand or tail, view selection is blind, healer or swapper, and view propagation is push or pushpull.

network. After growing is finished in cycle 20, the push-pull protocols almost instantly balance the degree distribution thereby removing the bottleneck. The push versions, however, get stuck in this unbalanced state.

This is not surprising, because when a new node joins the network and gets an initial contact node to start with, the only way it can get an updated view is if some other node contacts it actively. This, however, is very unlikely. Because all new nodes have the same contact, the view at the contact node gets updated extremely frequently causing all the joining nodes to be quickly forgotten. A node has to push its own descriptor many times until some other node actually contacts it.

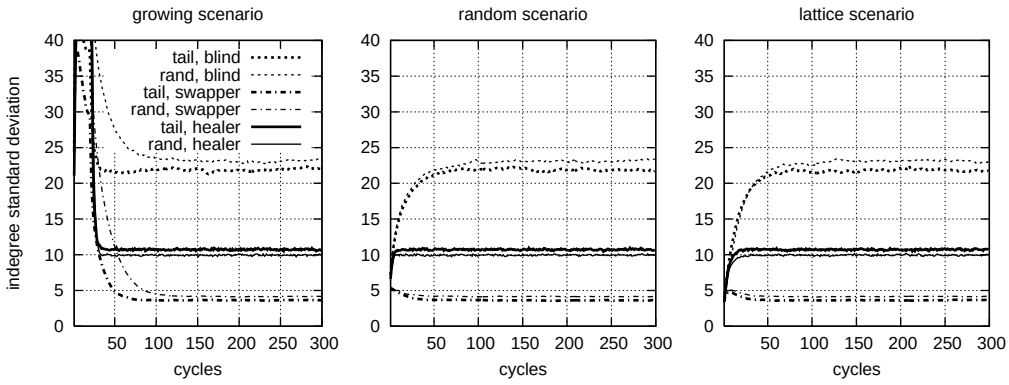


Figure 3.3: Evolution of standard deviation of indegree in all scenarios of pushpull protocols.

This also means that if the network topology moves towards the shape of a star, then the push protocols have extreme difficulty balancing this degree-distribution state again towards a random one.

We conclude that this lack of adaptivity and robustness effectively renders push-only protocols useless. In the remaining of this chapter we therefore consider only the pushpull model.

Figure 3.3 illustrates to convergence of the push-pull protocols. Note that the average indegree is always the view size ℓ. We can observe that in all scenarios the protocols quickly converge to the same value, even in the case of the growing scenario, in which the initial degree distribution is rather skewed. Other properties not directly related to degree distribution also show convergence, as discussed in Section 3.4.2.

Static Properties

In this section we examine the converged degree distributions generated by the different protocols. Figure 3.4 shows the converged standard deviation of the degree distribution. We observe that increasing both H and S results in a lower—and therefore more desirable—standard deviation. The reason is different for these two cases. With a large S, links to a node come to existence only in a very controlled way. Essentially, the only way new links to a node are created is by the node itself injecting its own fresh node descriptor during communication. On the other hand, with a large H, the situation is the opposite. When a node injects a new descriptor about itself, this descriptor is (exponentially often) copied to other nodes for a few cycles. However, one or two cycles later all copies are removed

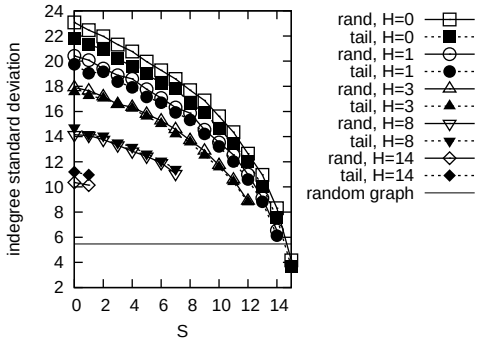


Figure 3.4: Converged values of indegree standard deviation.

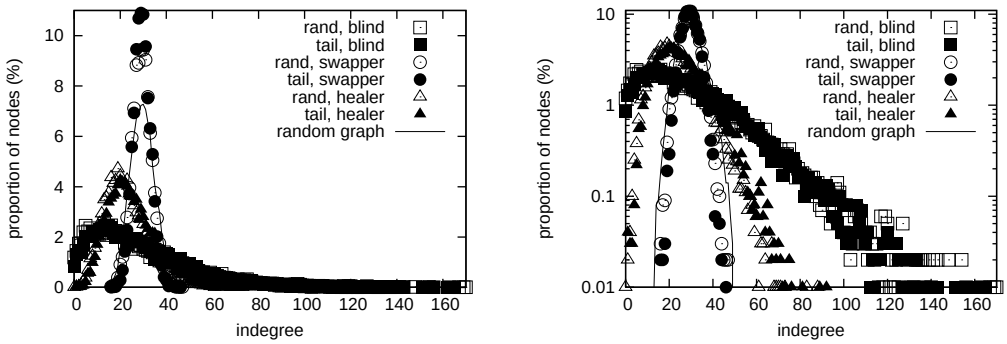


Figure 3.5: Converged indegree distributions on linear and logarithmic scales.

because they are pushed out by new links (i.e., descriptors) injected in the meantime. So the effect that reduces variance is the short lifetime of the copies of a given link.

Figure 3.5 shows the entire degree distribution for the three vertices of the design space triangle. We observe that the distribution of `swapper` is narrower than that of the random graph, while `blind` has a rather heavy tail and also a large number of nodes with zero or very few nodes pointing to them, which is not desirable from the point of view of load balancing.

Dynamic Properties

Although the distribution itself does not change over time during the continuous execution of the protocols, the behavior of a single node still needs to be

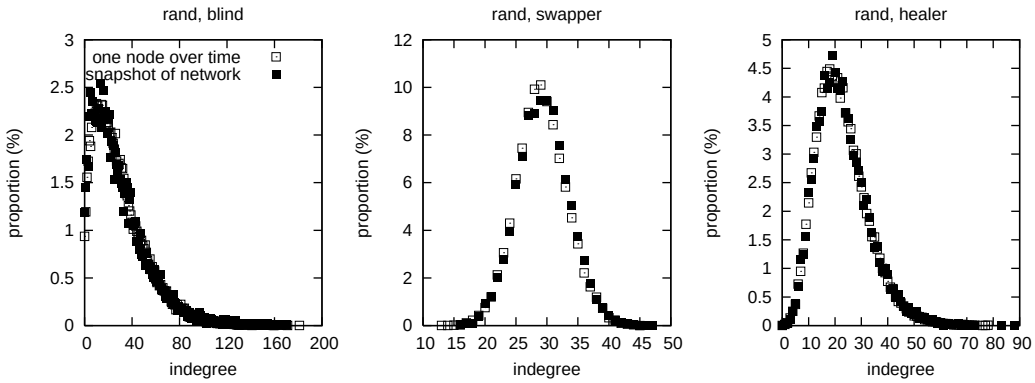


Figure 3.6: Comparison of the converged indegree distribution over the network at a fixed time point and the indegree distribution of a fixed node during an interval of 50,000 cycles. The vertical axis represents the proportion of nodes and cycles, respectively.

determined. More specifically, we are interested in whether a given fixed node has a variable indegree or whether the degree changes very slowly. The latter case would be undesirable because an unlucky node having above-average degree would continuously receive above-average traffic while others would receive less, which results in inefficient load balancing.

Figure 3.6 compares the degree distribution of a node over time, and the entire network at a fixed time point. The figure shows only the distribution for one node and only the random peer selection protocols, but the same result holds for tail peer selection and for all the 100 other nodes we have observed. From the fact that these two distributions are very similar, we can conclude that all nodes take all possible values at some point in time, which indicates that the degree of a node is not static.

However, it is still interesting to characterize *how quickly* the degree changes, and whether this change is predictable or random. To this end, we present auto-correlation data of the degree time-series of fixed nodes in Figure 3.7. The band indicates a 99% confidence interval assuming the data is random. Only one node is shown, but all the 100 nodes we traced show very similar behavior. Let the series $d_1, \ldots d_K$ denote the indegree of a fixed node in consecutive cycles, and $\overline{d}$ the average of this series. The autocorrelation of the series $d_1, \ldots d_K$ for a given time lag k is defined as

$$r_k = \frac{\sum_{j=1}^{K-k}(d_j - \overline{d})(d_{j+k} - \overline{d})}{\sum_{j=1}^{K}(d_j - \overline{d})^2},$$

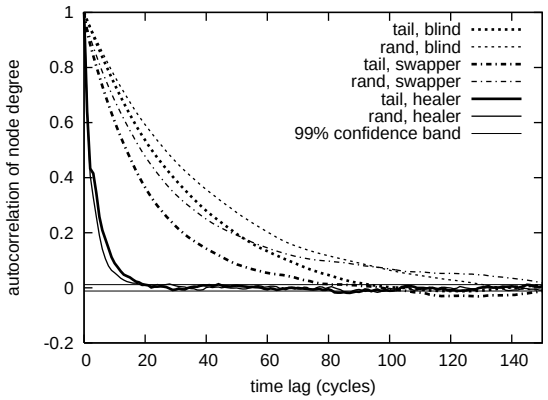


Figure 3.7: Autocorrelation of indegree of a fixed node over 50,000 cycles. Confidence band corresponds to the randomness assumption: a random series produces correlations within this band with 99% probability.

which expresses the correlation of pairs of degree values separated by k cycles.

We observe that in the case of `healer` it is impossible to make any prediction for a degree of a node 20 cycles later, knowing the current degree. However, for the rest of the protocols, the degree changes much slower, resulting in correlation in the distance of 80-100 cycles, which is not optimal from the point of view of load balancing.

3.4.2. Clustering and Path Lengths

Degree distribution is an important property of random graphs. However, there are other equally important characteristics of networks that are independent of degree distribution. In this section we consider the *average path length* and the *clustering coefficient* as two such characteristics. The clustering coefficient is defined over undirected graphs (see below). Therefore, we consider the undirected version of the overlay after removing the orientation of the edges.

Average path length The shortest path length between node a and b is the minimal number of edges required to traverse in the graph in order to reach b from a. The average path length is the average of shortest path lengths over all pairs of nodes in the graph. The motivation of looking at this property is that, in any information dissemination scenario, the shortest path length defines a lower bound on the time and costs of reaching a peer. For the sake of scalability a small average path length is essential. In Figure 3.8, especially in the growing and lattice

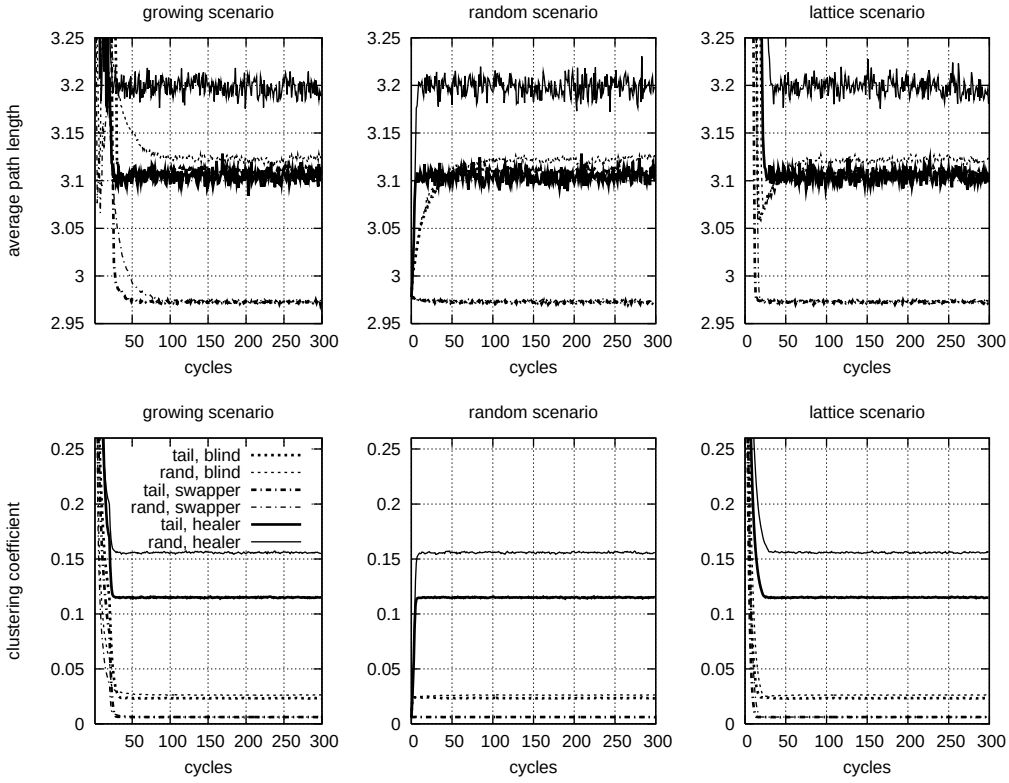


Figure 3.8: Evolution of the average path length and the clustering coefficient in all scenarios.

scenarios, we verify that the path length converges rapidly. Figure 3.9 shows the converged values of path length for the design space triangle defined by H and S. We observe that all protocols result in a very low path length. Large S values are the closest to the random graph.

Clustering coefficient The clustering coefficient of a node a is defined as the number of edges between the neighbors of a divided by the number of all possible edges between those neighbors. Intuitively, this coefficient indicates the extent to which the neighbors of a are also neighbors of each other. The clustering coefficient of a graph is the average of the clustering coefficients of its nodes, and always lies between 0 and 1. For a complete graph, it is 1, for a tree it is 0. The motivation for analyzing this property is that a high clustering coefficient has potentially damaging effects on both information dissemination (by increasing the number of redundant messages) and also on the self-healing capacity by weakening the

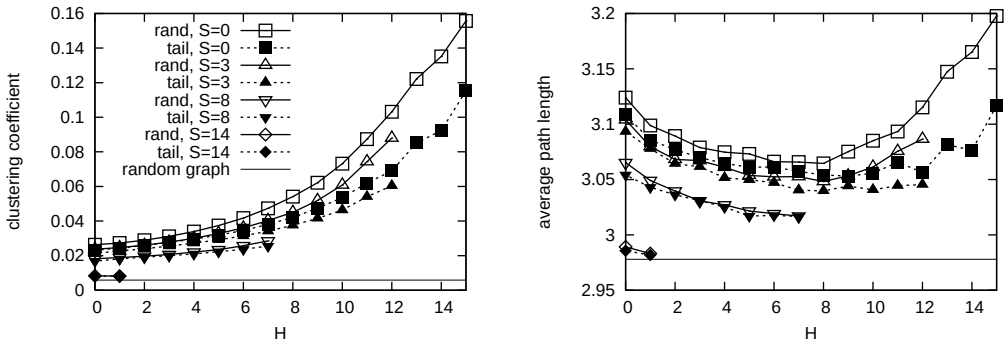


Figure 3.9: Converged values of clustering coefficient and average path length.

connection of a cluster to the rest of the graph thereby increasing the probability of partitioning. Furthermore, it provides an interesting possibility to draw parallels with research on complex networks where clustering is an important research topic (i.e., in social networks) [Watts and Strogatz 1998].

Like average path length, the clustering coefficient also converges (see Figure 3.8); Figure 3.9 shows the converged values. It is clear that clustering is controlled mainly by H. The largest values of H result in rather significant clustering, where the deviation from the random graph is large. The reason is that if H is large, then a large part of the views of any two communicating nodes will overlap right after communication, since both keep the same freshest entries. For the largest values of S, clustering is close to random. This is not surprising either because S controls exactly the diversity of views.

3.5. FAULT TOLERANCE

In large-scale, dynamic, wide-area distributed systems it is essential that a protocol is capable of maintaining an acceptable quality of service under a wide range of severe failure scenarios. In this section we present simulation results on two classes of such scenarios: *catastrophic failure*, where a significant portion of the system fails at the same time (e.g., due to network partitioning), and heavy *churn*, where nodes join and leave the system continuously.

3.5.1. Catastrophic Failure

Failures on network backbones typically split a network in two or more disjoint partitions of different geographic locations (e.g., nodes in Europe and nodes in

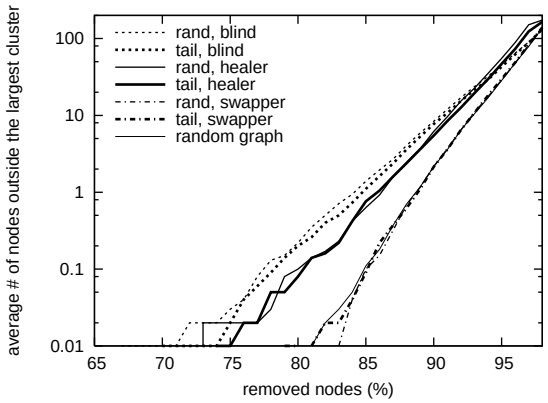


Figure 3.10: The number of nodes that do not belong to the largest connected cluster. The average of 100 experiments is shown. The random graph almost completely overlaps with the swapper protocols.

America), or administrative jurisdictions (e.g., nodes in ISP X and nodes in ISP Y). Note that, from the point of view of each partition, partitioning appears as if a large number of nodes (i.e., the ones belonging to a disjoint partition) died altogether.

In this section we test the PEER SAMPLING SERVICE against partitioning failures. In fact, we focus on a single partition, and examine how it emerges after partitioning has occurred. This is modeled by a catastrophic failure. More specifically, since our protocols are completely symmetric with respect to node locations, partitioning is modeled as the removal of a *random* subset of the nodes at once.

As in the case of the degree distribution, the response of the protocols to a massive failure has a static and a dynamic aspect. In the static setting we are interested in the self-healing capacity of the converged overlays to a (potentially massive) node failure, as a function of the number of failing nodes. Removing a large number of nodes will inevitably cause some serious structural changes in the overlay even if it otherwise remains connected. In the dynamic case we would like to learn to what extent the protocols can repair the overlay after a severe damage.

The effect of a massive node failure on connectivity is shown in Figure 3.10. In this setting the overlay in cycle 300 of the random initialization scenario was used as converged topology. From this topology, random nodes were removed and the connectivity of the remaining nodes was analyzed. In all of the $100 \times 6 = 600$ experiments performed we did not observe partitioning until removing 67% of the nodes. The figure depicts the number of the nodes outside the largest connected cluster. We observe consistent partitioning behavior over all protocol

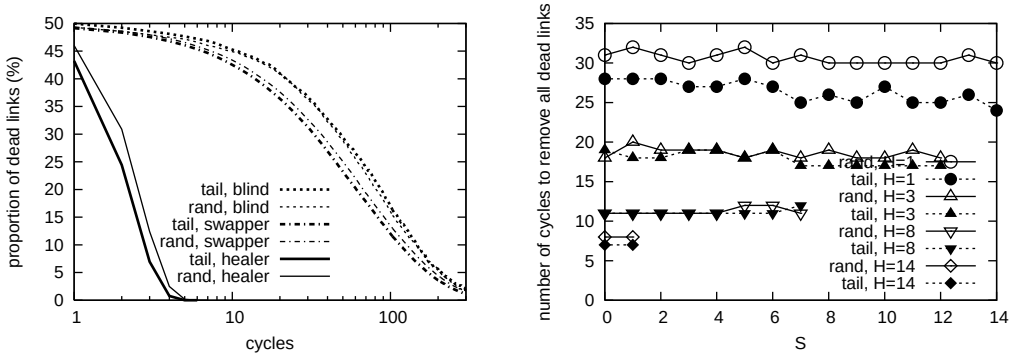


Figure 3.11: Removing dead links following the failure of 50% of the nodes in cycle 300.

instances (with `swapper` being particularly close to the random graph): even when partitioning occurs, most of the nodes form a single large connected cluster. Note that this phenomenon is well known for traditional random graphs [Newman 2002].

In the dynamic scenario we made 50% of the nodes fail in cycle 300 of the random initialization scenario and we then continued running the protocols on the damaged overlay. The damage is expressed by the fact that, on average, half of the view of each node consists of descriptors that belong to nodes that are no longer in the network. We call these descriptors dead links. Figure 3.11 shows how fast the protocols repair the overlay, that is, remove dead links from the views. Based on the static node failure experiment it was expected that the remaining 50% of the overlay is not partitioned and indeed, we did not observe partitioning with any of the protocols. Self-healing performance is fully controlled by the healing parameter H, with $H = 15$ resulting in fully repairing the network in as little as 5 cycles (not shown).

3.5.2. Churn

To examine the effect of churn, we define an artificial scenario in which a given proportion of the nodes crash and are subsequently replaced by new nodes in each cycle. This scenario is a *worst case* scenario because the new nodes are assumed to join the system for the first time, therefore they have no information whatsoever about the system (their view is initially empty) and the crashed nodes are assumed never to join the system again, so the links pointing to them will never become valid again. A more realistic trace-based scenario is also examined in Section 3.5.3 using the Gnutella trace described in [Saroiu et al. 2003].

We focus on two aspects: the churn rate, and the bootstrapping method. Churn rate defines the number of nodes that are replaced by new nodes in each cycle. We consider *realistic* churn rates (0.1% and 1%) and a *catastrophic* churn rate (30%). Since churn is defined in terms of cycles, in order to validate how realistic these settings are, we need to define the cycle length. With the very conservative setting of 10 seconds, which results in a very low load at each node, the trace described in [Saroiu et al. 2003] corresponds to 0.2% churn in each cycle. In this light, we consider 1% a comfortable upper bound of realistic churn, given also that the cycle length can easily be decreased as well to deal with even higher levels of churn.

We examine two bootstrapping methods. Both are rather unrealistic, but our goal here is not to suggest an optimal bootstrapping implementation, but to analyze our protocols under churn. The following two methods are suitable for this purpose because they represent two opposite ends of the design space:

Central We assume that there exists a server that is known by every joining node, and that is stable: it is never removed due to churn or other failures. This server participates in the gossip membership protocol as an ordinary node. The new nodes use the server as their first contact. In other words, their view is initialized to contain the server.

Random An oracle gives each new node a random live peer from the network as its first contact.

Realistic implementations could use a combination of these two approaches, where one or more servers serve random contact peers, using the PEER SAMPLING SERVICE itself. Any such implementation can reasonably be expected to result in a behavior in between the two extremes described above.

Simulation experiments were run initializing the network with random links and subsequently running the protocols under the given amount of churn until the observed properties reach a stable level (300 cycles). The experimental results reveal that for realistic churn rates (0.1% and 1%) all the protocols are robust to the bootstrapping method and the properties of the overlay are very close to those without churn. Figure 3.12 illustrates this by showing the standard deviation of the node degrees in both scenarios, for the higher churn rate 1%. Observe the close correspondence with Figure 3.4. The clustering coefficient and average path length show the same robustness to bootstrapping, and the observed values are almost identical to the case without churn (not shown).

Let us now consider the damage churn causes in the networks. First of all, for all protocols and scenarios the networks remain connected, even for $H = 0$. Still, a (low) number of dead links remain in the overlay. Figure 3.13 shows the average number of dead links in the views, again, only for the higher churn rate (1%). It is clear that the extent of the damage is fully controlled by the healing parameter H.

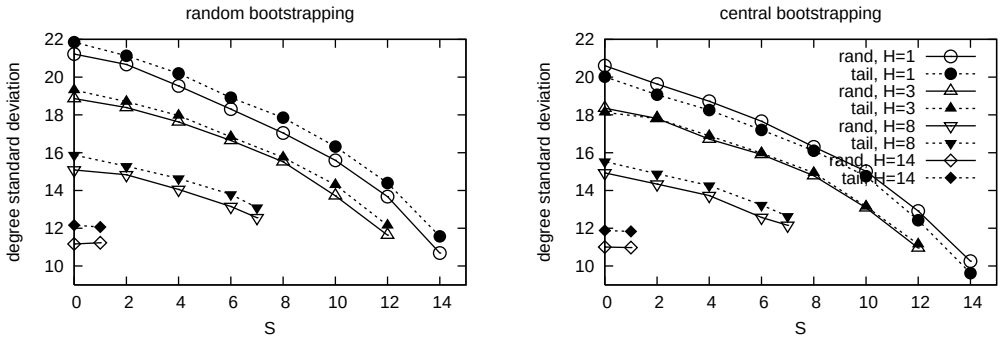


Figure 3.12: Standard deviation of node degree with churn rate 1%. Node degree is defined over the *undirected* version of the subgraph of *live* nodes. The $H = 0$ case is not comparable to the shown cases; due to reduced self-healing, nodes have much fewer live neighbors (see Figure 3.13) which causes relatively low variance.

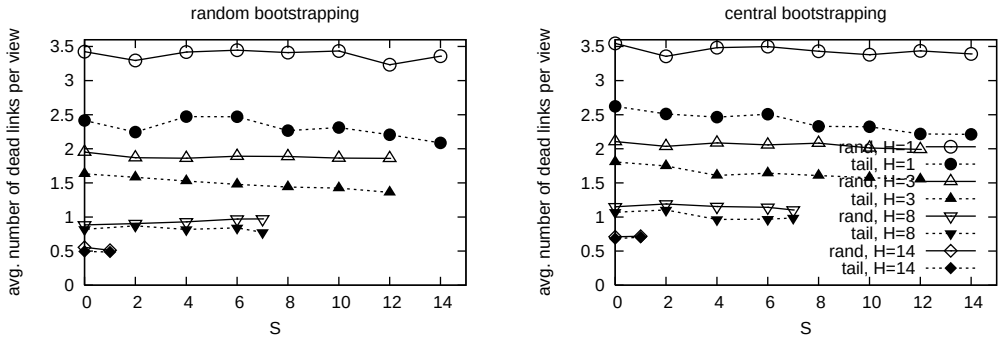


Figure 3.13: Average number of dead links in a view with churn rate 1%. The $H = 0$ case is not shown; it results in more than 11 dead links per view on average, for all settings.

Furthermore, it is clear that the protocols are robust to the bootstrapping scenario also in this case. If $H \geq 1$ then the *maximal* (not average) number of dead links in any view for the different protocol instances ranges from $5-13$ in the case of churn rate 1% and from $2-5$ for churn rate 0.1%, where the lowest value belongs to the highest H. If $H = 0$ then the number of dead links radically increases: it is at least 11 on average, and the maximal number of dead links ranges from 20-25 for the different settings. That is, in the presence of churn, it is essential for any implementation to set at least $H = 1$. We have already seen this effect in Section 3.5.1 concerning self-healing performance.

Although the server participates in the overlay, the plots showing results under the central bootstrapping scenario were calculated ignoring the server, because its properties sharply differ from the rest of the network. In particular, it has a high indegree, because all new nodes will have a fresh link to the server, and that link will stay in the view of joining nodes for a few more cycles, possibly replicated in the meantime. Indeed, we observe that for 1% churn, $12\% - 28\%$ of the nodes have a link to the server at any time, depending on H and S. However, if we assume that the server can handle the traffic generated by joining nodes, a high indegree is non-critical. The expected number of incoming messages due to indegree d is d/ℓ (where ℓ is the view length), with a very low variance. This means that the generated traffic is of the same order of magnitude as the traffic generated by the joining nodes. We note again however, that we do not consider this simplistic server-based solution a practical approach; we treat it only as a worst-case scenario to help us evaluate the protocols.

So far we have been discussing realistic churn rates. However, it is of academic interest to examine the behavior under *extremely* difficult scenarios, where the network suffers a catastrophic damage *in each cycle*. The catastrophic churn rate of 30% combines the effects of catastrophic failure (see Section 3.5.1) and churn.

Unlike with realistic churn rates, in this case the bootstrapping method has a strong effect on the performance of the protocols and therefore becomes the major design decision, although the parameters H and S still have a very strong effect as well. Consequently, we need to analyze the interaction of the gossip membership protocol and the bootstrapping method. In the case of the server-based solution, the overlay evolves into a ring-like structure, with a few shortcut links. The reason is that the view of the server is predominantly filled with entries of the newly joined nodes, since each time a new node contacts the server it also places a fresh entry about itself in the view of the server. These entries are served to the subsequently joining nodes, thus forming a linear structure. This ring-like structure is rather robust: it remains connected (even after removing the server) for all protocols with $H >= 8$. However, it has a slightly higher diameter than

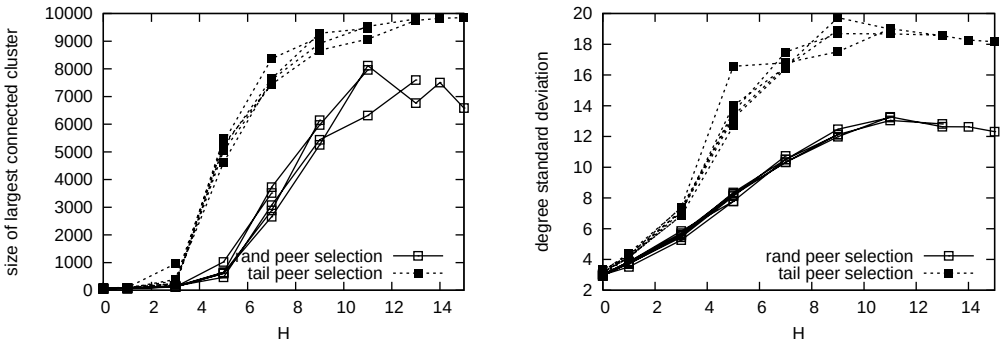


Figure 3.14: Size of largest connected cluster and degree standard deviation under catastrophic churn rate (30%), with the random bootstrapping method. Individual curves belong to different values of S but the measures depend only on H, so we do not need to differentiate between them. Connectivity and node degree are defined over the *undirected* version of the subgraph of *live* nodes.

that of the random graph (approximately 20-30 hops). For `healer` the average number of dead links per view is still as low as 10 and 9 for random and tail peer selection, respectively.

The random scenario is rather different. In particular, we lose connectivity for all the protocols, however, for large values of H the largest connected cluster almost reaches the size of the network (see Figure 3.14). Besides, the structure of the overlay is also different. As Figure 3.14 shows, tail peer selection results in a slightly more unbalanced degree distribution (note that the low deviation for low values of H is due to the low number of live nodes). The reason is that—also considering that tail peer selection picks the oldest *live* node—the nodes that stay in the overlay for somewhat longer will receive more incoming traffic because (due to the very high number of dead links in each view) they tend to be the oldest live node in most views they are in. For `healer` the average number of dead links per view is 11 and 9 for random and tail peer selection, respectively.

To summarize our findings: under realistic churn rates all the protocols perform very similarly to the case when there is no churn at all, independently of the bootstrapping method. Besides, some of the protocol instances, in particular, `healer`, can tolerate even catastrophic churn rates with a reasonable performance with both bootstrapping methods.

3.5.3. Trace-driven Churn Simulations

In Section 3.5.2 we analyzed our protocols under artificial churn scenarios. Here, we consider a realistic churn scenario using the, so called, *lifetime measurements* on Gnutella, carried out by Saroiu et al.[Saroiu et al. 2003]. These traces contain—among other information—the connection and disconnection times for a total of 17,125 nodes over a period of 60 hours. Throughout the trace, the number of connected nodes remains practically unchanged, in the order of 10^4 nodes.

We noticed a periodic pattern occurring every 404 seconds in the traces. In each 404-second interval, all connections and disconnections take place during the first 344 seconds, rendering the network static during the last 60 seconds. These recurring gaps would represent a positive bias for our churn simulations, as they periodically provide the overlay with some "breathing space" to process recent changes. However, these gaps are not realistic and are most probably an artifact of the logging mechanism. Therefore, we decided to eliminate them by linearly expanding each 344 second interval to cover the whole 404 seconds. Note that this transformation leaves the node uptimes practically unaltered.

We have taken the following two decisions with respect to the parameters in the experiments presented. First, peer selection is fixed to random. Section 3.5.2 showed that random is outperformed by tail peer selection in all cases. Therefore, random is a suitable choice for this section as the worst case peer selection policy. Second, the swapping parameter, S, is fixed to 0. Section 3.5.2 showed that $S = 0$ results in the highest (therefore worst) degree deviation, while it does not affect the number of dead links.

We apply two join methods: central and random, as defined in Section 3.5.2. The only difference is that a reconnecting node still remembers the links it previously had, some of which may be dead at reconnection time. This facilitates reconnection, but generally increases the total number of dead links.

The cycle length was chosen to be 1 minute. We anticipate that in reality the cycle length will be shorter, resulting in lower churn per cycle. The choice of a cycle length close to the upper end of realistic values is intentional, and is aimed at testing this specific gossip membership protocol under increased stress.

Figure 3.15 shows the node connections and disconnections as a percentage of the current network size. Connections are shown as positive points, whereas disconnections as negative. Although we ran the experiments for the whole trace, we focus on its most interesting part, namely cycles 2250 to 2750. Notice that at cycle 2367, around 450 nodes get disconnected at once and reconnect altogether 27 minutes later, at cycle 2394, probably due to a router failure. Similar temporary—but shorter—group disconnections are observed later on, around cycles 2450, 2550, and 2650, respectively.

Let us now examine the way the overlay is affected by those network changes.

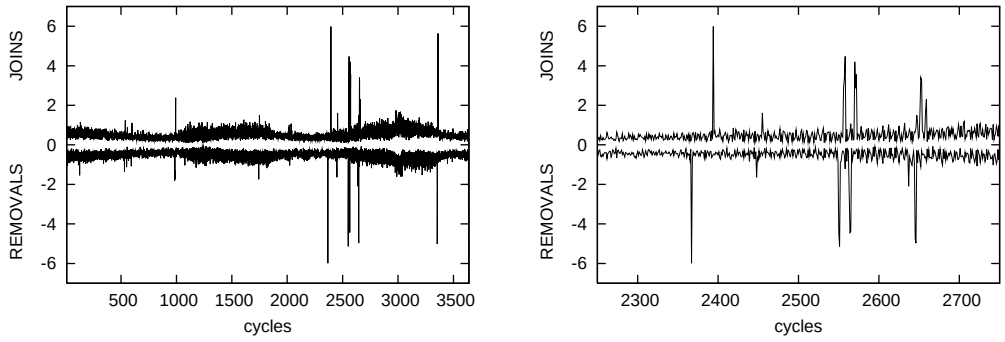


Figure 3.15: Churn in the Saroiu traces. Full time span of 3600 one minute cycles and zoomed in to cycles 2250 to 2750.

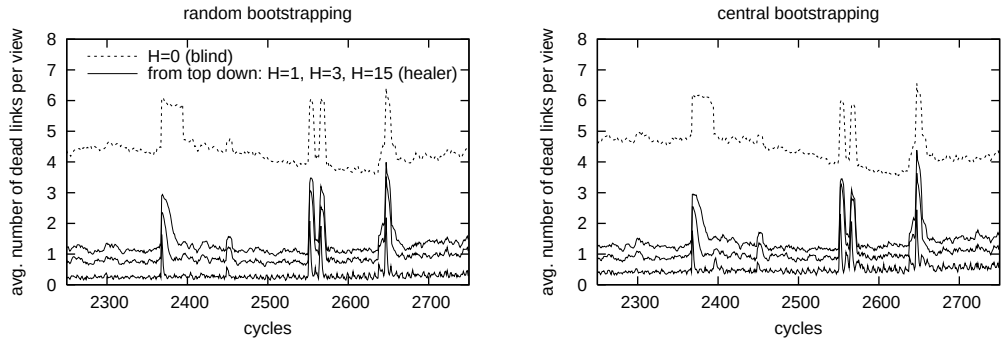


Figure 3.16: Average number of dead links per view, based on the Saroiu Gnutella traces. All experiments use random peer selection and $S = 0$.

Figure 3.16 shows that the number of dead links is always kept at fairly small levels, especially when H is at least 1. As expected, the number of dead links peaks when there are massive node disconnections and get back to normal quickly. However, it is not affected by the observed massive node reconnections, because these happen shortly after the respective disconnections, and the neighbors of the reconnected nodes are still alive.

Two observations regarding the effect of H can be made. First, higher values of H result in fewer dead links per view, validating the analysis in Section 3.5.2. Second, higher values of H trigger the *faster* elimination of dead links. The peaks caused by massive node disconnections are wider for low H values, becoming sharper as H grows higher. In fact, these two observations are related to each

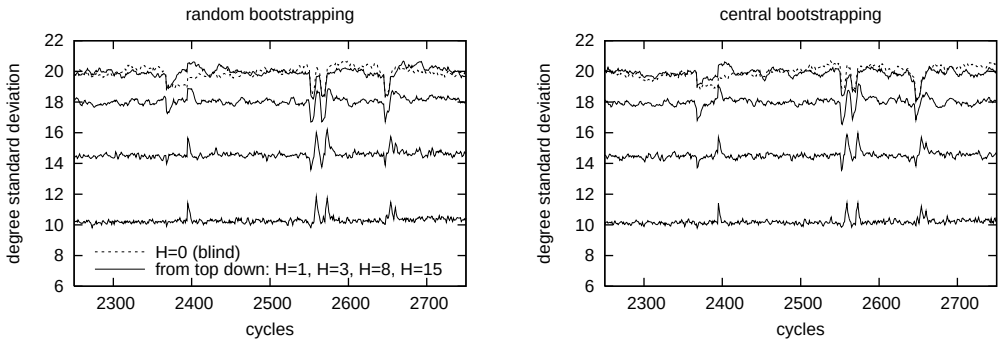


Figure 3.17: Evolution of standard deviation of node degree based on the Saroiu Gnutella traces. All experiments use random peer selection and $S = 0$.

other: in a persistently dynamic network, the converged average number of dead links depends on the rate at which the protocol disposes of them.

Figure 3.17 shows the evolution of the node degree deviation. It can be observed that for $H \geq 1$ the degree deviation under churn is very close to the corresponding converged values in a static network (see Figure 3.4). For $H = 0$ though, the higher number of pending dead links affects the degree distribution more. Note that both massive node disconnections *and* connections disturb the degree deviation, but in both cases a few cycles are sufficient to recover the original overlay properties.

To recap our analysis, we have shown that even with a pessimistic cycle length of 1 minute, all protocols for $H \geq 1$ perform very similarly to the case of a stable network, independently of the join method. Anomalies caused by massive node connections or disconnections are repaired quickly.

3.6. WIDE-AREA-NETWORK EMULATION

Distributed protocols often exhibit unexpected behavior when deployed in the real world, that cannot always be captured by simulation. Typically, this is due to unexpected message loss, network and scheduling delay, as well as events taking place in unpredictable, arbitrary order. In order to validate the correctness of our simulation results, we implemented our gossip membership protocols and deployed them on a wide-area network.

We utilized the DAS-2 wide-area cluster as our testbed [DAS-2]. The DAS-2 cluster consists of 200 dual-processor nodes spread across 5 sites in the Nether-

lands. A total of 50 nodes were used for our emulations, 10 from each site. Each node was running a Java Virtual Machine emulating 200 peers, giving a total of 10,000 peers. Peers were running in separate threads.

Although 200 peers were running on each physical machine, communication within a machine accounted for only 2% of the total communication. Local-area and wide-area traffic accounted for 18% and 80% of the total, respectively. Clearly, most messages are transferred through wide area connections. Note that the intra-cluster and inter-cluster round-trip delays on the DAS-2 are in the orders of 0.15 and 2.5 milliseconds, respectively. In all emulations, the cycle length was set to 5 seconds.

In order to validate our simulation results, we repeated the experiments presented in Figures 3.3 and 3.8 of Section 3.4, using our real implementation. A centralized coordinator was used to initialize the node views according to the bootstrapping scenarios presented in Section 3.4.1, namely *growing*, *lattice*, and *random*.

The first run of the emulations produced graphs practically indistinguishable from the corresponding simulation graphs. Acknowledging the low round-trip delay on the DAS-2, we ran the experiments again, this time inducing a 50 msec delay in each message delivery, accounting for a round-trip delay of 100 msec on top of the actual one. The results presented in this section are all based on these experiments.

Figure 3.18 shows the evolution of the in-degree standard deviation, clustering coefficient, and average path length for all experiments, using the same scales as Figures 3.3 and 3.8 to facilitate comparison. The very close match between simulation-based and real-world experiments for all three nodes of the design space triangle allows us to claim that our simulations represent a valid approximation of real-world behavior.

The small differences of the converged values with respect to the simulations are due to the induced round-trip delay. In a realistic environment, view exchanges are not atomic: they can be intercepted by other view exchanges. For instance, a node having initiated a view exchange and waiting for the corresponding reply, may in the meantime receive a view exchange request by a third node. However, the view updates performed by the active and passive thread of a node are not commutative. The presented results correspond to an implementation where we simply ignored this problem: all requests are served immediately regardless of the state of the serving node. This solution is extremely simple from a design point of view but may lead to corrupted views.

As an alternative, we devised and implemented three approaches to avoid corrupted views. In the first approach, a node's passive thread *drops incoming requests* while its active thread is waiting for a reply. In the second one, the node

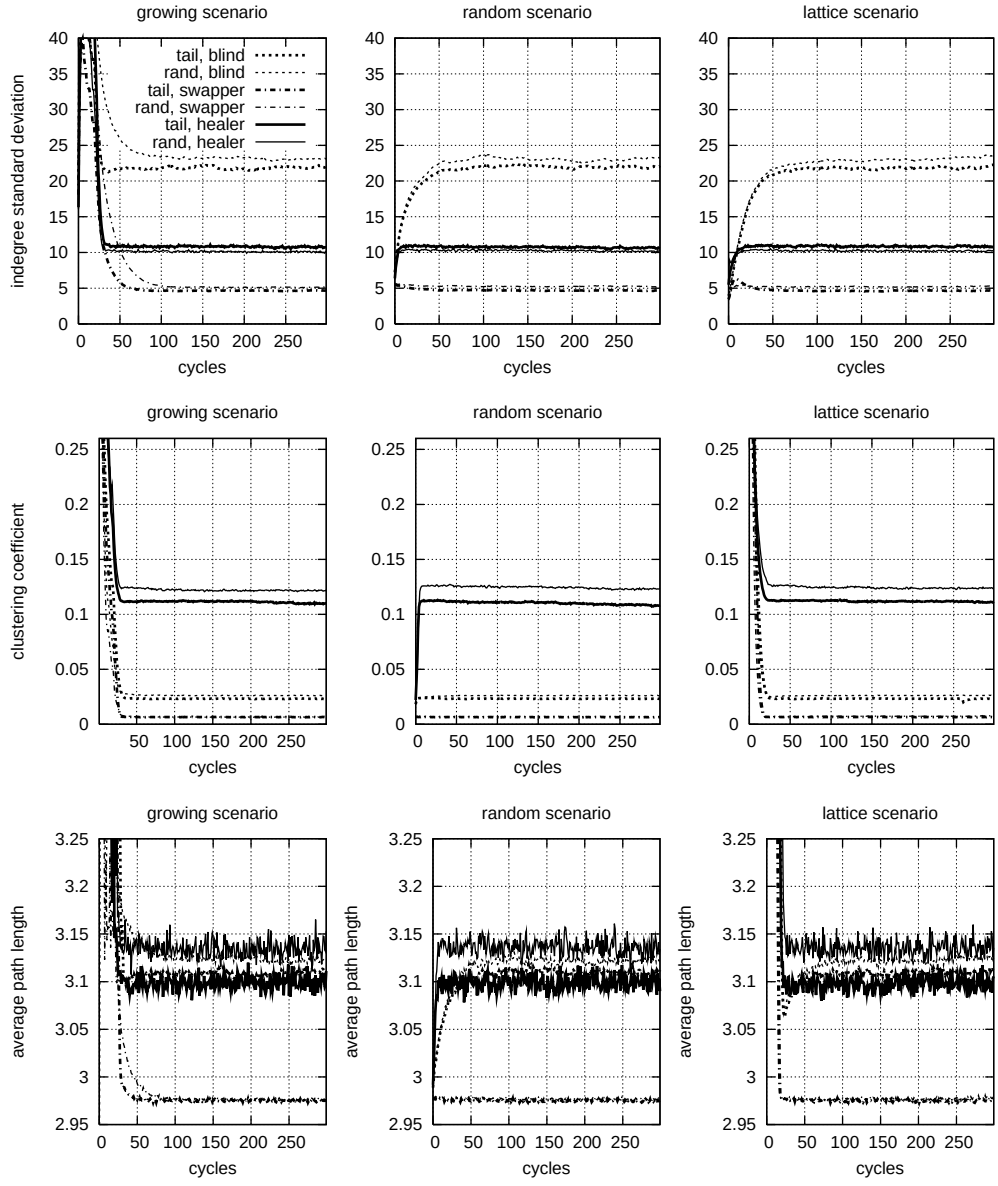


Figure 3.18: Evolution of in-degree standard deviation, clustering coefficient, and average path length in all scenarios for *real-world* experiments.

queues—instead of dropping—incoming requests until the awaited reply comes. As a third approach, a node's passive thread serves all incoming requests, but its active thread *drops a reply* if an incoming request intervened.

Apart from the added complexity that these solutions impose on our design, their benefit turned out to be difficult or impossible to notice. Moreover, undesirable situations may arise in the case of the first two: dropping or delaying a request from a third node may cause that node to drop or delay, in turn, requests it receives itself. Chains of dependencies are formed this way, which can render parts of the network inactive for some periods. Given the questionable advantage these approaches can offer, and considering the design overhead they impose, we will not consider them further. Based on our experiments, the best strategy is simply ignoring the problem, which further underlines the exceptional robustness of gossip-based design.

3.7. DISCUSSION

In this section we summarize and interpret the results presented so far. As stated in the introductory section, we were interested in determining the properties of various gossip membership protocols, in particular their randomness, load-balancing and fault-tolerance. In a sense, after we discussed in the last section why certain results were observed, we discuss here what the results imply.

3.7.1. Randomness

We have studied randomness from two points of view: local and global. Local randomness is based on the analogy between a pseudo random-number generator and the PEER SAMPLING SERVICE as seen by a fixed node. We have seen that all protocols return a random sequence of peers at all nodes with a good approximation.

We have shown, however, that there are important correlations between the samples returned at the different nodes, that is, the overlay graphs that the implementations are based upon are not random. Adopting a graph-theoretic approach, we have been able to identify important deviations from randomness that are different for the several instances of our framework.

In short, randomness is approached best by the view selection method `swapper` ($H = 0, S = 15$), irrespectively of the peer selection method. In general, increasing H increases the clustering coefficient. The average path length is close to the one of a random graph for all protocols we examined. Finally, with `swapper` the degree distribution has a smaller variance than that of the random graph.

This property can often be considered "better than random" (i.e., from the point of view of load balancing).

Clearly, the randomness required by a given application depends on the very nature of that application. For example, the upper bound of the speed of reaching all nodes via flooding a network depends exclusively on the diameter of the network, while other aspects such as degree distribution or clustering coefficient are irrelevant for this specific question. Likewise, if the sampling service is used by a node to draw samples to calculate a local statistical estimate of some global property such as network size or the availability of some resources, what is needed is that the local samples are uniformly distributed. However, it is *not* required that the samples are independent at different nodes, that is, we do not need global randomness at all; the unstructured overlay can have any degree distribution, diameter, clustering, etc.

Load Balancing

We consider the service to provide good load balancing if the nodes evenly share the cost of maintaining the service and the cost induced by the application of the service. Both are related to the degree distribution: if many nodes point to a certain node, this node will receive more sampling-service related gossip messages and most applications will induce more overhead on this node, resulting in poor load balancing. Since the unstructured overlays that implement the sampling service are dynamic, it is also important to note that nodes with a high indegree become a bottleneck only if they keep having a high indegree for a long time. In other words, a node is in fact allowed to have a high indegree temporarily, for a short time period.

We have seen that the `blind` view selection is inferior to the other alternatives. The degree distribution has a high variance (that is, there are nodes that have a large indegree) and on top of that, the degree distribution is relatively static, compared to the alternatives.

Clearly, the best choice to achieve good load balancing is the `swapper` view selection, which results in an even lower variance of indegree than in the uniform random graph. In general, the parameter S is strongly correlated with the variance of indegree: increasing S for a fixed H decreases the variance. The degree distribution is almost as static as in the case of `healer`, if $H = 0$. However, this is not a problem because the distribution has low variance.

Finally, `healer` also performs reasonably. Although the variance is somewhat higher than that of `swapper`, it is still much lower than `blind`. Besides, the degree distribution is highly dynamic, which means that the somewhat higher variance of the degree distribution does not result in bottlenecks because the indegree of the nodes change quickly. In general, increasing H for a fixed value of S

also decreases the variance.

Fault Tolerance

We have studied both catastrophic and realistic scenarios. In the first category, catastrophic failure and catastrophic churn was analyzed. In these scenarios, the most important parameter turned out to be H: it is always best to set H as high as possible. One exception is the experiment with the removal of 50% of the nodes, where `swapper` performs slightly better. However, `swapper` is slow in removing dead links, so if failure can be expected, it is highly advisable to set $H \geq 1$.

In the case of realistic scenarios, such as the realistic (artificial) churn rates, and the trace-based simulations, we have seen that the damaging effect is minimal, and (as long as $H \geq 1$) the performance of the protocols is very similar to the case when there is no failure.

3.8. RELATED WORK

3.8.1. Membership Management Protocols

Most gossip protocols for implementing peer sampling are covered by our framework: we mentioned these in Section 3.2.2. One notable exception is [Allavena et al. 2005] that we address here in a bit more detail. The protocol is as follows. In each cycle, all nodes pull the full partial views from F randomly selected peers. In addition, they record the addresses of the peers initiating incoming pull requests during the given cycle. The old view is then discarded and a new view is generated from scratch. In the most practical version, the new view is generated by first adding the addresses of the incoming requests and subsequently filling the rest of the view with random samples from the union of the previously pulled F views without replacement.

Note the two features that are incompatible with our framework: the application of $F \geq 1$ (in our case $F = 1$) and the asymmetry between push and pull, with pull having a larger emphasis: only one entry—the initiator peer's own entry—is pushed. Note that it is common to allow for $F \geq 1$ also in other proposals (e.g., [Eugster et al. 2003b]). In our framework, information exchange is symmetric, or fully asymmetric, without a finer tuning possibility.

To compare this protocol with our framework, we implemented it and ran simulations using the scenarios presented in this chapter. The view size and network size were the same as in all simulations, and F was 1, 2 or 3. The main conclusions are summarized below. The protocol class presented in [Allavena et al.

2005] has some difficulty dealing with the scenarios when the initial network is not random (the growing and lattice initializations, see Section 3.4.1). For $F = 1$ we consistently observed partitioning in the lattice scenario (which was otherwise never observed in our framework). For the growing scenario, the protocols occasionally get stuck in a local attractor where there is a star subgraph: a node with a very high indegree, and a large number of nodes with zero indegree and 1 as outdegree. Apart from these issues, if we consider self-healing, load balancing and convergence properties, the protocols roughly behave as if they were instances in our framework using pushpull, with $0 \leq H \leq 1$ and $S = 0$, with increasing F tending towards $H = 1$. Sinc1 e we have concluded that the "interesting" protocols in our space have either a high H or a high S value, based on the empirical evidence accumulated so far there is no urgent need to extend our framework to allow for $F > 1$ or asymmetric information exchange. However, studying these design choices in more detail is an interesting topic for future research.

With respect to membership management, not all proposed systems are gossip-based. It is worth noting Moshe [Keidar et al. 2002], a distributed membership management system that is based on a set of devoted servers. Each client registers the multicast group(s) it is interested in with one of several dedicated servers, preferably one in the same local-area network. Each server maintains complete knowledge of the membership information for the whole network. Membership changes are reported to servers through a third-party notification service monitoring the system. Upon a membership change, servers talk to each other to reach a consensus regarding the current state of membership. Once the network stabilizes and a consensus is reached, servers propagate to each client the complete membership information of the groups it is subscribed for.

Moshe is designed for systems where the full view of multicast groups is required by all group members. It focuses on lowering the bandwidth used for reaching an agreement among the servers, and for updating clients' views. In that respect, it is based on the assumption of a small, generally stable network, where membership changes occur occasionally. It has been successfully tested on 50 nodes distributed across different continents. However, it is not clear whether its design would be suitable for internet-scale applications, for two reasons. First, in internet-scale applications continuous membership change is the norm and the network never comes to a stable state. Second, full view of group membership by all nodes is practically infeasible for groups expanding to thousands or millions of nodes.

In the following we summarize a number of other fields that are related to the research presented in this chapter.

3.8.2. Complex Networks

The assumption of uniform randomness has only fairly recently become subject to discussion when considering large complex networks such as the hyperlinked structure of the WWW, or the complex topology of the Internet. Like social and biological networks, the structures of the WWW and the Internet both follow the quite unbalanced power-law degree distribution, which deviates strongly from that of traditional random graphs. These new insights pose several interesting theoretical and practical problems [Barabási 2002]. Several dynamic complex networks have also been studied and models have been suggested for explaining phenomena related to what we have described in this chapter [Dorogovtsev and Mendes 2002]. This related work suggests an interesting line of future theoretical research seeking to explain our experimental results in a rigorous manner.

An interesting direction of graph theory research concerns *expander graphs*. A graph consisting of N vertices is a $\beta-$*expander* if, for any subset of vertices S, such that $|S| < N/2$, the number of links from nodes in S to the rest of the nodes is at least $\beta \times |S|$. Essentially, an expander graph defines a lower limit in the number of outgoing links from *any* subset of the graph nodes, proportional to the subset size. Expander graphs have a wide spectrum of applications, including error-correction codes and probabilistically checkable proofs (PCPs). What makes them attractive for computer networks is the fact that they are rich in short, disjoint paths. This allows the establishment of virtual circuits along edge-disjoint paths, avoiding link congestion.

The expander graphs of practical interest for computer networks are the ones with bounded node degrees. These graphs have been shown to possess desirable properties regarding the existence of disjoint paths. A number of papers have studied the relation between the expansion properties of a bound-degree expander graph and the number of disjoint paths it can concurrently support [Peleg and Upfal 1987, 1989; Broder et al. 1994; Kleinberg and Tardos 1995; Kleinberg and Rubinfeld 1996; Broder et al. 1997, 1999]. These papers have also suggested polynomial-time algorithms based on random-walks for the discovery of such paths, which are out of the scope of this dissertation. This related work is highly relevant to our work, as it has been shown that regular random graphs are excellent expanders [Frieze and Zhao 1999]. The PEER SAMPLING SERVICE could, therefore, form the basis for building such expander graphs.

3.8.3. Unstructured Overlays

There are a number of protocols that are not gossip-based but that are potentially useful for implementing peer sampling. An example is the Scamp protocol [Ganesh et al. 2003]. While this protocol is reactive and so less dynamic, an

explicit attempt is made towards the construction of a (static) random graph topology. Randomness has been evaluated in the context of information dissemination, and it appears that reliability properties come close to what one would see in random graphs. Finally, some other protocols have also been proposed to achieve randomness [Law and Sui 2003; Pandurangan et al. 2003], although not having the specific requirements of the PEER SAMPLING SERVICE in mind.

3.8.4. Structured Overlays

A structured overlay [Rowstron and Druschel 2001a; Ratnasamy et al. 2001a; Stoica et al. 2001] is by definition not dynamic so to utilize it for implementing the PEER SAMPLING SERVICE random walks or other additional techniques have to be applied [Zhong et al. 2005; King and Saia 2004]. Another example of this approach is a method assuming a tree overlay [Kostić et al. 2003]. It is unclear whether a competitive implementation can be given considering also the cost of maintaining the respective overlay structure. More generally, structured overlays have also been considered as a basic middleware service to applications [Dabek et al. 2003]. Another issue in common with our own work is that graph-theoretic approaches have been developed for further analysis [Loguinov et al. 2003]. vanrenesse.tocs.2003 [Renesse et al. 2003] needs also be mentioned as a hierarchical (and therefore structured) overlay which although applies (non-uniform) gossip to increase robustness and to achieve self-healing properties, does not even attempt to implement or apply a uniform PEER SAMPLING SERVICE. It was designed to support hierarchical information aggregation and dissemination.

3.9. CONCLUDING REMARKS

Gossip protocols have recently generated a lot of interest in the research community. The overlays that result from these protocols are highly resilient to failures and high churn rates. The underlying paradigm is clearly appealing to build large-scale distributed applications.

This chapter factored out the abstraction implemented by the membership mechanism underlying gossip protocols: the PEER SAMPLING SERVICE. The service provides every peer with (local) knowledge of the rest of system, which is key to have the system converge as a whole towards global properties using only local information.

We described a framework to implement a reliable and efficient PEER SAMPLING SERVICE. The framework itself is based on gossiping. This framework is generic enough to be instantiated with most current gossip membership protocols[Eugster et al. 2003b; Jelasity et al. 2003; Stavrou et al. 2004; Voulgaris et al.

2005]. We used this framework to empirically compare the range of protocols through simulations based on synthetic and realistic traces as well as implementations. We point out the very fact that these protocols ensure local randomness from each peer's point of view. We also observed that as far as the global properties are concerned, the average path length is close to the one in random graphs and that clustering properties are controlled by (and grow with) the parameter H. With respect to fault tolerance, we observe a high resilience to high churn rate and particularly good self-healing properties, again mostly controlled by the parameter H. In addition, these properties mostly remain independent of the bootstrapping approach chosen.

In general, when designing gossip membership protocols that aim at randomness, following a push-only or pull-only approach is not a good choice. Instead, only the combination results in desirable properties. Likewise, it makes sense to build in robustness by purposefully removing old links when exchanging views with a peer. This situation corresponds in our framework to a choice for $H > 0$.

Regarding other parameter settings, it is much more difficult to come to general conclusions. As it turns out, tradeoffs between, for example, load balancing and fault tolerance will need to be made. When focusing on swapping links with a selected peer, the price to pay is lower robustness against node failures and churn. On the other hand, making a protocol extremely robust will lead to skewed indegree distributions, affecting load balancing.

To conclude, we demonstrated in this extensive study that gossip membership protocols can be tuned to both support high churn rates and provide graph-theoretic properties (both local and global) close to those of random graphs so as to support a wide range of applications. A complementary research direction would be to explore this spectrum theoretically.

CHAPTER 4

From Randomness to Structure: VICINITY

In the previous chapters we explored gossiping as a means to create randomness. We are now taking a shift and investigate how gossiping can be harnessed to create structure.

A crucial issue in nearly all distributed systems is the way nodes interact with each other. In particular, it is the "network of acquaintances" that determines the flow of communication, and plays a significant role in the overall system's behavior. Unlike some distributed systems, such as the PEER SAMPLING SERVICE, which impose no preference on *which* pairs of nodes to link, a number of systems entail a specific organization of nodes in certain topologies.

Distributed systems that impose a certain organization on nodes include two main classes. First, *structured* distributed systems, where nodes are linked in a well-defined and deterministic way, according to some property such as their ID, their position, etc. Distributed Hash Tables form a typical example of this class. The second class involves the group of *unstructured* overlays that exhibit some loose form of organization. In these systems, the connectivity of nodes is not strictly mandated, nevertheless certain links are favored over others. An example of this class are systems that cluster nodes demonstrating some sort of relationship or similarity, to enhance peer collaboration. In a file-sharing system for instance, forming links between nodes of similar interests can dramatically boost the ability to find requested files. It should be noted that, semantic clustering for the file-sharing scenario gave us the initial stimulation leading to the conception and development of the research presented in this chapter, and will be further studied in Chapter 7.

A number of customized solutions exist for individual applications. This can be seen, for instance, in Distributed Hash Tables [Rowstron and Druschel 2001a;

Zhao et al. 2001; Stoica et al. 2001; Ratnasamy et al. 2001a]. Each of them employs its own, tailored-made mechanism to build and maintain its specific structure.

In this chapter we present a *generic* topology construction framework, suitable for the construction of a large set of topologies. Through our framework, nodes flexibly and efficiently self-organize in a completely autonomous fashion to a largely arbitrary structure. Such structures may include semantic-based overlays, super-peer topologies, structured overlays such as rings, DHTs, etc., or various other types of topologies. Among the advantages of our approach are its generic applicability, its flexibility, and its simplicity. Note that some other work has been proposed along these lines, namely T-Man [Jelasity and Babaoglu 2005, 2006, 2004], which is further discussed in the related work section (4.8).

4.1. DESIGN PREAMBLE

The problem we are faced with is a *P2P topology construction* problem, namely the construction of overlays representing relationships between nodes. Nodes demonstrating a strong relationship should be grouped in highly clustered communities. That is, they should either be directly linked to each other, or be reachable via a small number of intermediate—also related—nodes. Nodes of composite interests may form links to multiple communities. In fact, communities' boundaries need not be strict, and the very notion of communities may be a bit fuzzy. Rather than partitioning the network in well-defined, isolated groups, we aim at forming a continuous mosaic of interconnected communities. In such an overlay, peers lying within a node's close vicinity are likely to be related to that node and among themselves. Such clustering of nodes may be required in the context of some application, such as the ones presented in Chapters 6, 7, and 8.

However, to handle dynamics requires the discovery and propagation of changes that may happen *anywhere* in the network. For this reason, overlay networks should also reflect desirable properties of random graphs and complex networks in general [Albert and Barabási 2002; Newman 2002]. These two conflicting demands generally lead to complexity when integrating solutions into a single protocol.

Protocols for topology construction in peer-to-peer networks should separate these concerns. In particular, we advocate that when it comes to organizing nodes in a relationship-based overlay, links between nodes should be optimal with respect to their relationships only, regardless of any other desirable property of the resulting overlay. Instead, a separate protocol should be used to handle network dynamics, and provide up-to-date information that will allow proper adjustments

in links, and thus leading to adjustments in the overlay network itself.

4.2. THE TOPOLOGY CONSTRUCTION FRAMEWORK

Per our discussion above, we suggest a topology construction framework composed of two layers. The top layer consists of a gossip-based protocol, VICINITY, that strives to optimize links with respect to the target structure. The bottom layer comprises the PEER SAMPLING SERVICE, studied in Chapter 3. It maintains the overlay connected in the face of node churn and failures, and feeds the top layer with network changes.

In this section we introduce our model, describe how target topologies are expressed, and justify the two-layered approach.

4.2.1. Model Outline

We assume a connected network infrastructure, supporting routing between any two nodes. Each node maintains a dynamic list of neighbors, called its VICINITY *view* V_{vic}, of fixed small length. The *view length*, ℓ_{vic}, is the same for all nodes.

Knowledge regarding neighbors is stored and exchanged by means of *node descriptors*. A given node's descriptor can only be created by that node exclusively. A node descriptor referring to peer P is a tuple containing the following three fields:

1. P's contact information (i.e., network address and port)
2. A numeric *age* field
3. P's application-specific *profile*

Note that a VICINITY node descriptor is essentially a CYCLON node descriptor augmented by the node's application-specific profile. As we will see, a node's profile determines its neighbors in the target structure.

The goal is to organize all VICINITY views so as to approximate the target structure as closely as possible. To this end, nodes regularly exchange node descriptors to gradually evolve their views towards the target. When gossiping, nodes send each other a subset of their views, of fixed small length g_{vic}, known as the *gossip length*. The gossip length is the same for all nodes.

4.2.2. The Selection Function

We consider a *selection function* $S(k, P, \mathcal{D})$, that, given the descriptor of peer P and a set $\mathcal{D}$ of peer descriptors, returns the set of k descriptors (or all of them,

if $|\mathcal{D}| < k$) that best approximate P's outgoing links in the target structure. The selection is based on node profiles. We assume function S to be globally known by all nodes in the system.

The selection function essentially defines the target structure. Each peer P aims at eventually establishing links to the "best" ℓ_{vic} peers, as defined by the outcome of $S(\ell_{vic}, P, \mathcal{D}_P^*)$, where $\mathcal{D}_P^*$ is the set of descriptors of all nodes in the network excluding P.

Often, the selection function S is based on a globally defined peer proximity metric. That is, $S(k, P, \mathcal{D})$ sorts all descriptors in $\mathcal{D}$ with respect to their proximity to peer P, and selects the k closest ones. Typical proximity metrics include semantic similarity, ID-based sorting, domain name proximity, geographic- or latency-based proximity, etc. Some applications may apply composite proximity metrics, combining two or more of the above. In certain cases, though, selecting appropriate neighbors involves more than a mere sorting based on some metric, typically when a peer's significance as a neighbor depends not only on the peer's proximity to a given node, but also on which *other* peers are being selected.

We assume that the selection function S exhibits some sort of *transitivity*, in the sense that if node P_2 is a "good" selection for node P_1 ($P_1 \overset{S}{\dashrightarrow} P_2$), and P_3 is a "good" selection for P_2 ($P_2 \overset{S}{\dashrightarrow} P_3$), then P_3 *tends* to be a "good" selection for P_1 too ($P_1 \overset{S}{\dashrightarrow} P_3$). Generally, the "better" a selection node Q is for node P, the more likely it is that Q's "good" selections are also "good" for P.

This transitivity is essentially a correlation property between nodes sharing common neighbors, embodying the principle "my friend's friend is also my friend". Surely, this correlation is fuzzy and generally hard to quantify. It is more of a desired property rather than a hard requirement for our topology construction framework. The framework excels for networks exhibiting strong transitivity. However, its efficiency degrades as the transitivity becomes weaker. In the extreme case that no correlation holds between nodes with common neighbors, related nodes eventually discover each other through random encounters, although this may take a long time.

4.2.3. Design Rationale

From our previous discussion, we are seeking a means to construct, for each node and with respect to the given selection function, the optimal view from all nodes currently in the system. There are two sides to this construction.

First, based on the assumption of transitivity in the selection function S, a peer should explore the nearby peers that its neighbors have found. In other words, if P_2 is in P_1's VICINITY view, and P_3 is in P_2's view, it makes sense to check whether P_3 would also be suitable as a neighbor of P_1. Exploiting the transitivity in S

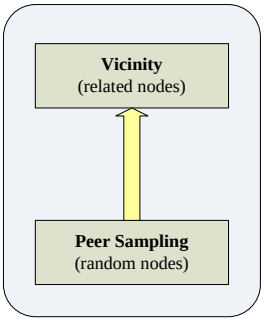


Figure 4.1: The two-layered framework

should then quickly lead to high-quality views. The way a node tries to improve its VICINITY view resembles *hill-climbing* algorithms. However, instead of trying to locate a single optimal node, here the objective is to optimize the selection of a whole set of nodes, namely the view. In that respect, VICINITY can be thought of as a distributed, collaborative hill-climbing algorithm.

Second, it is important that *all* nodes are examined. The problem with following transitivity alone is that a node will be eventually searching only in a single cluster of related peers, possibly missing out on other clusters of also related—but still unknown—peers, in a way similar to getting locked in a local maximum in hill-climbing algorithms. Analogously to the special "long" links in small-world networks [Watts 1999], a node needs to establish links outside its neighborhood's cluster. Likewise, when new nodes join the network, they should easily find an appropriate cluster to join. These issues call for a randomization of candidates for including in a view.

In our design we decouple these two aspects by adopting a two-layered set of gossip protocols, as can be seen in Figure 4.1. The lower layer is the PEER SAMPLING SERVICE, responsible for maintaining a connected overlay and for periodically feeding the top-layer protocol with nodes uniformly randomly selected from the whole network. In its turn, the top-layer protocol, called VICINITY, is in charge of discovering peers that are favored by the selection function. Each layer maintains its own, separate view, and communicates to the respective layer of other nodes, as shown in Figure 4.2.

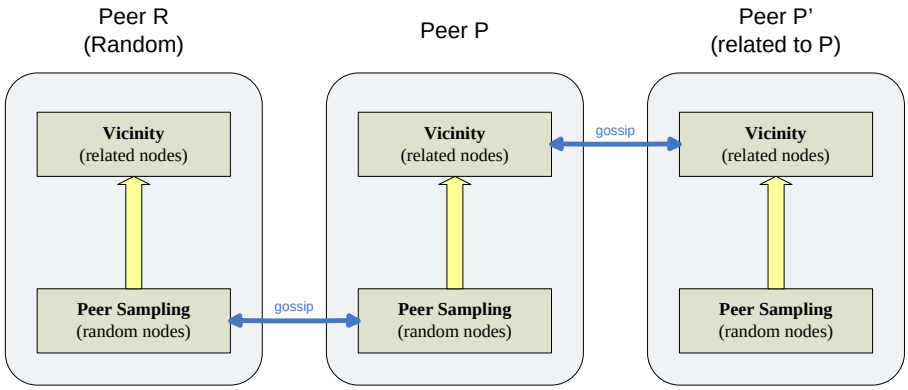


Figure 4.2: Communication in the two-layered framework. Each layer gossips to the respective layer of other nodes.

4.3. THE VICINITY PROTOCOL

VICINITY employs periodic gossiping in a way similar to CYCLON. The key difference lies in the selection of *which* descriptors to keep in one's view following a view exchange. Unlike CYCLON, in VICINITY nodes do not *swap* neighbors when gossiping. They, instead, *send* each other a few descriptors, and each of them decides independently which ones to keep and which to discard to optimize its own view. As mentioned in Section 4.2.1, the number of descriptors sent is defined by the gossip length g_{vic}, and is the same for all nodes. The decision on which links to send is based upon the selection function S.

Like in CYCLON, nodes initiate view exchanges periodically, yet not synchronized. Each node P initiates gossiping once every T time units, by executing the following nine steps:

1. Increase the age of each neighbor by one.
2. Select neighbor Q with the highest age among all neighbors, and remove it from the VICINITY view: $V_{vic} = V_{vic} - \{Q\}$.
3. Merge the VICINITY and PEER SAMPLING SERVICE views in one: $V_P = V_{vic} \bigcup V_{pss}$.
4. Add own descriptor with own profile and age 0 to the merged view: $V_P = V_P \bigcup \{P\}$.
5. Strip down V_P to its g_{vic} best descriptors for Q, by applying the selection function from Q's perspective: $V_P = S(g_{vic}, Q, V_P)$.

```
do forever
{
  wait(T time units)
  incr. all descriptors' age by 1
  Q ← selectPeer()
  remove Q from view
  buf_send ← selectToSend()
  send buf_send to Q
  receive buf_recv from Q
  view ← selectToKeep()
}
```

Active thread

```
do forever
{
  receive buf_recv from Q
  buf_send ← selectToSend()
  send buf_send to Q
  view ← selectToKeep()
}
```

Passive thread

Figure 4.3: The generic gossiping skeleton for CYCLON and VICINITY.

6. Send V_P to peer Q.

7. Similarly, receive V_Q from peer Q, containing a set of (up to) g_{vic} descriptors known by Q, optimally selected for P.

8. Merge the VICINITY, PEER SAMPLING SERVICE, and received views in one: $V = V_{vic} \bigcup V_{pss} \bigcup V_Q$.

9. Rebuild the VICINITY view by selecting the best ℓ_{vic} neighbors from V: $V_{vic} = S(\ell_{vic}, P, V)$.

The receiving node Q executes all steps from 3 on in a symmetric way. Note that when merging views (steps 3 and 8), in the presence of multiple descriptors of the same node only the one with the lowest age is kept.

Each node essentially runs two threads. An *active* one, which periodically wakes up and initiates communication to another peer, executing steps 1 through 9. Another thread, a *passive* one, responds to the communication initiated by another peer, and executes steps 3 through 9.

Note that the VICINITY protocol can also be modeled by the generic gossiping skeleton we introduced for CYCLON (see Fig. 2.2). Figure 4.3 reproduces the same skeleton here for convenience. Figure 4.4 lists the description of each hook of the skeleton, for the VICINITY protocol.

Hook	Action taken
`selectPeer()`	Select descriptor with the oldest age
`selectToSend()`	Merge the VICINITY and PEER SAMPLING SERVICE views. Add own descriptor with own profile and age 0. Select the best g_{vic} descriptors for Q.
`selectToKeep()`	Merge the VICINITY, PEER SAMPLING SERVICE, and received views. Select the best ℓ_{vic} descriptors for P.

Figure 4.4: Implementation of the generic gossiping skeleton hooks, for the VICINITY protocol.

4.4. DISCUSSION ON THE DESIGN CHOICES

A number of interesting design choices are veiled behind VICINITY's 9 simple steps.

To start with, we use the *tail peer selection* policy, as defined in Section 3.2.3. That is, nodes always select their neighbor with the highest age to gossip with. The motive behind this policy is twofold. First, in a way similar to CYCLON, it serves garbage collection of obsolete node descriptors. A descriptor may become obsolete as a result of network dynamics, either because the node it points at is no longer alive, or because new—"better"—neighbors have been discovered, pushing that descriptor off the list with the ℓ_{vic} optimal neighbors. Getting back to the protocol, when—in step 2—node P picks neighbor Q with the highest age in P's view, it also *removes* Q from its view. If Q is alive, it generates a new descriptor with age 0 when executing step 4, and sends it back to P in step 7. At step 9, node P reestablishes a fresh link to Q, if the latter still qualifies as one of the best ℓ_{vic} neighbors. If not, or if Q has not responded at all, it simply remains out of P's VICINITY view, essentially having been garbage collected. Selecting always the neighbor with highest age to gossip with, ensures that no obsolete neighbor lingers indefinitely in a node's view.

The second reason for adopting tail peer selection, is that it imposes a round-robin-like screening of a node's neighbors. After a node's oldest neighbor has been contacted for gossiping, its new descriptor—if it remains a neighbor at all—has age 0, that is, it has the lowest priority among all neighbors in being contacted for a future gossip exchange. Consequently, a node rotates through its view, contacting its neighbors in a roughly circular order. This improves a node's chances to optimize its view faster, by increasing the number of *different* potentially good

neighbors the node encounters. It is not hard to envisage that probing a single neighbor multiple times in a short time frame has little value, as the neighbor is unlikely to have new useful information to trade every time. In contrast, maximizing the intervals at which a given neighbor is probed, maximizes the potential utility of each gossip exchange. Given the rather static nature of a node's VICINITY view (contrary to a PEER SAMPLING SERVICE view), this is achieved by visiting neighbors in a round-robin fashion. Our experiments during the initial development of VICINITY have shown this effect.

Another point deserving attention is the role of the PEER SAMPLING SERVICE layer, and the way it interacts with VICINITY. Its contribution comes about in two places: In steps 3 and 8, descriptors from the PEER SAMPLING SERVICE view are incorporated as candidates for being sent to the other peer, and for being selected for the node's updated VICINITY view, respectively.

In essence, the PEER SAMPLING SERVICE offers an alternative source of links to randomly chosen nodes all over the network. These random links are crucial for target topologies splitting the network in separate, disjoint clusters. In the converged state of such an overlay, every node's VICINITY links point at nodes within the same cluster. A new node that happens to be connected in a wrong cluster (e.g., a newly joined node) would be "trapped" in that cluster unless random links existed to nodes outside that cluster. Even for target topologies that consist of a single, connected cluster, random links play an important role. When such a network is converged, each node's VICINITY links point at peers close to its nearby neighborhood. A newly joined node that joins at a random node away from its neighborhood, will take a large number of steps to slowly "crawl" to its target position. However, if random links are available, it is very likely to find some suitable long-range links to "fly" him quickly close to its neighborhood.

In the following section we will demonstrate the importance of the random links provided by the PEER SAMPLING SERVICE, both for connected and clustered target topologies.

4.5. OUTLINE OF EVALUATION

In this section we will explain how we evaluate the topology construction framework, and present the experimental setting.

4.5.1. Selection of Test Cases

Our topology construction framework constitutes a generic substrate for topology-oriented self-organization. As already mentioned, a multitude of topologies can be constructed by setting the appropriate selection function. A number of examples

are given in the chapters of Part II of this dissertation. The structuring efficiency of our framework is related to the profiles of a given node population and the specifics of the respective selection function and target topology. As a consequence, it is infeasible to provide an exhaustive evaluation of the framework. Instead, the evaluation of the framework for each specific application is left for the respective chapters.

Nonetheless, in this chapter we will attempt to demonstrate our framework's operation by focusing on two test cases underlining its two key functions: first, to gradually improve views of nodes by consulting their nearby neighbors; second, to keep an eye all over the network, in search of related nodes.

Along these lines, we focus on the following two test cases:

Forming a 2-D Spatial Grid Nodes are assigned two-dimensional coordinates, and their goal is to establish links to their closest neighbors. Building the target topology in this test case is primarily based on the Euclidean proximity heuristic. Quite informally, the general idea is that nodes gradually improve their views with closer neighbors, which they then probe to find new, even closer neighbors, eventually reaching their closest ones. This emphasizes the utility of the top layer, VICINITY.

Clustering Nodes in Groups In this test case, nodes are split up in uncorrelated groups. Each node's goal is to cluster with other nodes of the same group. The key difference with the previous test case is that nodes cannot gradually connect to groups "closer" to their own, as there is no notion of proximity between groups. Finding a node of the same group can be accomplished only by means of random encounters, which highlights the role of the lower layer, the PEER SAMPLING SERVICE.

Both test cases are definitely artificial scenarios, selected to demonstrate the individual contribution of each layer. In most real applications, though, such as semantic-based clustering (Chapter 7) and the SUB-2-SUB publish/subscribe system (Chapter 8), both proximity and random heuristics are equally important for efficient self-organization, rendering the combination of the two layers critical.

4.5.2. Generic Experimental Settings

Although VICINITY is not a synchronous protocol, it is convenient to study the evolution of its behavior in time by means of *cycles*. Just like in CYCLON, as well as the generic model of the PEER SAMPLING SERVICE, a cycle is defined as the time period during which every node initiates gossiping exactly once, and, therefore, coincides with the gossiping period ΔT. To further simplify our analysis, in our experiments we set the same gossiping period for both the PEER SAMPLING

SERVICE and VICINITY protocols. Therefore, a cycle is the time period during which *each* node initiates exactly *one* gossip exchange *per protocol*.

Given that establishing random links among nodes is a trivial task for the PEER SAMPLING SERVICE irrespectively of the bootstrapping method (see Chapter 3), we bypass this step and start our experiments with nodes already having random links. More specifically, in all experiments presented in this chapter, both views of each node were initially filled up with descriptors of random other nodes, all with age 0. Such an initialization provides uniformity in the conditions for our experiments, and shields the experiments from behavior irrelevant to VICINITY.

Regarding the PEER SAMPLING SERVICE in the lower layer, we chose to use CYCLON. CYCLON forms an excellent choice for the topology construction framework, as it results in low correlation between the neighbors of a given node (low clustering), which, in turn, maximizes the randomness of the new neighbors received in each gossip exchange.

Finally, all experiments presented in this chapter were carried out with the PeerSim open source simulator for peer-to-peer protocols [PeerSim].

4.6. TEST CASE A: FORMING A 2-D SPATIAL GRID

We consider a two-dimensional space. We assign each node (x,y) coordinates, such that they are (virtually) aligned in a regular square grid organization. A node's coordinates constitute its profile. Each node's goal is to establish links to its four closest neighbors, to the north, south, east, and west.

The natural choice of a selection function for such a target topology is one that gives preference to neighbors spatially closer to the reference node. More formally, we define the distance between two nodes P and Q, with coordinates (x_P, y_P) and (x_Q, y_Q) respectively, to be their two-dimensional Euclidean distance:

$$dist(P,Q) = \sqrt{(x_P - x_Q)^2 + (y_P - y_Q)^2}$$

The selection function $S(k,P,\mathcal{D})$ sorts node descriptors in $\mathcal{D}$ by their Euclidean distance to the reference node P, and returns the k closest ones.

4.6.1. Demonstration of Test Case A

To demonstrate test case A, we simulated a network of 2,500 nodes, forming a 50×50 grid. Figure 4.5 graphically illustrates the evolution of this overlay by depicting its snapshots at different stages. We have arbitrarily chosen small view lengths of 10 descriptors per protocol ($\ell_{vic} = \ell_{cyc} = 10$). We have also set the gossip lengths to be equal to the view lengths ($g_{vic} = g_{cyc} = 10$). This decision

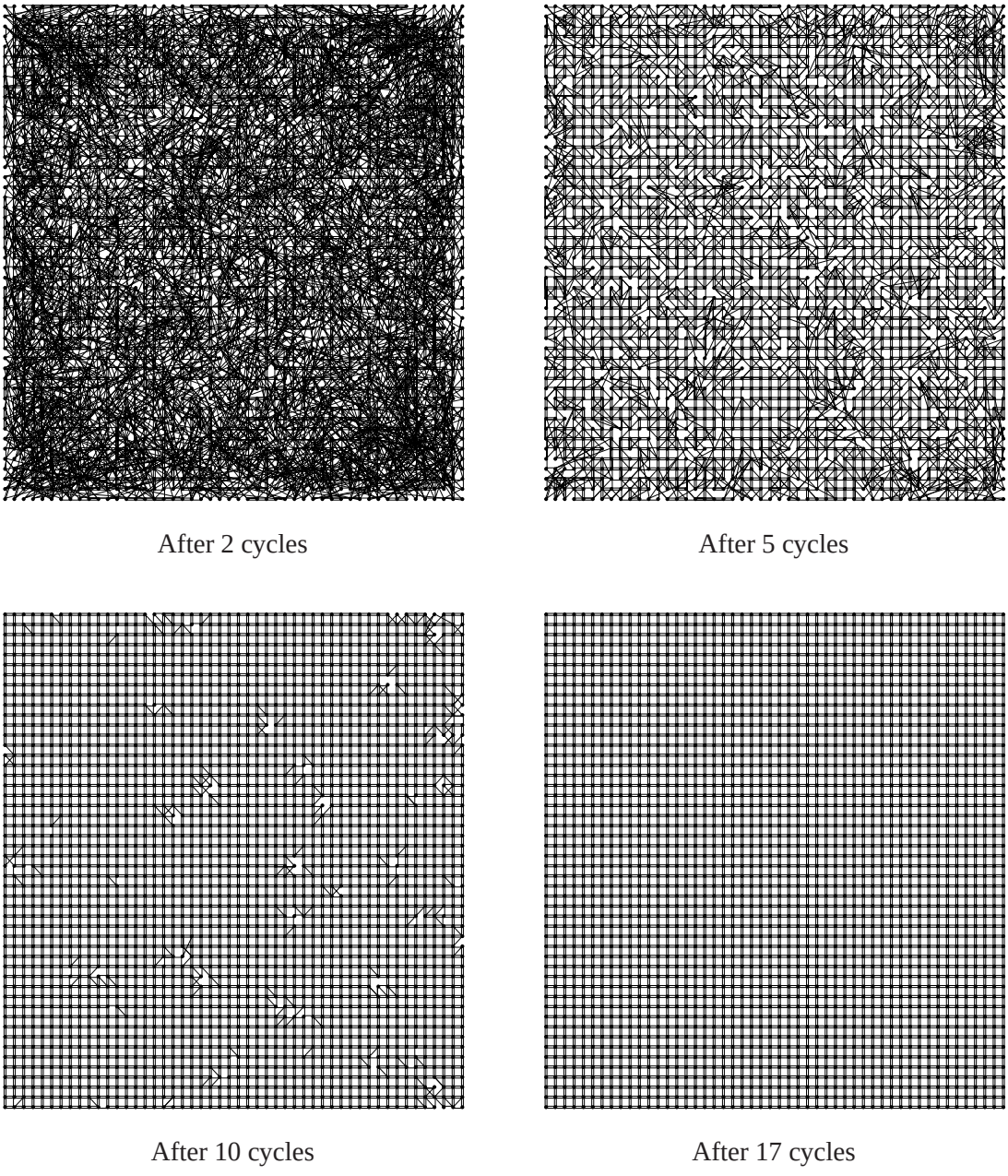


Figure 4.5: Snapshots depicting the evolution of a 2,500 node overlay, forming a 50×50 grid structure.

	$\ell_{vicinity}$	ℓ_{cyclon}
VICINITY/CYCLON	10	10
VICINITY-alone	20	–
CYCLON-alone	–	20

Figure 4.6: The three protocol settings we compare.

was arbitrary, but we opted for small view lengths to underline the framework's power in forming a topology efficiently, even when each node has a very small number of links.

For clarity of the snapshots, only the best four outgoing links of each node's VICINITY view are shown in Figure 4.5, except for edge nodes (three links shown) and corner nodes (two links shown). Note the existence of either one or two lines between two connected nodes. This is because links are directed. A single line means a single link from one node to the other (directionality not shown). A double line means that both nodes have established a link to each other. In the completed target topology (last snapshot) all links are double.

As stated in Section 4.5.2, node views were initialized with random links. This clearly shows in the first snapshot of Figure 4.5, taken only two cycles after the experiment started. In the subsequent snapshots, it can be clearly seen that random, long-range links are gradually being replaced by shorter links to closer neighbors. Already after 10 cycles, most nodes have accomplished their goal, by having established links to their four closest neighbors. Even nodes that have not yet discovered all four best neighbors, have established links to nodes just a couple of hops away. After 17 cycles, all nodes are connected to their four best neighbors. We say that the overlay has converged to the target topology.

4.6.2. Analysis of Test Case A

Having taken a glimpse at the way our framework swiftly builds the target grid topology, let us now observe the experiment's evolution more closely. Additionally, in order to demonstrate each layer's role in building the target topology, it is interesting to also observe the structuring behavior when either one of the two layers operates individually.

Along these lines, we consider two additional protocol settings, with only one of the two layers being active at a time. We compare all three protocol settings over the same overlays. To provide for a fair comparison to the initial experiment, where each layer has a view size and gossip length of 10, the single-layer settings are allotted a view size and gossip length of 20. This way, nodes maintain and exchange the same number of total links in all three settings. Figure 4.6 summarizes

the three protocol settings we compare.

To study an experiment's progress, we record the number of target links that have been established at the end of each cycle. Recall that a node's target links are the four links to its direct north, south, east, and west neighbors. Figure 4.7(a) shows the number of target links that have *not* been placed yet, as a function of the number of cycles elapsed. This initially accounts for 10,000 links (2,500 nodes $\times$ 4 target links), and gradually drops to zero. Similarly, Figures 4.7(b) and 4.7(c) show the evolution of the same experiments for networks of 10K and 40K nodes, respectively.

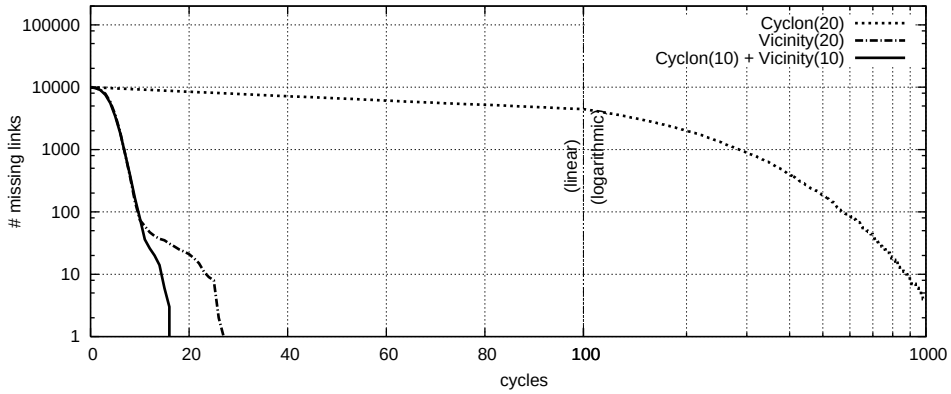
A number of observations can be made from these graphs. Most importantly, we easily identify VICINITY as the primary component responsible for efficient self-organization. More specifically, we see that CYCLON-alone performs several orders of magnitude worse than the other two protocol settings, whose performances are comparable to each other. This indicates that, for the given target topology, the crucial element accelerating self-organization is VICINITY.

Let us now take a look at each protocol setting separately:

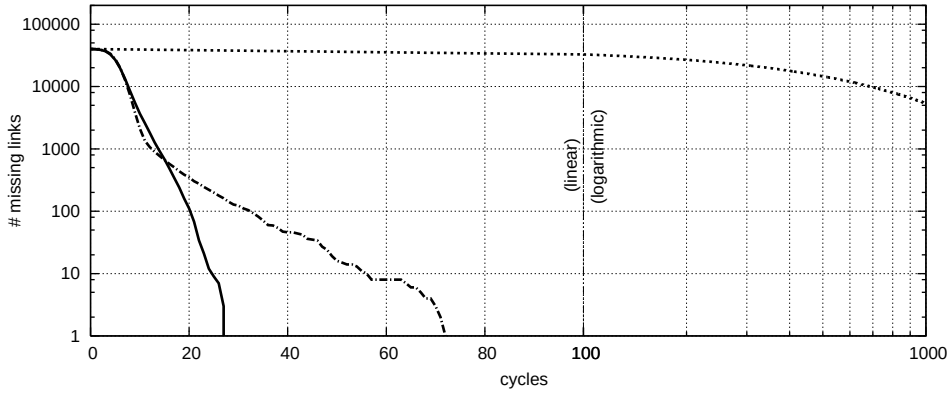
CYCLON-alone It is not hard to see why CYCLON-alone is so inefficient. A node's only hope to find a target link is if that link shows up in its CYCLON view. The CYCLON view is periodically refreshed with random nodes from the whole network. In other words, a node is fishing for target links at blind. As expected, its time to converge increases significantly as the size of the network grows, since the chance of finding a target link at random diminishes.

VICINITY-alone Consider now the VICINITY-alone setting. Like before, a node discovers a target link once it shows up in its view, which is the VICINITY view now. The crucial difference, though, is that a node's VICINITY view continuously *improves*, as opposed to being random. In each cycle some of a node's VICINITY neighbors are replaced by neighbors at least as close as the previous ones. This way, nodes gradually optimize their views, and promptly reach their target neighbors.

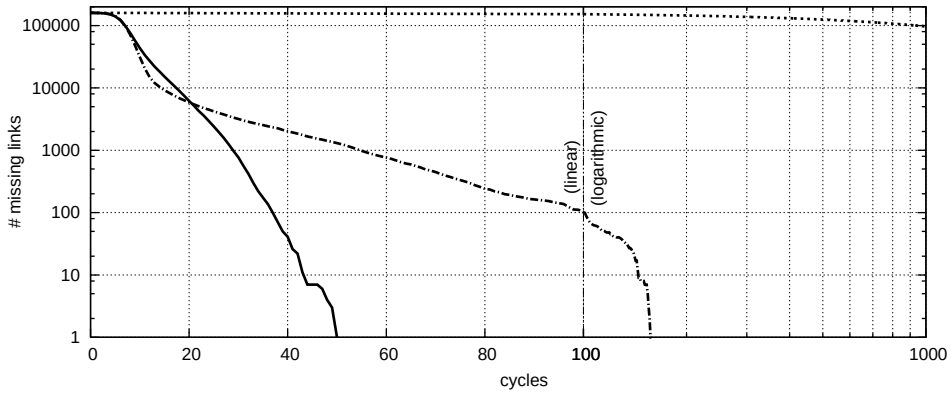
A seemingly minor, yet important observation, is that the progress of VICINITY-alone slows down after some point, for any network size. This can be explained as follows. In these experiments nodes are initialized with a few random links all over the network. These links generally are long-range links. Also, nodes start optimizing their views (by gossiping) simultaneously. Some of these nodes happen to have, or acquire through their neighbors, long-range links that bring them quickly close to their target vicinity, that is, help them find nodes with very close coordinates. Apparently this is the case for the vast majority of nodes, as more than 90% of the target links



(a) Grid of 2,500 nodes (50×50).



(b) Grid of 10K nodes (100×100).



(c) Grid of 40K nodes (200×200).

Figure 4.7: Evolution of topology construction for test case A.

are in place within the first 10 cycles (99% for the 2,500 node network). As a side-effect, the number of long-range links drops dramatically. This hinders the remaining few nodes that have not discovered their proximal neighbors yet, from getting quickly in their respective vicinities. Instead, they have to reach their target vicinities in many, small steps, "crawling" in the almost converged overlay. The number of steps is related to the network diameter, as can be seen in the graphs for different size networks.

VICINITY/CYCLON Quite visibly, the combination of VICINITY and CYCLON outperforms any of the two constituent protocols alone. In fact, its performance comes very close to the performance of VICINITY-alone, except it does not exhibit the sudden progress slowdown we discussed above. The explanation comes straightforward from the relevant discussion. What causes the slowdown in VICINITY-alone is the lack of long-range links. Here we do not face this issue, as the CYCLON layer ensures the continuous existence of random, long-range links.

4.7. TEST CASE B: CLUSTERING NODES IN GROUPS

In this test case, we provide each node with a *group ID*, which constitutes its profile. The goal is to form clusters of nodes that share the same group IDs. From a node's perspective, the goal is to establish links to other nodes with the same group ID.

Note the clear distinction between the terms group and cluster in the context of this test case. A *group* is defined as the set of all nodes sharing a common group ID. A *cluster* is a set of nodes forming a weakly connected (sub-)graph, that is, a connected (sub-)graph considering the undirected version of links between nodes. Nodes should self-organize into clusters, in such a way that clusters eventually coincide with groups.

The only comparison operator defined on node profiles is equality of group IDs. By comparing their profiles, nodes can tell whether they belong to the same group or not. However, no other type of comparison or proximity metrics apply, for example, we do not define any ordering of groups.

The selection function $S(k,P,\mathcal{D})$ is simple and straightforward. It starts by selecting in a random sequence descriptors from $\mathcal{D}$ whose group ID is the same as P's. If these are fewer than k, it continues by selecting randomly from the rest of the descriptors.

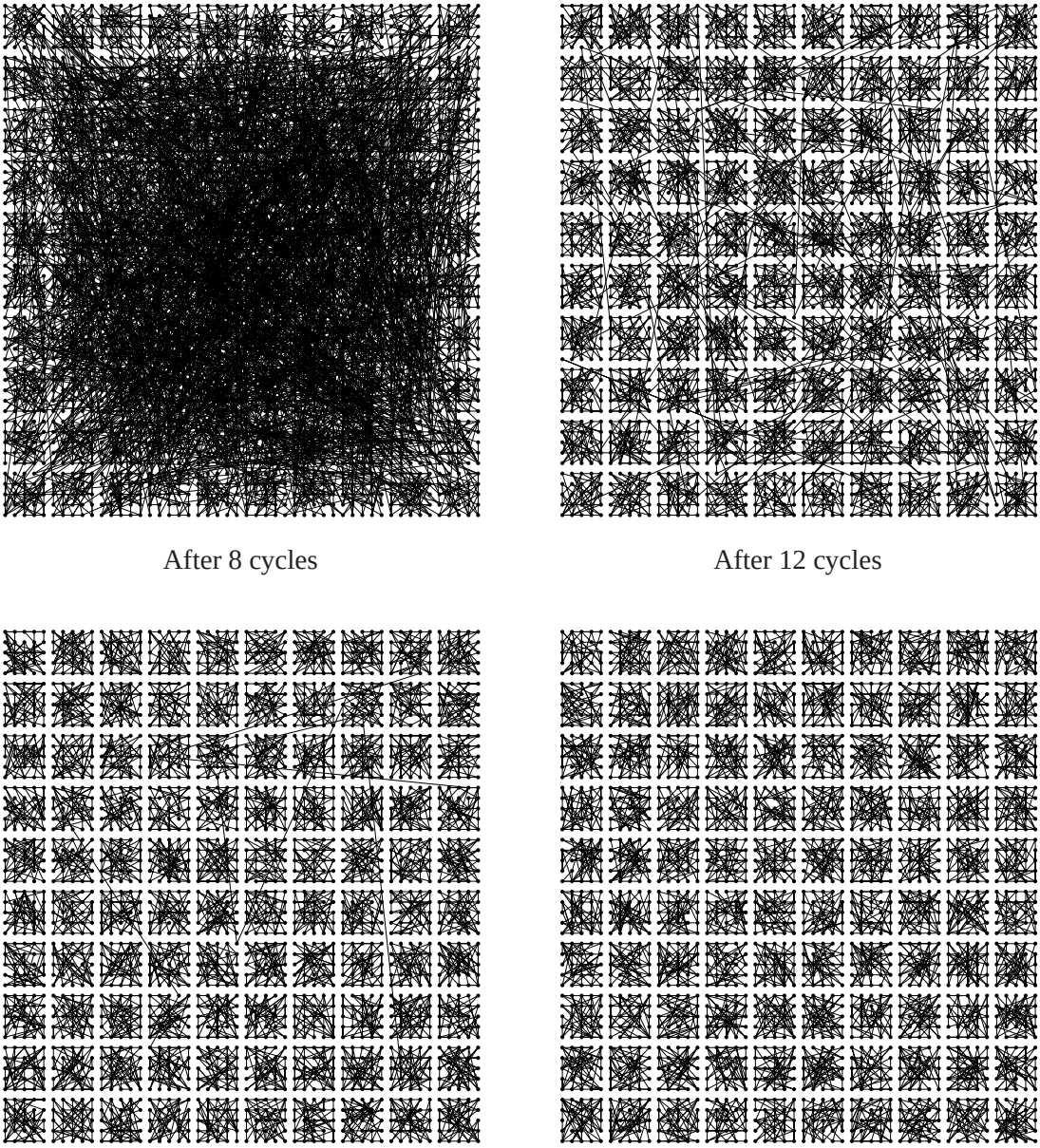


Figure 4.8: Snapshots depicting the evolution of a 2,500 node overlay clustering in 100 groups of 25 nodes each.

4.7.1. Demonstration of Test Case B

Like in the demonstration of test case A, we demonstrate test case B by considering a network of 2,500 nodes. Each node runs VICINITY and CYCLON, both with view lengths and gossip lengths 10 ($\ell_{vic} = \ell_{cyc} = 10$ and $g_{vic} = g_{cyc} = 10$). Nodes are assigned group IDs such that a total of 100 groups exist, each having 25 nodes.

Figure 4.8 presents a series of snapshots of the network, illustrating the evolution of clustering. For presentation clarity, nodes are displayed in a 50×50 grid organization, and all 25 nodes of any given group have been placed in contiguous locations, forming a 5×5 sub-grid. We emphasize that the sole purpose of this layout is to facilitate visual comprehension of clustering, while nodes are not aware of their position in this layout, but only of their group ID.

To avoid cluttering the graph, only 2 random outgoing VICINITY links of each node are shown. However, links to nodes of different groups are drawn with a strictly higher priority than those to the same group. Consequently, when a node in Figure 4.8 appears to have no links to groups other than its own, it is guaranteed that *all* its VICINITY links point at nodes within its group.

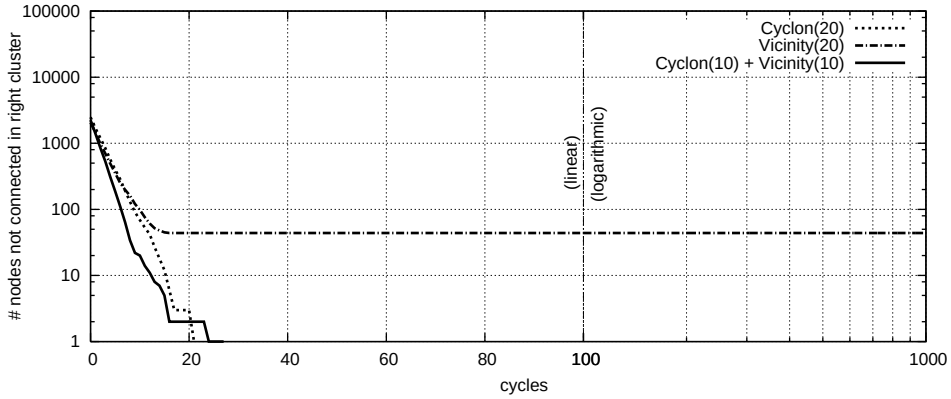
As mentioned in Section 4.5, nodes are initialized with random neighbors. Indeed, after cycle 7 the network still seems to be dominated by random links between different groups. Soon after that, though, the effect of clustering starts becoming evident. Quite visibly, nodes of the same group start forming clusters. Intergroup links are gradually being replaced by intragroup ones, leading (after cycle 22) to a fully clustered overlay, where clusters perfectly match the respective groups. At that point the construction of the target topology is completed.

4.7.2. Analysis of Test Case B

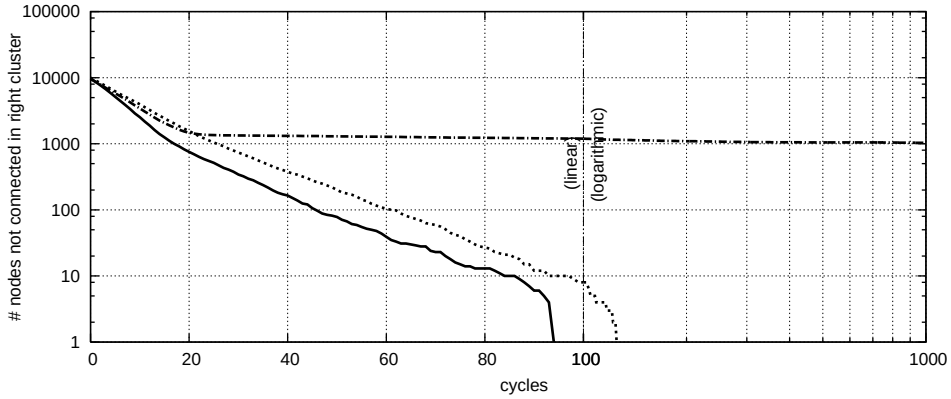
Similarly to the analysis of test case A in Section 4.6.2, we investigate and compare the performance of our two-layered framework to that of each component acting individually. We run experiments for the same three protocol settings, as listed in Figure 4.6.

As stated already, a node's goal is to get clustered with other nodes of its group. This task is divided in two steps: first, discover the right cluster; second, get well connected in it. VICINITY excels in the second. Through a single link to the target cluster, a node rapidly learns and becomes known to additional nodes in that cluster. It turns out that the crucial step in this test case is the first one: discovering the target cluster. Along these lines, we quantify the progress of our experiments by counting the number of nodes that have established at least one link to a node of their group.

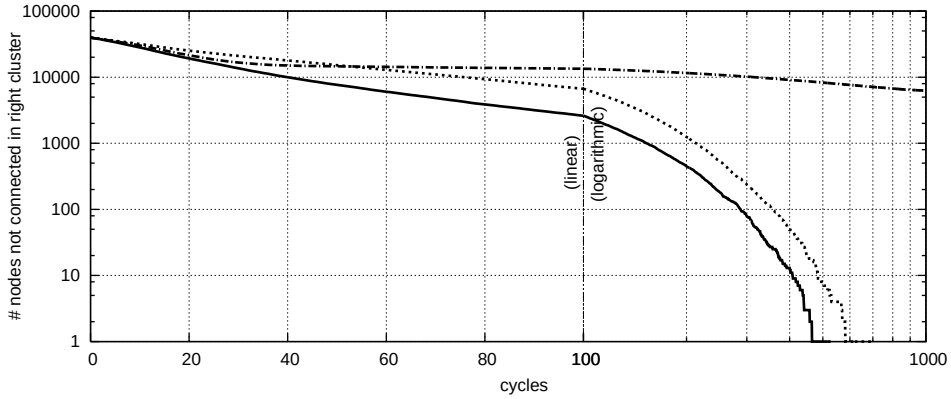
Figure 4.9 plots the complementary metric, that is, the number of nodes that do *not* have any links to other nodes of their group yet, as a function of the number



(a) Network of 2,500 nodes: 100 groups of 25 nodes.

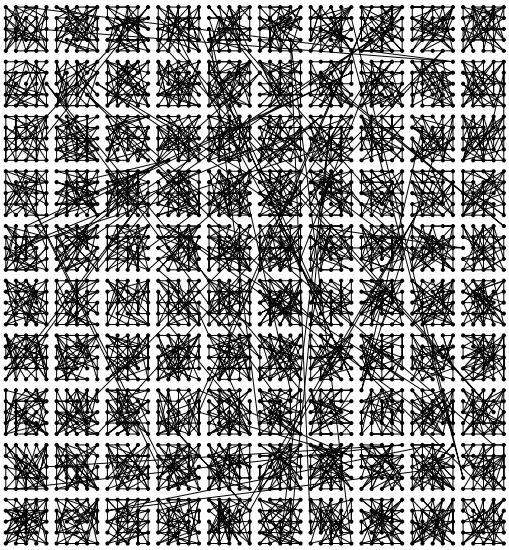


(b) Network of 10K nodes: 400 groups of 25 nodes.



(c) Network of 40K nodes: 1600 groups of 25 nodes.

Figure 4.9: Evolution of topology construction for test case B.



After 1000 cycles

Figure 4.10: Snapshot of the overlay produced by VICINITY-alone. Some nodes are still not connected to their group's cluster after 1000 cycles, and they will never be.

of cycles elapsed. The three graphs correspond to networks of size 2500, 10K, and 40K nodes. In each graph, the progress of all three protocol settings is plotted. In all cases, groups consist of 25 nodes each, so we have 100, 400, and 1600 groups in the networks of 2500, 10K, and 40K nodes, respectively.

The most significant observation is that, unlike test case A, the worst behavior is now exhibited by VICINITY operating on its own. VICINITY-alone stops converging after some point, failing to fully build the target structure. On the contrary, the other two settings, namely, VICINITY/CYCLON and CYCLON-alone, perform comparably to each other. This indicates that in this test case the key component is CYCLON, or more generally, the PEER SAMPLING SERVICE.

Let us now elaborate on each protocol setting separately:

VICINITY-alone Apparently, no matter how long VICINITY runs on its own, some nodes never manage to discover their groups. This is also depicted in Figure 4.10, which shows a snapshot of the network with 2500 nodes, after 1000 cycles of VICINITY-alone execution.

It is not hard to see why this happens. As nodes start clustering with other nodes of the same group, the pool of intergroup links in the network shrinks

significantly. In fact, this is accelerated by the absolute transitivity property this selection function exhibits. Once a node forms a link and gossips to another node of its group, chances are it will acquire links to more nodes of the same group, rapidly trading its intergroup for intragroup links. In not so many cycles, most nodes end up having neighbors from their own group exclusively. Also, clusters start becoming self-contained, that is, their nodes link among themselves, but not to nodes of other groups. Note that this behavior matches the intentions of VICINITY.

The problem comes with nodes that have not come across other nodes of their group early enough. If a node's neighbors are all from other groups, and these groups have already clustered into closed, self-contained clusters, the node has no chances whatsoever to be handed a link to a node of its own group, ever. A neighbor from such a self-contained foreign group can only provide alternative neighbors of that same, foreign group. The node, thus, finds itself in a dead end. Obviously, in our experiments a number of nodes end up trapped outside their groups (see Fig. 4.9).

This demonstrates the need of a source of random, long-range links, to prevent such dead end scenarios. This role is undertaken by the PEER SAMPLING SERVICE.

CYCLON-alone When CYCLON operates alone, descriptors float around in a random fashion. Coming across a node of the same group is purely a matter of luck. As naive as it may seem, it is the only way to discover nodes of the same group, since nothing but the equality operator is defined for comparing the profiles of nodes.

Note that CYCLON-alone and VICINITY-alone perform comparably in the beginning of each experiment. However, later on, the latter is hindered by nodes being already clustered, and thus, not useful for providing fresh information. CYCLON-alone does not suffer from this problem.

VICINITY/CYCLON When we apply our framework in its normal form, that is, VICINITY and CYCLON combined, we get the best results.

The VICINITY/CYCLON protocol setting converges at a rate very similar to CYCLON-alone, except for a number of cycles in the beginning of each experiment, during which it converges faster. The explanation of this behavior lies in the details of gossip exchanges in each setting, and is an interesting topic to elaborate on.

In CYCLON-alone, when a node initiates a gossip exchange it contacts a neighbor, and retrieves 20 new, random neighbors. Its chance to find a node

of its own group in these new neighbors, is roughly 20 over the network size.

In VICINITY/CYCLON, when a node P initiates gossiping it contacts *two* neighbors, Q_c on account of CYCLON, and Q_v on account of VICINITY. It retrieves 10 new neighbors from each of them, giving a total of 20, exactly as many as in the CYCLON-alone case. From Q_c, it retrieves 10 random neighbors. From Q_v, however, it retrieves the 10 *optimal* descriptors out of both its views. Thus, if a descriptor from P's group lay *either* in Q_v's VICINITY *or* Q_v's CYCLON view, it is guaranteed to be one of the 10 descriptors sent from Q_c back to P. As a consequence, P will acquire a neighbor of its own group with a chance of roughly 30 over the network size, which explains the faster convergence over CYCLON-alone. Later on, when nodes start clustering per group, VICINITY views increasingly consist of links to nodes of a single group. Their utility in providing links to a variety of groups diminishes, and P's chance of finding a neighbor from its group drops to roughly 20 over the network size (10 random nodes from Q_c, and 10 more from its Q_v's CYCLON links).

In this test case, we saw that the PEER SAMPLING SERVICE is essential to the successful construction of target topologies that are clustered in disjoint groups. In a more general sense, the PEER SAMPLING SERVICE is necessary to all target topologies for which VICINITY alone could result in nodes becoming permanently trapped in a non-optimal neighborhood.

The necessity of the PEER SAMPLING SERVICE layer becomes more evident when new nodes join an already clustered network. Then, unless they happen to join by chance to the right neighborhood directly, they will never manage to reach it.

4.8. DISCUSSION AND RELATED WORK

In this chapter we presented a framework, by means of which networks can self-organize into a given target topology. The target topology is expressed by means of a selection function that selects which neighbors are optimal for each node. The framework consists of two layers. The top layer, VICINITY, strives for optimizing a node's neighbors with respect to the selection function. The lower layer, an instance of the PEER SAMPLING SERVICE, ensures that each node also maintains some random neighbors.

The most significant advantages of the framework are that it is simple and generic. Simple, because each node acts autonomously, performing a sequence of simple steps and taking local decisions, without requiring the deployment of

any dedicated infrastructure. Generic, because it is applicable to a wide variety of target topologies, given the appropriate selection function. The applicability of the framework will be demonstrated by some applications in Part II of this dissertation, without, though, this being an exhaustive set of potential applications.

On the theoretic side, Kleinberg's results on navigation in small-world networks are highly relevant to our work [Kleinberg 2000b, a, 2001, 2004]. Kleinberg studied the qualitative properties of networks that optimize the effectiveness of decentralized routing algorithms in discovering short paths between two nodes, based exclusively on local routing decisions. He proved that for each overlay there is a critical distribution of long-range links, that allow decentralized routing algorithms construct routing paths of length $O(logn)$. For all other long-range link distributions, decentralized algorithms can only find asymptotically longer routing paths. More specifically, in a k-dimensional lattice network, optimal navigability is achieved when long-range links between nodes of lattice distance r are established with probability proportional to $1/r^k$.

In principle, Kleinberg's results are key to distributed topology construction, as the joining of a new node is essentially a problem of navigating to its optimal neighbors. Let alone that decentralized routing is often the purpose of the target topology itself. In practice, however, these results are not applicable to our work for two reasons. First, unlike our demonstration test cases in Sections 4.6 and 4.7, in most applications nodes are not laid out in a lattice structure (e.g., see applications in Part II). Second, in a number of applications distance is not meaningful between *all* pairs of nodes (e.g., see test case B, Section 4.7, and Chapter 7). Nevertheless, the combination of a number of random long-range links (provided by the PEER SAMPLING SERVICE) with short-range links (provided by VICINITY) appears to provide sufficient topology navigation for all applications presented in Part II. A demanding application can improve its overlay's navigational efficiency by increasing the number of short- and long-range links maintained per node, even if the efficiency achieved is not optimal for that number of links.

Other work aiming at self-organizing a network to a certain topology has concentrated on solutions tailored for very specific problems. Distributed Hash Tables [Rowstron and Druschel 2001a; Zhao et al. 2001; Stoica et al. 2001; Ratnasamy et al. 2001a] are such an example, as each of them includes its own, customized technique for building the respective overlay.

The only other research our work comes close to, is the T-Man protocol [Jelasity and Babaoglu 2005, 2006], which has been developed independently. Although there were significant differences with the original T-Man protocol [Jelasity and Babaoglu 2004], the most recent version shows a strong similarity with our work. A difference remains that T-Man employs the *random peer selection* policy, as opposed to the tail policy used in our work. The benefits of tail peer

selection in finding good neighbors faster are discussed in Section 4.4. An additional difference stemming from the peer selection policy, lies in the way the two protocols do garbage collection. In T-Man, a predefined number of links of highest age are discarded in each cycle, with the motive that good neighbors that are still alive will soon become known again. This is the same garbage collection technique used in the PEER SAMPLING SERVICE (see parameter *H* in Section 3.2.2). Although this is efficient for garbage collection, it results in good neighbors being temporarily forgotten. In our framework we do not explicitly remove neighbors of high age. Instead, we select to contact the neighbor with oldest age (tail peer selection), and remove it from the VICINITY view. If it is still alive, it is reinserted in the view right away. If not, it simply remains removed. This way, a node's *alive* good neighbors remain continuously its neighbors. On a smaller note, our framework provides higher flexibility with respect to the number of descriptors exchanged when gossiping. This has a direct impact on the bandwidth used, notably when node profiles carry bulky information.

Concluding, the framework presented in this chapter constitutes a fundamental, generic protocol. We have already applied it in different areas [Voulgaris and van Steen 2005; Voulgaris et al. 2006], and it is employed in multiple applications throughout this dissertation, where it is also analyzed more specifically.

Part II

APPLICATIONS

CHAPTER 5

Routing Table Management: Building Pastry

One of the fundamental subjects of this dissertation, the PEER SAMPLING SERVICE studied in Chapter 3, introduces a model of communication based on random contacts. The aim is to exploit randomness to disseminate information across a large set of nodes in a simple, robust, and timely manner. Disseminating membership information itself results in highly connected and remarkably robust overlays that are adaptive to network changes, and appear to demonstrate self-healing behavior in the face of major network disasters. And all that comes with a very simple framework, free of any type of centralized structures or administration, operating in a fully autonomous, self-organizing fashion.

A different significant direction of recent research in P2P systems, has been in designing overlay networks for routing. Such systems are widely known as *Distributed Hash Tables* (DHTs). The respective protocols operate in the application layer, on top of an existing network physically interconnecting all nodes (such as the Internet). They assign each participating node an ID, and route messages to a node based on that, rather than based on its IP address. Performance (in terms of routing hops) is usually inferior compared to traditional IP routing, but this is not the point. Their significance lies in the fact that they map ID keys to nodes, in a way similar to traditional hash tables mapping numeric keys to table entries. This function is crucial as a building block for numerous other applications, such as distributed network storage (OceanStore [Kubiatowicz et al. 2000] and PAST [Rowstron and Druschel 2001b]), distributed web caching, etc. A number of DHT systems have been proposed to date, most important representatives being Pastry [Rowstron and Druschel 2001a], Tapestry [Zhao et al. 2001, 2004], Chord [Stoica et al. 2001], and CAN [Ratnasamy et al. 2001a]. Their common property is that they all try to form and maintain some sort of structure across a

large number of participating nodes, that is then used to route packets among them. However, their behavior is uncertain in the presence of highly dynamic environments, or serious disasters (i.e. half of the nodes disconnecting simultaneously), or when bootstrapping a system from scratch.

In this chapter we present an alternative approach to building and managing DHT routing tables, based on the PEER SAMPLING SERVICE. Our approach combines the advantages of DHTs with those of highly fault tolerant, self-healing gossiping networks. More specifically, we focus on the Pastry DHT [Rowstron and Druschel 2001a], and we employ the NEWSCAST instance of the PEER SAMPLING SERVICE to bootstrap and maintain Pastry-like routing tables. We substantiate our claims by presenting experimental results from emulation in a real, wide-area network.

Note that the research presented in this chapter constitutes our initial efforts towards merging the structured with the unstructured worlds. As a historical note, it has preceded the VICINITY protocol, which is more powerful in building and maintaining topologies. Nevertheless, this research deserves a place in this dissertation, as an illustration of our first step in harnessing randomness to create structure.

5.1. PASTRY-LIKE P2P ROUTING

Two of the most popular DHTs, Pastry and Tapestry, perform routing based on the same concept: incrementally matching the destination's ID, digit by digit. In this section we present the organization and function of the principal structures used for routing in these systems, the *routing tables*.

5.1.1. Basic Concept

Each node is assigned a unique numeric identifier, its *node ID*, or simply *ID*. When presented with a message and a numeric *key*, a node routes the message towards the node whose ID is equal to the given key. Node IDs and keys are N-bit integers, forming a node ID space that spans from 0 to $2^N - 1$. N has a typical value of at least 64 to provide a sufficiently large ID space to accommodate possibly billions of nodes. Nodes pick their IDs randomly with uniform probability from the set of N-bit strings. It is, therefore, assumed that IDs are uniformly distributed across all geographic regions, multiple jurisdictions, and various networks.

5.1.2. Internal Structure of the Routing Tables

For the purpose of routing, node IDs and keys can be thought of as a sequence of digits in base 2^b (b-bit long digits), where b is a configuration parameter with typical value 4 (which implies *hexadecimal* digits). Routing a message to its destination is achieved gradually, by matching one additional digit of the message's key at a time, say, from left to right. That is, in each step the message is normally forwarded to a node whose ID shares with the key a prefix at least one digit (b bits) longer than the prefix the key shares with the present node's ID, if such a node is known. If such a node is not known, routing of that message fails.[1]

To implement the logic described above in message routing, each node maintains its *routing table*. The routing table of a node consists of N/b rows of 2^b entries each. An entry contains the ID of a node, and its corresponding IP address. A given row of the routing table contains 2^b entries, and represents a matching prefix in the node ID up to a digit position. Entries in the r-th row ($r \in \{1,\ldots,N/b\}$) contain nodes whose IDs share the same $(r-1)$-digit prefix with the present node. The c-th entry of the r-th row contains such a node, with the additional constraint that its ID's r-th digit is equal to c. For instance, assuming b=4 (hexadecimal digits for the node ID), the 2nd entry of the 3rd row of the routing table for node 437BF52... ($N/4$ hex digits in total) is some node whose ID starts with 43**2**, while the 8th entry of its 5th row has a node whose ID starts with 437B**8**.

5.1.3. Routing

Upon receiving a message, a node compares the message's key to its node ID. If they share a common prefix of i digits, it should forward it to a node whose ID shares a prefix of $i+1$ digits with the key. To accomplish that, the present node looks up the $(i+1)$-th row of its routing table, which contains nodes sharing with the key the same i first digits. Out of that row, it picks the k-th entry, where k is the value of the key's $(i+1)$-th digit, and forwards the message to that node. That node not only shares with the key the same first i digits, but also the $(i+1)$-th one. This process continues either until the node whose ID matches all digits of the message's key is reached, or, else, until the message cannot be forwarded any further.

[1] In fact, current systems consult auxiliary structures that they maintain (such as the *leaf set* in Pastry), which will not be considered in this chapter.

5.2. BUILDING ROUTING TABLES

An important issue in DHT-based peer-to-peer systems is managing the routing tables. These tables are kept up-to-date by having nodes that join or leave the system contact other nodes explicitly. To handle failures, heartbeat algorithms are used to probe nodes and to take measures when a failure is detected.

We propose a different approach, namely to separate routing from table management, similar to the separation deployed in Internet routing protocols such as OSPF [Moy 1994] or RIP [Hendrick 1988]. We believe such a separation often leads to a cleaner and simpler design, although sometimes at the cost of performance.

In Chapter 3 we showed that by means of the PEER SAMPLING SERVICE an overlay can handle dynamics and stay up-to-date in a lazy fashion. For DHT-based peer-to-peer systems, we propose to deploy the PEER SAMPLING SERVICE for maintaining routing tables. Our method is completely decentralized, highly robust, and quickly adjusts itself to major changes in the network. These advantages come at the price of continuous bandwidth consumption.

5.2.1. The Principal Idea

The PEER SAMPLING SERVICE has a number of important properties, as described in section 2. It maintains a strongly connected overlay, sustains disasters, adapts fast to (possibly major) network changes, and is highly scalable. The goal is to combine its adaptivity strength with the efficiency of the routing scheme presented in Section 5.1, to create a robust, highly fault resilient, peer-to-peer overlay network for efficient routing.

The principal idea is to harness the knowledge of random neighbors to populate routing table entries. Every node's view is periodically refreshed with a number of new neighbors, chosen at random among *all* the participating nodes. All a node has to do, is *keep* those—randomly acquired—neighbors that are suitable for filling up its routing table entries.

A node's routing table consists of N/b rows. Let us first concentrate on building the first row. Considering a division of nodes in classes based on their ID's *first* digit, we have 2^b classes (since a digit is b bits long). The first row requires one arbitrary representative from each class (excluding the present node's class). Since node IDs are evenly spread across the ID space, each class accounts for roughly $\frac{1}{2^b}$ of the nodes. Therefore, with reasonably high probability, a node will have come across at least one representative from each class when a few more than 2^b random nodes become known to it. Assuming $2^b = 16$ (for $b = 4$) and a view size $\ell_{pss} = 20$, this could even happen in one cycle, or, otherwise, in a couple more. So, each node's first row can be filled up in a matter of a couple of cycles

by pulling suitable neighbors from the dynamically refreshed PEER SAMPLING SERVICE view.

For the second routing table row of a node P, we consider the set of nodes whose IDs start with the same first digit as P's ID, and we split them in classes based on their ID's *second* digit. Again, we have 2^b classes. However, each of these classes corresponds to $\left(\frac{1}{2^b}\right)^2$ of the whole network, one order of magnitude smaller fraction than the respective classes based on the first digit. Generally, the respective classes for row k correspond to $\left(\frac{1}{2^b}\right)^k$ of the nodes, which becomes a very small fragment of the network for higher values of k. Instead of relying on random nodes from the *whole* network to hit upon representatives of these classes, we pick random nodes from a pool containing *only* nodes having IDs with a given first digit. Such a pool is realized by a separate instance of the PEER SAMPLING SERVICE, involving only nodes whose IDs start with the same first digit as P's ID. In that pool, each class corresponds to $\frac{1}{2^b}$ of its nodes, so representatives of all classes are discovered rapidly, like representatives for the first row. Similarly, additional instances of the PEER SAMPLING SERVICE are employed for the efficient discovery of neighbors suitable for the rest routing tables rows. This leads us to the multilayer PEER SAMPLING SERVICE architecture explained in the following section.

5.2.2. Multilayer Architecture

To efficiently build and maintain routing tables, we run multiple instances of the PEER SAMPLING SERVICE. A single physical node can participate in several of them, by running a separate *agent* (i.e., software acting as a virtual peer) for each one, maintaining its own, separate view of neighbors, and gossiping independently from the other agents. In fact, each node runs exactly N/b agents, participating in an equal number of PEER SAMPLING SERVICE instances. Each agent is responsible for maintaining one of the N/b rows in the node's routing table. The agent responsible for row $r \in \{1,\ldots,N/b\}$ of node P will be referred to as *agent #r of node P*.

Each agent of a node sets its own criteria on which neighbors to accept. Agent #i of node P maintains *only* neighbors whose IDs start with the same $i-1$ digits as P's ID. Moreover, a node's agent #i gossips *exclusively* with agent #i of other nodes, as shown in Figure 5.1. In other words, when agents #i of two nodes gossip with each other, it implies the respective IDs of their nodes start with (at least) the same $i-1$ digits. Otherwise, neither would have been a neighbor of the other, which is necessary to initiate gossiping. Additionally, the rest of their neighbors have IDs starting with the same $i-1$ digits too. What we see is that, agents #i of nodes that share the same first $i-1$ ID digits, essentially run their own

instance of the PEER SAMPLING SERVICE, forming a cluster among themselves. Consequently, the view of a node's agent #*i* is periodically refreshed with new neighbors of its group, that is, new neighbors whose IDs start with the same $i-1$ digits. That is, a node's agent #*i* provides a source of random nodes from the pool of nodes whose IDs start with a given $(i-1)$-long prefix. Note that agents #1 of all nodes participate in the same PEER SAMPLING SERVICE overlay, including *all* nodes (as they all—trivially—start with the same 0 first digits), thus, maintaining the whole network in a single, connected cluster.

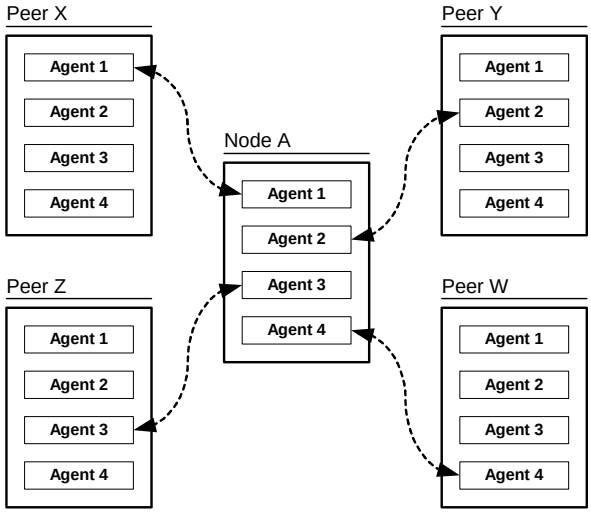


Figure 5.1: Communication of node *A* during one communication cycle.

To facilitate and speed up the clustering of *all* nodes of a given ID prefix in the appropriate PEER SAMPLING SERVICE cluster, we devise the following interaction between a node's agents. Neighbors acquired by agent #*i* of a node, therefore sharing a common $(i-1)$-long ID prefix, could be of interest to agents $\#(i+1)$, $\#(i+2)$, and so on, of that node, if they additionally share the same *i*-th, $(i+1)$-th, etc., ID digits, respectively. In these lines, neighbors that become known to a node's agent #*i* are *also* reported to the same node's agent $\#(i+1)$, $\#(i+2)$, and so on. These agents filter the neighbors received from agent #*i* and keep the ones matching their prefix requirement in their view (by replacing the oldest view items).

An important observation is that once agents #*i* of *all* nodes that share the same first $i-1$ ID digits have formed a single cluster, agents $\#(i+1)$ of the nodes among them that also share an arbitrary same *i*-th digit form a single connected cluster very fast. Each agent #*i* learns about ℓ_{pss} random peers with the same

first $i-1$ digits every ΔT time units. Assuming evenly distributed node IDs, we expect that on average $\ell_{pss}/2^b$ of the peers that become known every ΔT time units share the i-th ID digit too with the present node in addition to the first $i-1$ digits. Given typical parameter values of $\ell_{pss} = 20$ and $b = 2$, one or more peers sharing i digits become known every ΔT time units on average. This partly explains why all agents of every node form clusters quickly, as we shall see later.

Notice that, initially, every node's agent $\#(i+1)$ forms its own (trivial) cluster, disjoint from all the rest. Such a cluster generally expands on each cycle of agent #i, since a random peer satisfying the prefix requirement of agent $\#(i+1)$ is introduced. Moreover, two clusters of n and m nodes unite if any of the n nodes of one of them happens to learn about the existence of any of the m nodes of the other. Therefore, the larger disjoint clusters get, the more likely it becomes they will unite. What we are seeing, is an increasingly accelerating behavior in the process of merging disjoint clusters. It is therefore reasonable to state that agents $\#(i+1)$ of a set of nodes sharing the same first #i digits form a cluster in just a few cycles, provided agents #i of nodes sharing the same first $\#(i-1)$ digits form a cluster too.

As mentioned earlier, all nodes' agents #1 participate in the same instance of the PEER SAMPLING SERVICE, and guaranteeing a (quickly formed) single connected cluster of *all* existing nodes. This helps clusters of agents #2 to form fast too. By induction, and based on the claims of the previous paragraph, we expect all instances of the PEER SAMPLING SERVICE executed by all agents of all nodes, to quickly form the clusters they are designed for.

Our claims are further strengthened by the presentation of emulation analysis in Section 5.4.

5.3. EXPERIMENTAL SETTING

We implemented the architecture described in section 5.2.2 in Java and deployed it on the DAS-2, a 400-processor cluster geographically distributed over a wide-area network across the Netherlands [DAS-2]. We carried out experiments with a set of 65,536 nodes, a number of them running on each DAS-2 processor simultaneously.

We considered node IDs of length $N = 16$ bits, and digits of length $b = 4$ bits (hexadecimal digits). This setting resulted in $N/b = 4$ rows and $2^b = 16$ columns per routing table.

The instance of the PEER SAMPLING SERVICE we chose to employ in this chapter's experiments is the NEWSCAST protocol, described in Section 2.8. Each node was running 4 NEWSCAST agents, one for each of its 4 routing table rows. A

view size of $\ell = 20$ was used for each NEWSCAST agent. We ran our experiments with the same refresh interval of $\Delta T = 10sec$ for all agents. That is, every 10 seconds *each* of the 4 agents of each node initiated a gossip exchange. We recorded and analyzed the behavior of our architecture at intervals of 60 seconds, that is, we logged the whole network's state every 6 communication cycles.

Another facet of our experiments that is worth noting is the *bootstrapping* mechanism. By bootstrapping we refer to the procedure of providing nodes with the information required to jump-start the overlay network's formation. In principle, a new node joins by contacting *any* existing node and gossiping with it. When the whole network starts from scratch, a systematic way has to be present to provide one or more initial communication points to each node. In our experiments, all nodes' agents #1 were provided with the address of one single node's agent #1, forming a star topology. Providing agents with a choice of—possibly random—agents to initially connect to, enhances the randomness of the network from the early cycles. However, a bootstrapping mechanism as simple and centralized as the one we chose further endorses our claims of our architecture's fast convergent behavior, as discussed in the following section.

Finally, we imposed a fake large-scale failure while the experiment was running, in order to observe and analyze the behavior of our system in such cases. In particular, we killed 50% of the nodes in the middle of the experiment. Our observations of the experiments and their analysis are presented in the following section.

5.4. EXPERIMENTAL RESULTS AND ANALYSIS

This section presents the output of our experiments with 65,536 nodes. We recorded and analyzed two aspects of the system's behavior: dynamic forming of the routing tables when bootstrapping, and following a large-scale failure.

5.4.1. Bootstrapping

This experiment aims at observing the system's behavior while bootstrapping. Figure 5.2 presents the system's fast convergence to a fully operative routing substrate. It shows the average number of routing table rows that are completely filled per node, as a function of the number of cycles elapsed from the experiment's start. A node's i-th routing table row being completely filled means that the node can route *any* message whose key shares $i-1$ digits with the node's ID to a peer node whose ID additionally matches the i-th digit of the message's key. Note that the system manages to fill *all* routing table entries in *all* nodes in less than 30 cycles.

Figure 5.3 illustrates the effectiveness of routing messages. From each node

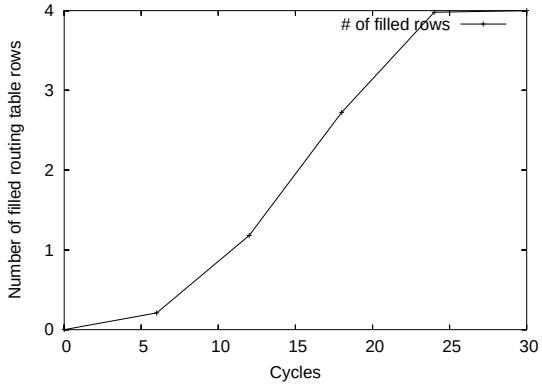


Figure 5.2: Average number of filled routing table rows.

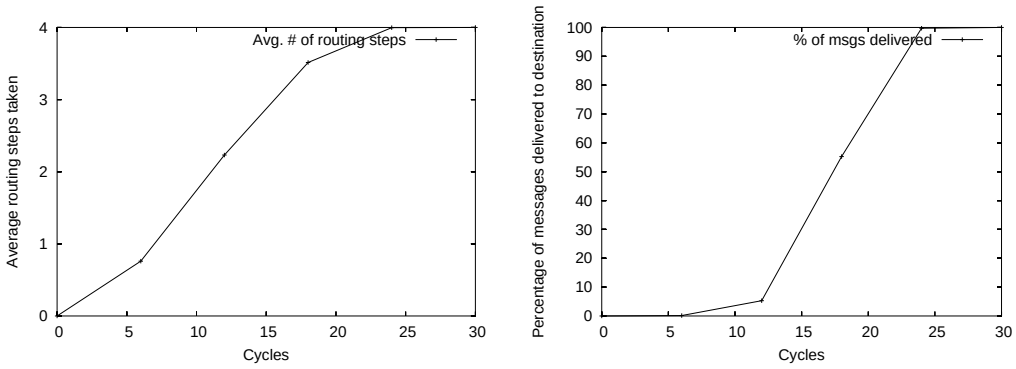
we routed a number of messages to random nodes. Figure 5.3(a) shows the number of routing steps (hops) messages took en route to their destination, on average. Initially, routing tables are empty, so messages cannot take any steps towards their destinations. However, as routing tables start being built, messages follow increasingly more routing steps towards their destinations. This graph is akin to Figure 5.2, as the number of routing steps a message takes is directly dependent on the number of routing table rows that have been filled.

Figure 5.3(b) shows the percentage of messages that manage to reach their destinations, as a function of the number of cycles. For the first 10 cycles few or none of the messages reach their targets. As routing tables are filled, more messages are routed all the way through to their destinations. As it turns out, after the first 24 cycles, 99.74% of the messages are delivered to their destinations, and after 30 cycles, this fraction increased to 99.998%.

5.4.2. Robustness to Large-Scale Failures

To test the system's behavior in the face of large-scale failures, we intentionally killed *half* of the nodes in the network, at some point when all routing tables were guaranteed to be completely filled. That point corresponded to approximately 10 minutes after the start of the experiment. We refer to the cycle when this catastrophic failure occurred as cycle d. More specifically, we killed all nodes with an odd ID. As expected, NEWSCAST maintains the surviving nodes connected in a single cluster. As we will see, the overlay adapts very quickly to reflect the modified network.

Figures 5.4 and 5.5 are analogous to the previous figures, 5.2 and 5.3, respectively. Figure 5.4 shows the average number of routing table rows that are



(a) (b)

Figure 5.3: **(a)** Number of routing steps taken on average; **(b)** Percentage of messages reaching their destination.

completely filled (with valid entries), per node. Note that outdated entries of crashed nodes (the ones with odd IDs) are not considered valid, and therefore are not counted. Immediately after the crash none of the routing table rows are fully filled, which implies that all routing rows of all nodes also contained some entries with odd node IDs. However, as can be seen in the diagram, routing tables are filled very quickly. Within 30 cycles from the point of disaster, the first three routing table rows of all nodes have been filled. Note that this is the maximum number of rows that can be filled per node. The 4th rows of all routing tables cannot be filled, as they would require nodes that match all possible cases for the last digit of their IDs. Since nodes with odd IDs do not exist any more, it is not possible to fill up these rows. This, however, does not affect routing, as routing paths to all existing nodes (i.e. nodes with an even ID) do exist and are complete.

The system's capability to route messages can be seen in Figure 5.5. Figure 5.5(a) shows the average number of hops a message traverses. Initially, since half of the nodes have been removed, messages are routed on average half-way through to their destination. As routing tables are adjusting to the modified network, messages follow more hops towards their destinations. Figure 5.5(b) shows the percentage of messages that reach their destinations. Just like in the bootstrapping case, routing tables are formed very quickly. It takes less than 20 cycles from the moment of the crash to form routing tables that can route any message from *any* source to *any* destination.

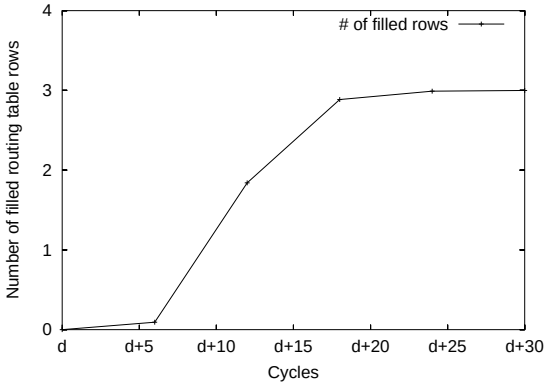


Figure 5.4: Average number of filled routing table rows when recovering from a 50% node crash that happened at cycle d.

5.4.3. Bandwidth Considerations

In this section we provide an estimation of the individual (per node) and aggregate bandwidth used in our experiments. Despite the 16-bit node IDs we used in our experiments, we make the estimation assuming node IDs of 64 bits, which would be the ID size in real operation.

A node descriptor consists of 16 bytes: 8 bytes for the node's 64-bit ID, 4 bytes for its IP address, 2 bytes for the port, and 2 bytes for the descriptor's timestamp. The view maintained by each agent of a node has $\ell = 20$ descriptors, which account for 320 bytes. A view exchange involves sending the view to a peer *and* receiving the peer's view, causing traffic of 640 bytes in total. Every ΔT, each agent of every node initiates exactly one view exchange, and also participates on average in one view exchange initiated elsewhere. Therefore, two view exchanges per agent cause traffic of 1280 bytes. For all four agents, a single node exchanges $4 \times 1280 = 5120$ bytes every $\Delta T = 10sec$. That is, 512 bytes per second, or 4096bps (4Kbps). This is the price to pay for achieving fully operative routing tables in less than $30 \times 10 = 300$ seconds, which is 5 minutes.

For the aggregate bandwidth we multiply the individual node bandwidth by the number of nodes and divide by two, since the traffic caused by each view exchange has been counted twice, once for the gossip initiator and once for the gossip recipient. Therefore, we have a total bandwidth of $65,536 \times 4/2 = 131,072$Kbps, which is 128Mbps. Note that even though this bandwidth seems too high, it is in fact distributed across the whole (possibly world-wide) network.

In a real system, with 64-bit node IDs, and a digit length of 4 bits, we would need 16 agents running per node. This would require the exchange of $16 \times 1280 =$

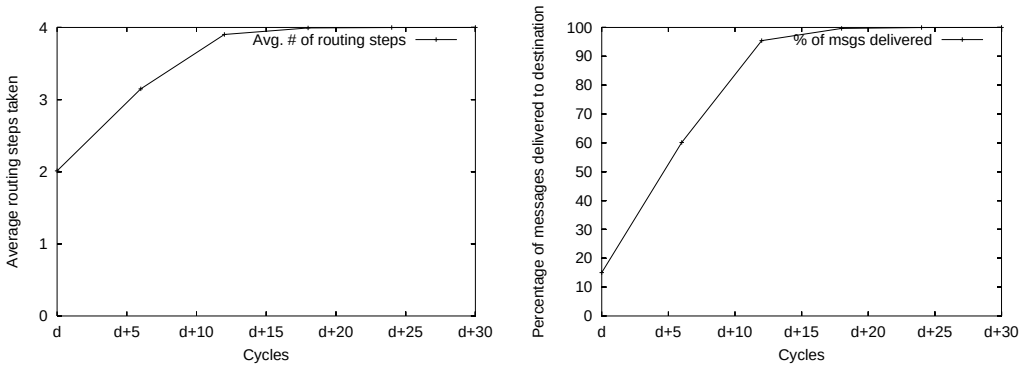


Figure 5.5: Message routing while recovering from a 50% node crash that happened at cycle d. Left: Average routing steps taken. Right: Percentage of messages delivered.

$20,480$ bytes every ΔT per node. Note that the refresh interval, ΔT, is a configuration parameter. By setting a longer refresh interval, we can lower the bandwidth used by each node, at the expense of slower completion of the routing tables. For instance, a refresh interval of $\Delta T = 60sec$ would require a bandwidth of $20,480/60 \simeq 341$ bytes per second, or roughly 2.7Kbps. However, in that case routing tables would take longer to be filled, around 30 minutes.

5.5. CONCLUSIONS AND RELATED WORK

In this chapter we demonstrated the potential of the PEER SAMPLING SERVICE in forming a structured overlay. In particular, we focused on managing routing tables for DHT-based peer-to-peer networks. We introduced a multi-layer architecture consisting of multiple instances of the PEER SAMPLING SERVICE, and investigated the system's behavior through experimentation. We showed that the proposed system forms routing tables fast, in a totally decentralized, self-organized manner. We also showed that it demonstrates self-healing behavior in the face of large-scale network disasters.

A more recent version of Pastry [Castro et al. 2003] has adopted gossiping for lightweight selection of neighbors of closer proximity. Nodes periodically gossip with neighbors of every row. It resembles our protocol in the sense that neighbors of row r are contacted in search of better neighbors of the respective row. However, nodes gossip a node's neighbors with respect

It resembles our protocol in the sense that nodes gossip with their neighbors

of a

In [Jelasity and Babaoglu 2005], Jelasity and Babaoglu suggest an alternative method to build a Pastry-like DHT based on NEWSCAST and T-MAN, a protocol very similar to VICINITY. They assign each node an ID, and define the distance between two nodes as the number of different bits in the binary representations of the two IDs. The T-MAN *ranking function* (similar to the selection function in VICINITY) gives higher preference to nodes of shorter distance. Eventually, nodes get to establish links to all other nodes whose IDs differ only in one bit to their own ID. This topology is quite similar to the one defined by Pastry's routing tables. Their solution appears to be more efficient than the one proposed in this chapter.

Montresor et al. apply T-MAN and NEWSCAST in [Montresor et al. 2005] to bootstrap and maintain the Chord DHT. In Chord, nodes are assigned IDs in a cyclic ID space, and each node maintains links to nodes of successive IDs, at steps 2^i, $i = 0, 1, 2, \ldots$ Montresor et al. define a T-MAN ranking function that sorts nodes in a ring structure (based on numeric ID distance). As nodes discover neighbors of increasingly closer ID distance, they remember the ones at numeric distance closer to 2^i, forming an overlay that resembles Chord. Although their method is efficient, the whole process needs to be regularly restarted at all nodes, to refresh nodes' routing tables and accommodate for changes in the network.

A more sophisticated solution would include a number of optimizations, such as giving higher preference to neighbors of high bandwidth or low latency, dynamically adjusting the gossiping frequency at each layer, aggregating gossip messages to save bandwidth, etc. However, such optimizations are left as future work. We emphasize that the goal of the research presented in this chapter was to prove the concept of using an unstructured overlay to bootstrap and maintain a structured one, in a simple, self-organizing manner, exhibiting self-healing properties.

The results of the experiments suggest that our system can provide highly robust, non-centralized routing table management. However, given the exploratory nature of this research, no special concern has been given to bandwidth consumption. Our architecture could possibly use significantly less bandwidth if adaptive refresh intervals were applied. Also, each agent of a node could have an individually optimized set of configuration parameters, such as view size and gossiping period. However, future research in this field should exploit VICINITY's power in building structured overlays.

CHAPTER 6

Information Dissemination

Reliable group communication forms an important class of operations in distributed systems. It is crucial to a number of applications, including event notification systems [Eugster et al. 2003a], distributed database replication [Demers et al. 1987], and distributed failure detection [van Renesse et al. 1998]. In this chapter we are particularly interested in *information dissemination*, that is, the broadcasting of messages to a set of nodes.

6.1. BACKGROUND AND RELATED WORK

One class of solutions for information dissemination is specifically designed to take advantage of the physical properties of certain types of networks. For example, a number of solutions are built to disseminate messages over Ethernet networks [Babaoglu 1987; Clegg and Marzullo 1997; Cristian 1990; Abdelzaher et al. 1996]. Such protocols provide high efficiency and strong reliability. However, their use is limited to the types of networks they are designed for, and, therefore, they are not suitable for large-scale, wide-area networks.

Another class of solutions, suitable for wide-area networks, consists of IP Multicast protocols. These protocols are not reliable. They rely on functionality embedded in routers, that enables the dynamic construction of a dissemination tree (or generally graph) reaching all participating nodes. A number of solutions have been proposed on top of IP Multicast, such as SRM [Floyd et al. 1997] and RMTP [Lin and Paul 1996], to improve its reliability. Nevertheless, IP Multicast is not widely deployed in the Internet.

Application-layer multicast forms a third class of solutions that has emerged in the recent years. The main advantage of these solutions is that they are very generic, and, therefore, they can be directly deployed over today's network infras-

tructure. There exist application-layer multicast protocols that provide reliability guarantees [Hadzilacos and Toueg 1993]. However, many of them do not scale well to a large number of nodes [Piantoni and Stancescu 1997].

A class of application-layer multicast has recently emerged [El-Ansary et al. 2003; Castro et al. 2002; Zhuang et al. 2001], based on the structure of DHTs such as Chord [Stoica et al. 2003], Pastry [Rowstron and Druschel 2001a], and Tapestry [Zhao et al. 2004, 2001]. What is common in these DHTs is that, in their respective overlays, each node is the root of a tree spanning the whole network. These spanning trees are used for message dissemination. Although systems of this class are nearly optimal with respect to message overhead, a single failure along a spanning tree can result in a whole branch missing a message. Failures are disregarded as a whole in [El-Ansary et al. 2003], where the assumption of reliable communication is made. Scribe [Castro et al. 2002] provides by default best-effort delivery. Reliability is improved to some extent by imposing TCP connections among nodes, a rather heavy assumption for dynamic, large-scale P2P networks. Finally, Bayeux [Zhuang et al. 2001], a system mainly targeted at data streaming, improves on reliability by redundantly disseminating messages across different paths of a spanning tree. However, its design is exposed to scalability problems, as each request to join a group is routed to a single node managing that group.

Another class of protocols targeted at large-scale dissemination of data has recently gained enormous popularity. BitTorrent [Cohen 2003] is the most popular example, but numerous other protocols have been suggested, such as Tribler [J.Pouwelse et al. 2006], Slurpie [Sherwood et al. 2004], ChunkCast [Chun et al. 2006], ChunkySpread [Venkataraman et al. 2006], and CoopNet [Padmanabhan and Sripanidkulchai 2002]. These systems are designed for the dissemination of large files to a set of nodes. More specifically, they aim at reducing the download time for large files, and at the same time off-loading the servers serving these files. The principal idea behind these protocols is splitting a large file in many small parts, distributing different parts to various peers, and then letting those peers talk to each other to retrieve the parts they are missing. Such mechanisms can be significantly effective in enhancing the download speed of large files, while reducing the load of the server offering a file, notably when a large set of downloading peers is involved. However, they are not suitable for the dissemination of small messages, which we deal with in this chapter.

Gossip-based protocols, such as Bimodal Multicast (*pbcast*) [Birman et al. 1999] and Directional Gossip [Lin and Marzullo 1999] form an alternative to strongly reliable broadcasting approaches. Each node forwards a message to a small random subset of the network, and so on. These protocols generally provide only *probabilistic* guarantees for message delivery. However, they are attractive because they are easy to deploy and resilient to node and link failures, due to re-

dundant message deliveries. On the other hand, scalability can suffer if nodes are required to maintain full knowledge of the network, notably when node churn is at stake. Optimizations have been suggested in [Birman et al. 1999] to overcome such scalability issues.

Other gossiping protocols, such as *lpbcast* [Eugster et al. 2001, 2003b] and [Kermarrec et al. 2003; Gupta et al. 2006] provision for membership management too. In particular, [Gupta et al. 2006] describes a hybrid dissemination system, that multicasts messages using a tree-based hierarchical structure, and locally switches to gossiping when a large number of failures is detected. These protocols drop the assumption of full knowledge of the network. Each node maintains a small view of the network, consisting of a few links to neighbors, which are used for dissemination. This makes them highly scalable. However, due to their probabilistic nature, a message may fail to reach the whole network even in a fail-free environment. To alleviate this, highly redundant message forwarding is employed.

In this chapter we present a gossip-based protocol that is deterministic in fail-free networks. When failures occur, its reliability degrades gracefully with the number of failures. Our protocol is based on the VICINITY/CYCLON framework, presented in Chapter 4.

6.2. EVALUATING A DISSEMINATION SYSTEM

A number of issues are of concern when evaluating or comparing information dissemination systems. It is essential for the rest of this chapter to list the metrics used to evaluate the effectiveness and usefulness of a dissemination system.

Hit ratio This is defined as the ratio of nodes that receive a message over the total number of nodes for which that message is intended. It is a metric of a dissemination system's reliability. Ideally, a reliable dissemination system should always achieve a hit ratio 100%. In our evaluation (Section 6.6), we present graphs of the complementary *miss ratio* metric, defined as follows:

$$MissRatio = 1 - HitRatio$$

Resilience to failures and churn For a dissemination system to be meaningful in a real-world dynamic network, it should operate reasonably well in the presence of node or link failures, and node churn. The operation under such conditions is evaluated by means of the hit ratio, described above.

Dissemination speed The time required for the dissemination of a particular message to complete. Obviously, the faster a message is disseminated the better. Dissemination speed depends on two main factors. First, the delay

in forwarding messages (processing delay on nodes plus network latency). Second, the number of hops a message takes to reach the last node. In our evaluation we focus on the latter factor.

Message overhead The overall number of times a message is forwarded during its dissemination. For a message to reach N recipients, it has to be forwarded a minimum of N times. In practice, however, message overhead is often multiple times higher. Message overhead is a metric of a dissemination system's behavior with respect to preserving or wasting network resources.

Load distribution The distribution of load over nodes, in terms of messages received and messages forwarded. Ideally, load should be evenly distributed among participating nodes.

In this chapter we are interested in reliable dissemination of messages originating at *any* node to *all* participating nodes. We do not focus on optimizing the dissemination of messages with respect to any proximity metric or by building a spanning tree. Also, we do not consider positive or negative acknowledgements, or requests for retransmission of lost messages. Instead, we introduce redundancy in message dissemination and examine its relation to the level of reliability achieved. We investigate the power of epidemics at disseminating messages to all nodes, with a high probability.

6.3. DETERMINISTIC DISSEMINATION

Consider a system consisting of N nodes, and a set of directed links among them. A *message* can originate at any of the participating nodes, and aims at reaching the whole network. A node that generates a new message or receives a message for the first time, forwards it across *all* its outgoing links. If a node receives a message for the second time, it simply ignores it. As an optimization, a message is never forwarded back to the node it was just received from. This basic algorithm is often referred to as *flooding*. Figure 6.1(a) shows the pseudocode for the dissemination algorithm.

The distinguishing characteristic of flooding is that one can deterministically control dissemination by imposing the appropriate overlay on the nodes. The underlying requirement to guarantee complete dissemination starting from any participating node, is to form a *strongly connected directed graph*[1] including *all* nodes. A multitude of overlays have been proposed for information dissemination

[1] a directed graph in which there is a directed path between any ordered pair of nodes

```
when node P generates message m,
        or receives m from node Q do
    if m not already seen then
        targets ← selectGossipTargets(Q)
        foreach T ∈ targets do send(T, m)
    endif
end
```

(a)

```
function selectGossipTargets(Q)
    targets ← view-{Q}
    return targets
end
```

(b)

Figure 6.1: (a) The generic dissemination algorithm. (b) Gossip target selection for deterministic dissemination (flooding).

by means of flooding, each one demonstrating a different behavior with respect to the metrics listed in the previous section.

Spanning trees or simply *trees* were among the first types of overlays proposed for flooding. Their strong point is that they are optimal with respect to the number of links maintained and, consequently, to the message overhead associated with dissemination. Indeed, in a network consisting of N nodes, the complete dissemination of a message over a tree involves exactly $N-1$ point-to-point communications. Their main disadvantage, though, is that a single failure of any link or any non-leaf node disconnects the tree, prohibiting messages from reaching all nodes. Also, maintaining a valid tree structure, ensuring the graph is connected and yet acyclic, is not a trivial task in the presence of failures. For these reasons, trees are not suitable for dynamic environments where failures can happen.

A special type of tree-based overlays for flooding is the *server-based* class (*star graphs*), where all nodes are connected by bidirectional links to a single node acting as a relay server. In these overlays all but the server node are leaf nodes, therefore their failure has no effect on the remaining nodes, but the server becomes a single point of failure. In addition, such overlays demonstrate the worst possible load distribution, the server node being linearly loaded by the number of nodes and number of messages being disseminated, rendering it a non-scalable solution.

On the other end of the spectrum lie *cliques* (*complete graphs*). In such a setting, every node has a complete view of the network. A node broadcasts a message by sending it to every other node in the network. This provides maximum reliability, at the cost of high maintenance costs. Although messages always reach all nodes irrespectively of how many nodes have failed, maintaining this type of overlay is impractical. Maintaining a fully connected graph is expensive in networks larger than a few dozen nodes, notably when the membership changes continuously.

A class of flooding overlays deserving more attention is the one based on

Harary graphs, introduced by Harary in [Harary 1962], further studied by Jenkins and Demers [Jenkins and Demers 2001], and applied by Lin et al. [Lin et al. 2000] in flooding. A Harary graph of connectivity t is a minimal link graph that is guaranteed to remain connected when up to $t-1$ nodes or links fail. Its minimum cut, therefore, consists of t links. Moreover, in a Harary graph links are evenly distributed across nodes, each node having either t or $t+1$ bidirectional links. An example Harary graph of connectivity 2 is a bidirectional ring, that we will use later in Section 6.5.1. Such overlays are very appealing for information dissemination in the presence of failures, as they are guaranteed to sustain up to a certain number of failures while imposing the minimum message overhead (for the corresponding reliability guarantees), and this overhead is evenly balanced across all nodes. The maintenance of such graphs, notably of higher connectivity t, can be a complicated and expensive task for large-scale, dynamically changing networks.

6.4. PROBABILISTIC DISSEMINATION

Acquiring reliability by imposing systematic structure on overlays is infeasible in dynamic networks of massive scale. In this section we take a look at an appealing alternative, *probabilistic dissemination* algorithms, which trade-in deterministic reliability guarantees in return of overlay construction and maintenance simplicity.

In these algorithms, dissemination is not guaranteed by means of a strategic topology, but by increased redundancy in message forwarding. The basic idea is that a node receiving a message forwards it to a number of *random* other nodes. It turns out that if that number is sufficiently high, messages reach all nodes with a high probability [Kermarrec et al. 2003]. The choice of random nodes to forward messages to can be easily handled by a peer sampling protocol as described in Chapter 3. The main advantage of probabilistic dissemination algorithms is that they are very simple to implement and inherently tolerant to dynamic environments, at the cost of an increased message overhead.

6.4.1. The RANDCAST Dissemination Algorithm

We consider a system consisting of N nodes. Each node runs the PEER SAMPLING SERVICE, providing it with a small, random, partial view of the network. A *message* can originate at any of the participating nodes, and aims at reaching the whole network. A node that generates a new message or receives a message for the first time, forwards it to (up to) F nodes, called the node's *gossip targets*, chosen randomly from its membership protocol's view. F is a system-wide parameter, called the *fanout*. A message is never forwarded back to the node it was

```
function selectGossipTargets(Q)
    targets ← F random nodes from view-{Q}
    return targets
end
```

Figure 6.2: Gossip target selection for the RANDCAST dissemination algorithm.

just received from. Figure 6.2 shows the pseudocode for the selection of gossip targets in the RANDCAST dissemination algorithm.

Note that this algorithm is quite efficient at spreading a message to a considerable percentage of the nodes in the network very fast, specifically at exponential speed with base F. Indeed, a new message progressively reaches F^0 (=1, the message generator), F^1, F^2, ... other nodes. Consequently, a message spreads very fast even for small values of $F \geq 2$. Of course, dissemination slows down when the message is forwarded to nodes that have already received it. However, in an overlay with highly randomized links (low clustering), such as the one formed by CYCLON, this slowdown turns out to be negligible until the message reaches a substantial percentage of the network.

Despite its strength at spreading messages fast, RANDCAST is not as efficient at achieving *complete dissemination*, that is, to reach every single node in the network. It is by nature a probabilistic algorithm. Even in the absence of failures, it provides no hard guarantees that a message will reach *all* nodes. It is not hard to see why. By forwarding messages at random, a node has no guarantees that at least one of its incoming links will be chosen to forward the disseminated message. To alleviate this, abundant redundance should be introduced by means of a large fanout. However, this is not desirable, because message overhead increases proportionally to the fanout, as we will see in the evaluation in Section 6.6.

The RANDCAST dissemination algorithm has been analyzed and evaluated by Kermarrec et al in [Kermarrec et al. 2003]. In that paper, a node's view is chosen uniformly at random among the whole network. In the version of RANDCAST studied in this dissertation, nodes' views are managed by CYCLON.

In the following section we introduce a novel, hybrid dissemination algorithm, combining deterministic and probabilistic dissemination. We defer the evaluation of both protocols until Section 6.6, where they are compared side by side.

6.5. HYBRID DISSEMINATION

As we discussed above, although probabilistic protocols are good at spreading messages fast even for small values of F, a large value of F is mandated to reach every single node in the network. This inefficiency can be tackled by introducing some *determinism* in the selection of gossip targets, ensuring any possible dissemination graph is connected and includes all nodes.

Hybrid dissemination protocols aim at combining probabilistic and deterministic behavior. To that end, they establish two types of links among nodes. Random links (*r-links*) contribute to their probabilistic behavior, and deterministic links (*d-links*) bring in determinism. R-links are simply links randomly selected, just like in purely probabilistic dissemination protocols. When presented with a message, a node forwards it across a few r-links. Consequently, messages initially spread to a large portion of the network at close to exponential speed.

However, a disseminated message should reach every single node in the network. That is, it should have been forwarded across at least one incoming link of each node. The basic idea is to establish a set of d-links, and have nodes deterministically forward messages across *all* their outgoing d-links, in addition to a few of their outgoing r-links. If the set of d-links forms an overlay compliant to the deterministic dissemination protocols' requirement, that is, it forms a strongly connected directed graph including all nodes, complete dissemination of messages is guaranteed. In such a graph, each node's indegree is at least 1. Moreover, if we ensure that each node has at least t incoming d-links, then complete dissemination is guaranteed even in the presence of up to $t-1$ faulty nodes.

Hybrid protocols effectively decouple the two fundamental goals in information dissemination. On one hand, spreading a message to a large percentage of the nodes fast, and on the other, reaching every single node. The probabilistic component carries out the bulk of the dissemination task, while the deterministic one takes care of the fine-grained details.

What makes hybrid dissemination protocols attractive, is that the set of d-links does not need to form a particularly sophisticated and hard-to-maintain structure. The sole requirement is that the set of d-links forms a strongly connected directed graph over all nodes. A simple structure satisfying this requirement is a ring. In the following section we explore how it can be used as a basis for a practical hybrid dissemination system.

6.5.1. The RINGCAST Dissemination Algorithm

We introduce RINGCAST, a novel *hybrid* dissemination algorithm that—even with a very low fanout—guarantees complete dissemination in a failure-free environment. In the presence of failures, its performance degrades gracefully, nev-

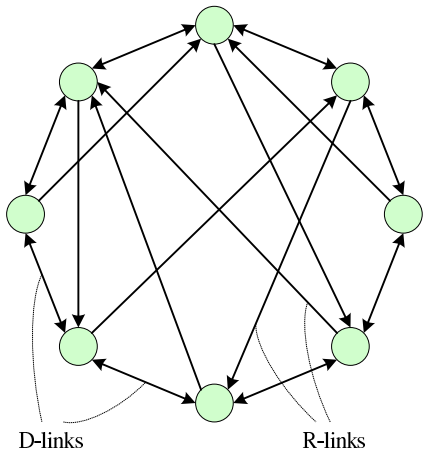


Figure 6.3: Example of a RINGCAST overlay. Nodes are organized in a bidirectional ring (by means of the *d-links*), and each one has a number (in this case only one) outgoing random links (*r-links*).

ertheless still outperforming RANDCAST. Finally, when confronted with continuous churn, RINGCAST proves again more reliable than RANDCAST, excluding nodes that joined the system very recently (for which it performs worse).

Operation

As discussed above, hybrid dissemination algorithms maintain two types of links between nodes, namely r-links and d-links. R-links are random links, in the case of RINGCAST formed by CYCLON, just like in RANDCAST. With respect to d-links, RINGCAST organizes nodes in a *global bidirectional ring* structure. A bidirectional ring constitutes a strongly connected graph, as required by deterministic dissemination protocols. Figure 6.3 illustrates an example RINGCAST overlay, where nodes form a bidirectional ring, and each one has a single outgoing r-link.

A unidirectional ring would be appropriate as well, as it forms a strongly connected graph too, though less reliable than a bidirectional one in the face of failures. However, the crucial reason for deciding on a bidirectional ring lies in the ring construction, and will be discussed in the following section.

Just like in the dissemination protocols discussed earlier, a node that generates a new message or receives a message for the first time, forwards it to (up to) F nodes, where F is the system-wide fanout parameter. However, in the case of RINGCAST, a node always forwards a message to its two ring neighbors (sending it across its two outgoing d-links), and across $F-2$ randomly selected r-links.

```
function selectGossipTargets(Q)
    targets ← {}
    if ringNeighbor1 ≠ Q then targets ← targets + {ringNeighbor1}
    if ringNeighbor2 ≠ Q then targets ← targets + {ringNeighbor2}
    targets ← targets + (F − targets.size) random nodes from (view − {Q})
    return targets
end
```

Figure 6.4: Gossip target selection for the RINGCAST dissemination algorithm.

If the message was received through one of the node's ring neighbors, the node forwards it to the other ring neighbor, and across $F-1$ random r-links. Figure 6.4 shows the pseudocode for the selection of gossip targets in the RINGCAST dissemination algorithm.

Note that the minimal cut in a bidirectional ring is two. That is, although no single node failure can break the ring in two disjoined partitions, prohibiting complete dissemination to the remaining nodes, such a situation *will* occur if two non-adjacent nodes fail. In most cases, however, this is not a crucial problem for dissemination, as d-links are only one facet of the process. R-links can carry the message to arbitrary nodes, most often bridging the gap between two or more disjoined ring partitions. Effectively, it suffices if any *one* node of an isolated ring partition receives the message, as the message will propagate to the whole partition over the d-links. Figure 6.5 presents a complete dissemination scenario over a ring split in several partitions. As we will see in the evaluation in Section 6.6, RINGCAST achieves a high hit ratio (higher comparatively to RANDCAST) even in the presence of many failed nodes.

Topology Construction

We employ the VICINITY/CYCLON topology construction framework for building and maintaining RINGCAST overlays. By applying the appropriate VICINITY selection function, we organize nodes in a bidirectional ring structure. The links of this ring constitute the d-links. R-links are provided directly by CYCLON.

Letting nodes self-organize in a ring structure involves a global agreement on a consistent ordering of nodes. Along these lines, each node selects an arbitrary *sequence ID* (simply *ID*), out of a large ID space (e.g., 128 bits). Each node forms links to the two nodes with closest IDs, one in each direction: the node with just higher, and the node with just lower sequence ID, using modulo arithmetic. These links define the bidirectional ring.

The VICINITY selection function, $S(k,P,\mathcal{D})$, used to form this topology is

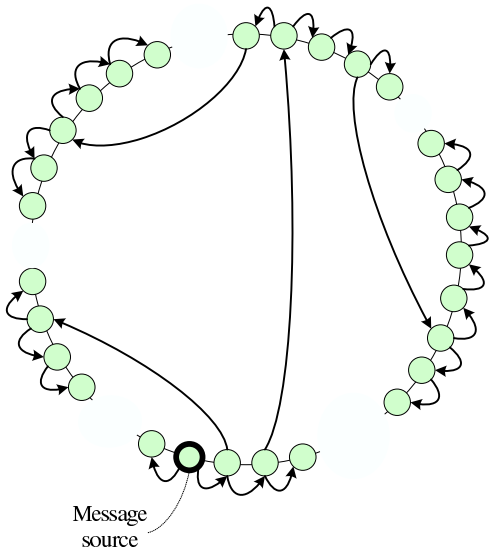


Figure 6.5: Example of a message dissemination in a partitioned ring. For clarity, only a few of the followed r-links are shown.

simple and straightforward. First, it sorts peers in $\mathcal{D}$ based on their sequence IDs. Then, it selects the k ones with IDs closest to the ID of P, $k/2$ in each direction. That is, it selects $k/2$ peers with just higher, and $k/2$ with just lower ID than the ID of P, using modulo arithmetic. The two peers with closest IDs—one in each direction—are used as ring neighbors.

The choice of a bidirectional—as opposed to unidirectional—ring, was not made only to improve dissemination reliability, but was also a crucial design choice for topology construction. Recall from Section 4.2.2 that VICINITY is more efficient if its selection function exhibits transitivity. That is, if peer P_2 is a "good" selection for peer P_1 ($P_1 \overset{S}{\dashrightarrow} P_2$), and P_3 is a "good" selection for P_2 ($P_2 \overset{S}{\dashrightarrow} P_3$), then, P_3 should have a good chance of being a "good" selection for P_1 too ($P_1 \overset{S}{\dashrightarrow} P_3$), preferably a "better" selection than P_2. This way, P_1 can gradually improve its links.

Obviously, this does not hold for selection functions that select nodes of close IDs in a *single* given direction (e.g., only with just higher IDs). In this case, a node's neighbor with higher ID would only have neighbors with even higher IDs, which would not help the original node improve its links. In contrast, if nodes were maintaining neighbors of close IDs in *both* directions, a node's neighbor with higher ID would also have some neighbors with lower (than his) ID, which

would be potentially "good" candidates for the original node.

6.6. EVALUATION

We evaluate the two protocols side-by-side in three scenarios. First, in a static and failure-free network. Second, in a static network right after a catastrophic failure, that is, after the sudden failure of a large number of nodes. Finally, in a dynamic network under continuous node churn. Evaluation was done with respect to the following criteria, as discussed in Section 6.2:

1. Hit ratio
2. Dissemination speed
3. Message overhead

We do not explicitly address load balancing, because both protocols are by nature distributing the load across all nodes evenly. A node receiving a message forwards it to F others, just like any other node.

Experiments were carried out using the PeerSim simulator [PeerSim]. We tested all scenarios by instantiating a network of 10,000 nodes. Each node was running CYCLON and, in the case of RINGCAST, VICINITY too, as described above, with view length 20 for each protocol. Nodes were initially supplied with a certain single contact in their CYCLON views, forming a star topology. VICINITY views were initially empty. After letting the network self-organize for 100 cycles, we started disseminating messages from various nodes picked at random.

We assume a very simple dissemination model, that allows us to study the evolution of disseminations in terms of discrete rounds, that we call *hops*. The generation of a message is marked hop 0. At hop 1, the message reaches F neighbors of the origin node. At hop 2, it further reaches the neighbors' neighbors, and so on. This way, we can evaluate the progress of a dissemination by counting the number of messages sent and the number of new nodes notified per hop.

An implicit assumption underlying our dissemination model is that the processing delay and network latency between all pairs of nodes are the same. Although latencies vary in a real wide-area network, our assumption does not have an effect on the macroscopic behavior of dissemination with respect to the hit ratio. Dissemination relies on nodes forwarding the messages they receive. A node that receives a message for the first time, forwards it to the same number of neighbors picked with the same logic, irrespectively of the time this happens. Consider for instance two scenarios of RANDCAST, executing over the same static overlay (assume gossiping is currently stalled), starting from the same origin and each node

picking the same gossip targets in both cases. If pair-wise latencies are different in the two scenarios, the order in which nodes are notified may change, but the exact same set of nodes will have been eventually notified. In the case of RINGCAST, the set of nodes notified may change, but the same macroscopic behavior is maintained.

6.6.1. Evaluation in a Static Failure-free Environment

We first evaluate and compare the two protocols side-by-side by considering a failure-free static environment.

We instantiated a network of 10,000 nodes in PeerSim. Each node was running CYCLON and, in the case of RINGCAST, VICINITY too as described above, with view length 20 for each protocol. Nodes were initially supplied with a given single contact in their CYCLON views, forming a star topology. VICINITY views were initially empty. After letting the network self-organize for 100 cycles, we started posting messages and observing their dissemination.

We ran a number of experiments—not presented here—to investigate the effect of gossiping speed on dissemination. More precisely, we explored the relation between the gossiping period and message forwarding time, that is, the time is takes a node to process a message and forward it to a neighbor. We varied the message forwarding time from zero to several times the gossiping period. We recorded no effect whatsoever on the macroscopic behavior of disseminations. That is, although changing the message forwarding time results in different experiments, with different nodes being reached each time and in a different order, all macroscopic properties, such as the hit ratio, dissemination speed, and message overhead, are preserved. It is not hard to see why. With respect to VICINITY-managed d-links, they are not even altered by gossip exchanges once the optimal sets have been obtained. With respect to CYCLON-managed r-links, these are random links anyway, irrespective of whether they are updated fast or are currently fixed. Consequently, forwarding a message along a few of them has an equivalent effect regardless of whether gossiping runs at a high rate or is currently stalled.

Having verified this, we chose to disseminate messages over *fixed* overlays in all experiments presented in this section. This choice was primarily made to limit simulation execution to a reasonable time, considering the large number of experiments we carried out. So, in each experiment, after self-organizing for 100 cycles the overlay was frozen and only then did disseminations start.

For each value of F ranging from 1 to 20 (the CYCLON view length), we posted 100 messages from various nodes picked at random, resulting in a total of 2000 experiments for each protocol. Since the hit ratio approaches 100% even for small values of F, it is more meaningful to present the miss ratio instead, in logarithmic scale. Figure 6.6(a) presents the dissemination miss ratio averaged

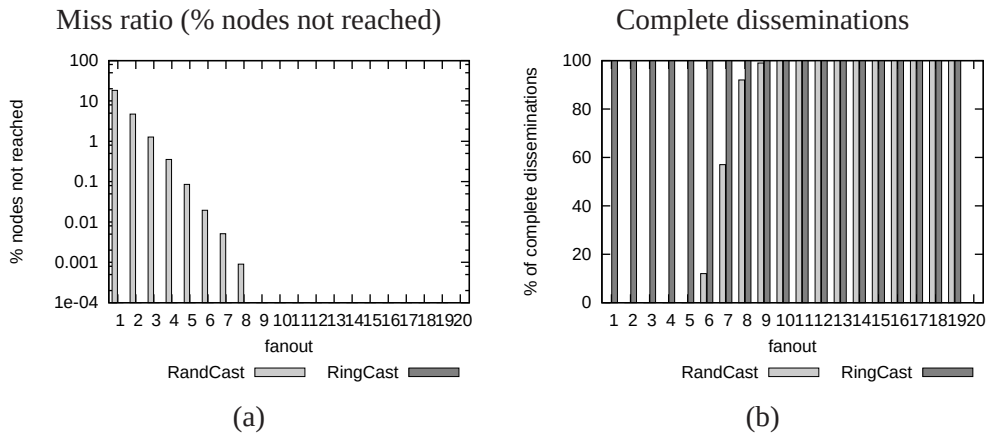


Figure 6.6: Dissemination effectiveness as a function of the fanout, for a failure-free static network of 10K nodes. (a) Miss ratio averaged over 100 experiments; (b) Percentage of 100 experiments that resulted in complete dissemination.

over 100 experiments for each value of F. RANDCAST and RINGCAST are represented by light and dark bars, respectively. The miss ratio for RANDCAST appears to be dropping exponentially as a function of the fanout F. Note that no dark bars appear in this graph, as the miss ratio for RINGCAST is zero for any choice of F. This comes as no surprise, as RINGCAST's operation guarantees complete dissemination in failure-free static networks.

Figure 6.6(b) shows the percentage of experiments that resulted in a complete dissemination, for each value of F. With respect to RANDCAST, it is interesting to see that the transit from 0% to 100% follows a rather steep curve. For instance, even with a fanout of 6, although the overall hit ratio was above 99.9% (Fig. 6.6(a)), none of the 100 experiments resulted in a complete dissemination. With a fanout of 8, more than half of the disseminations were complete, while by further increasing the fanout to 11 or higher we get only complete disseminations. As far as RINGCAST is concerned, this graph validates once again that disseminations are always complete, irrespectively of the chosen fanout.

Having seen to what extent messages eventually spread, we now take a closer look at the evolution of dissemination hop by hop. Figure 6.7 shows the progress of all 100 dissemination for each protocols, for four different fanouts. More specifically, it shows the number of nodes that have not yet been notified, as a function of the hops taken.

Four main observations can be made by examining these graphs. First, for a given fanout, all experiments of a protocol demonstrate very small variations

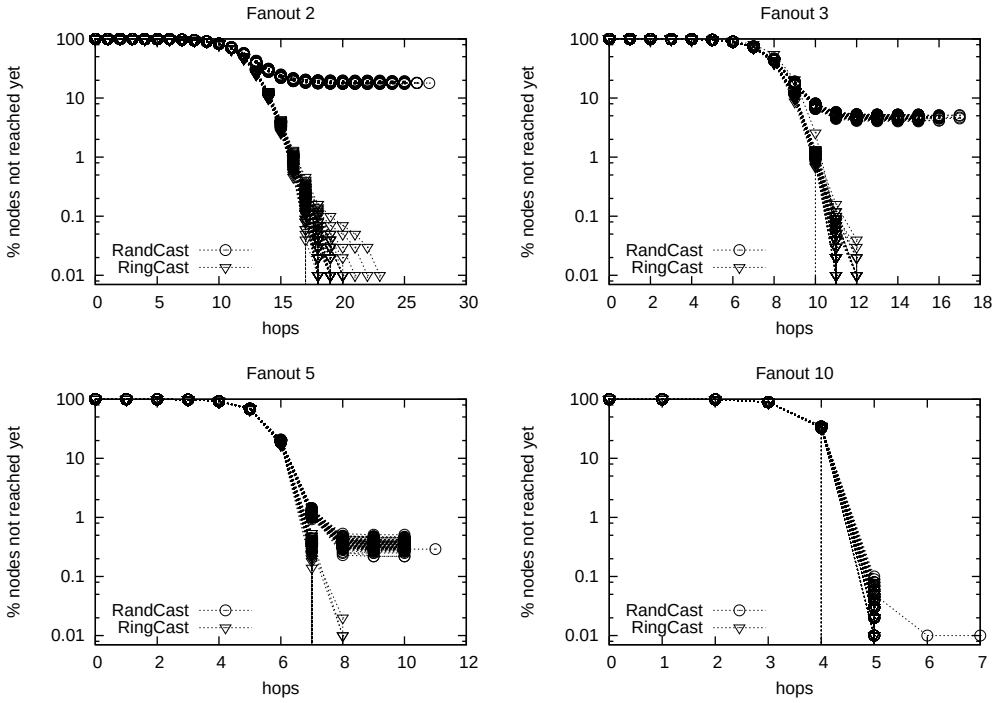


Figure 6.7: Dissemination progress in a static failure-free network of 10K nodes. 100 experiments of each protocol are shown.

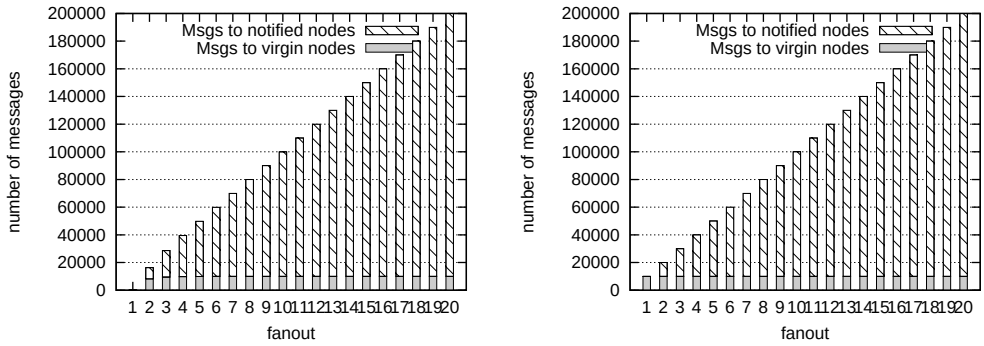


Figure 6.8: Total number of messages sent, divided in messages sent to not-yet-notified and already notified nodes.

in their progress with respect to the hit ratio and dissemination latency. This is important as it shows that by selecting the appropriate fanout value, we can tune a system's dissemination behavior to a good level of accuracy. Second, we notice a clear—expected—influence of the fanout on dissemination latency. The higher the fanout, the shorter a dissemination's duration. Third, we observe that the progress of disseminations for the two protocols is alike for a few initial hops, when the message has not yet reached a significant portion of the network. The protocols differentiate only after a substantial percentage of the nodes (i.e., at least 80%-90%) have been notified. This is a direct effect of the two protocols' operation. By forwarding messages at random, RANDCAST hardly reaches any more non-notified nodes, in an already saturated network. On the contrary, by also forwarding messages along the ring, RINGCAST exhaustively reaches out to every single node. Finally, we see that the higher the fanout the more similarly the two protocols disseminate messages. However, in all cases RINGCAST reaches the last node in fewer hops, demonstrating a lower dissemination latency.

The third metric we are interested in is message overhead. As we already mentioned in Section 6.4.1, message overhead increases proportionally to the fanout. Indeed, if a node forwards a newly received message to F other nodes and N_{hit} nodes are reached in a dissemination, the total number of messages sent is $F \times N_{hit}$. Figure 6.8 confirms this assessment. The shaded segments represent the number of messages reaching nodes for the first time (noted as "virgin" nodes). The striped segments represent the number of *redundant messages*, that is, messages reaching already notified nodes, and therefore constitute a waste of network resources. As the network consists of $10K$ nodes, for a given fanout F a complete

dissemination involves $F \times 10K$ total messages, out of which $10K$ are messages to "virgin" nodes, and the rest $(F-1) \times 10K$ are redundant. The two graphs are practically identical except for low fanouts, for which RANDCAST disseminations do not reach all nodes. These graphs are illustrative with respect to the reason the fanout should be kept as low as possible.

6.6.2. Evaluation after Catastrophic Failure

For a system to be usable in a realistic environment, it has to cope with failures. In this section we explore the behavior of the two protocols in the face of catastrophic failures, that is, when a number of nodes suddenly break down.

We set up the experiments like the ones in the previous section, but before starting the disseminations we kill a randomly chosen portion of the nodes. That is to say, for each experiment we simulate a network of 10,000 nodes, let it self-organize for 100 cycles, and stall gossiping. We subsequently remove a randomly chosen set of the nodes and examine dissemination over the remaining ones.

Unlike failure-free static networks where ongoing gossiping has no influence on dissemination after some point (see Section 6.6.1), in the face of failures gossiping *does* have an effect, namely a positive one. Following a catastrophic failure, gossiping allows the network reorganize itself, removing links to dead nodes and reestablishing valid ring links. In our experiments gossiping was *not* allowed following the catastrophic failure, exploring the ability of a partially damaged overlay to disseminate messages without giving it the chance to self-heal. This was our deliberate choice, aiming at testing a catastrophic failure's worst-case influence on dissemination.

Figure 6.9 presents the dissemination effectiveness for both protocols after catastrophic failures killing 1%, 2%, 5%, and 10% of the nodes. Similarly to Figure 6.6 in the previous section, the graphs on the left show the miss ratio, and the ones on the right the percentage of disseminations that reached all nodes, as a function of the fanout F. One can clearly see that RINGCAST is more effective at disseminating messages in all experiments. A closer look at these graphs shows that as the volume of the catastrophic failure grows larger, the difference between the two protocols' effectiveness decreases. However, even when 10% of the nodes are killed at once, RINGCAST demonstrates an order of magnitude lower miss ratio than RANDCAST. The lower miss ratio of RINGCAST reflects on the significantly higher percentage of complete disseminations for small fanouts.

Figure 6.10 shows the evolution of disseminations after a catastrophic failure of 5% of the nodes, in accordance to Figure 6.7 in the previous section. Once again, the relation between the chosen fanout and dissemination latency is verified. We also see that the evolution of disseminations exhibits small variations for a given configuration, like in the case of a failure-free static network.

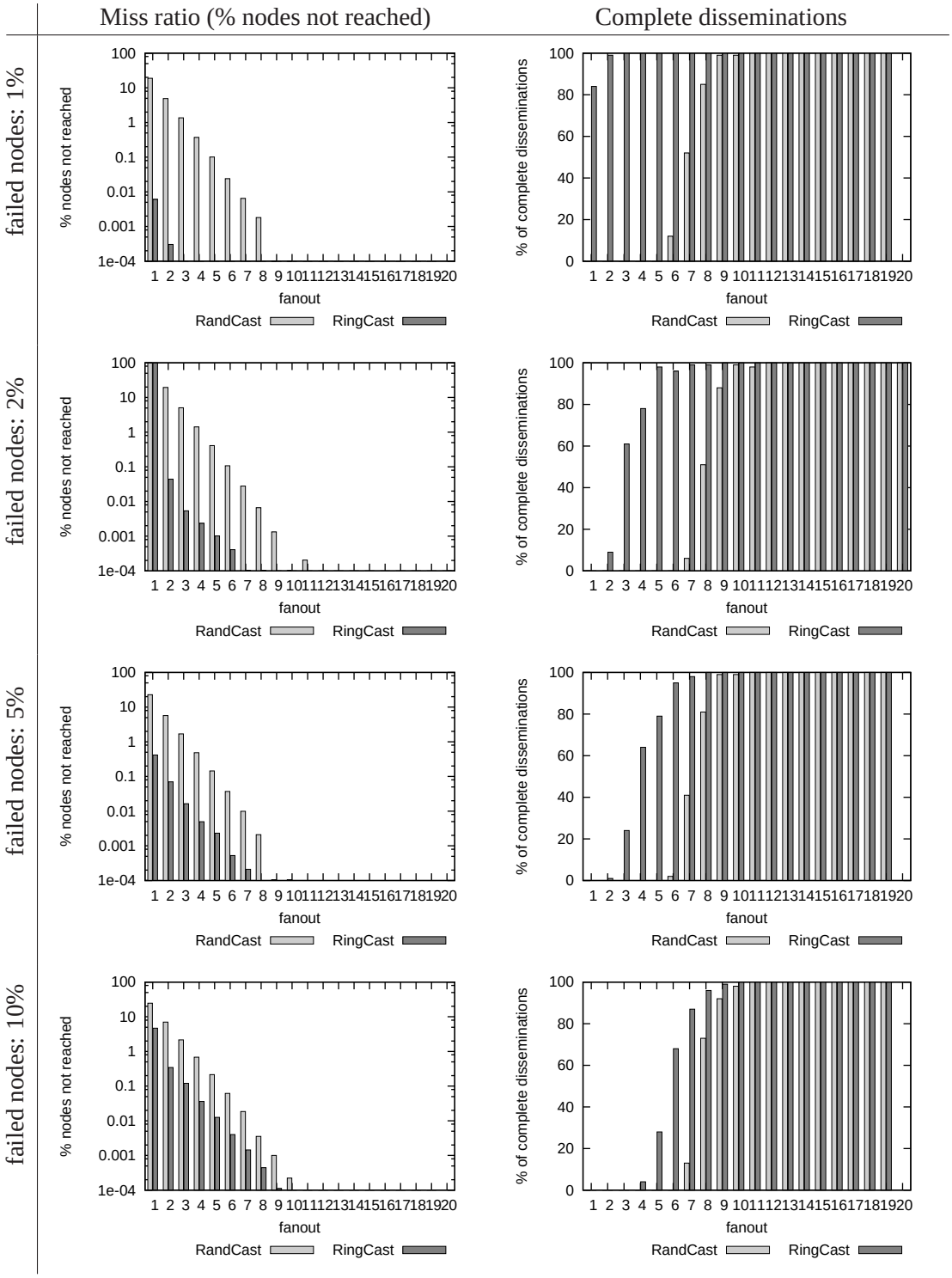


Figure 6.9: Dissemination effectiveness as a function of the fanout for static network of 10K nodes, after catastrophic failures of 1%, 2%, 5%, and 10% of the nodes.

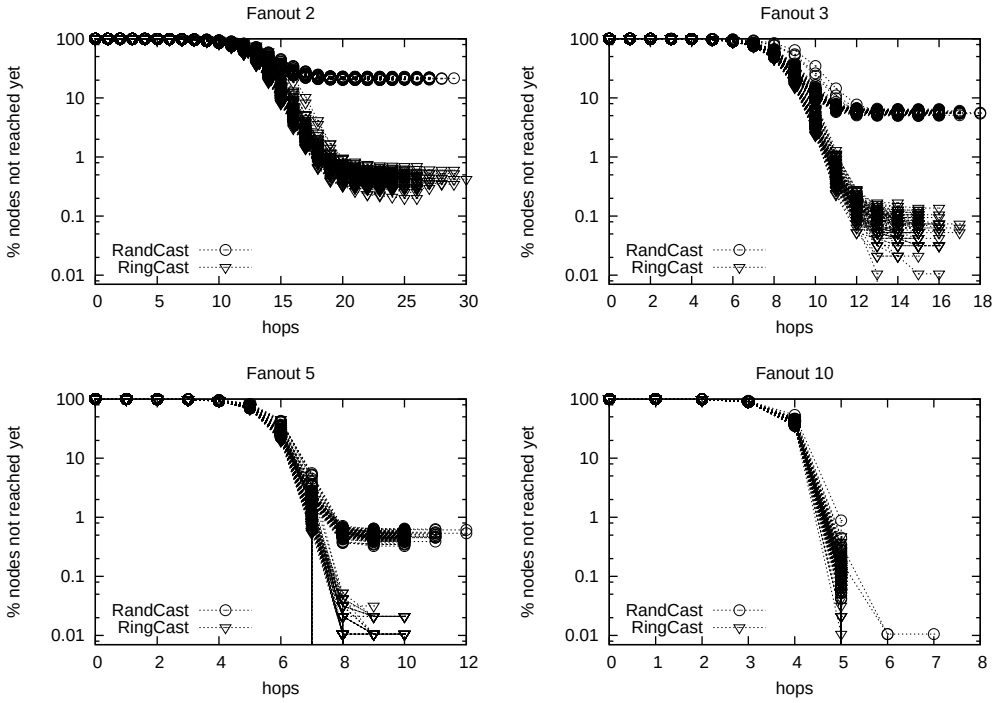
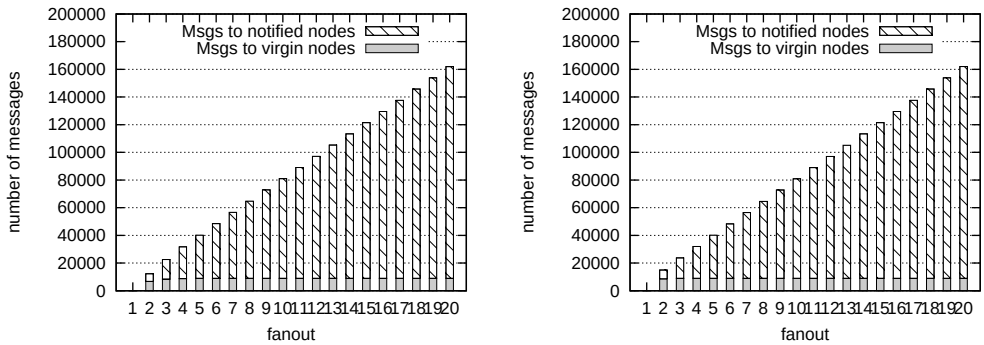


Figure 6.10: Dissemination progress in a static network of 10K nodes, after catastrophic failure killing 500 nodes (5%). 100 experiments of each protocol are shown.



RANDCAST RINGCAST

Figure 6.11: Total number of messages sent, divided in messages sent to not-yet-notified, already notified, and dead nodes.

Finally, Figure 6.11 illustrates the message overhead for dissemination in the presence of a catastrophic failure of 10% of the nodes. Again, the total number of messages is proportional to the chosen fanout. The lightly shaded segments show the number of messages reaching nodes for the first time. The striped ones represent messages reaching already notified nodes. Finally, the darkly shaded segments show the number of messages sent to dead nodes (and therefore lost). We see that the number of messages sent to dead nodes increases linearly for both protocols.

6.6.3. Evaluation under Churn

Apart from catastrophic failures, a system should also be able to deal with node churn, that is, continuous node arrivals and departures. In this section, we examine the behavior of the two protocols under churn.

We evaluate the two protocols against the artificial churn model introduced in Section 3.5.2. In that model, in each cycle a given percentage (known as the churn rate) of randomly selected nodes are removed, and the same number of new ones join the network. Recall that this constitutes a worst case churn scenario, as removed nodes never come back, so dead links never become valid again, and new nodes have to join from scratch. We tested both protocols with a churn rate of 0.2%, which, given a gossiping period of 10 seconds, corresponds to the churn rate observed in the Gnutella traces by Saroiu et al [Saroiu et al. 2003].

Unlike experiments on static networks where a small number of cycles sufficed to warm up the respective overlays (Sections 6.6.1 and 6.6.2), experiments on dynamic networks required significantly more warm-up cycles. A network of 10,000 nodes was let gossip in the presence of continuous artificial churn, until every node had been removed and reinserted at least once. For all experiments this took several thousand cycles. Then the respective network was frozen, and the resulted overlay was tested with respect to dissemination effectiveness.

Figure 6.12 shows the miss ratio and the percentage of complete disseminations as a function of the fanout. Although RINGCAST results in a lower miss ratio than RANDCAST for low fanouts (2 to 5), it performs slightly worse for fanouts 6 or higher. Also, none of the protocols achieves any complete disseminations, except when maximizing the fanout, in which case RANDCAST appears to be performing better again.

By looking at these quantitative graphs alone, one could come to the conclusion that RINGCAST is not any better—if not worse—than RANDCAST when node churn is at stake. A closer, qualitative examination of *which* groups of nodes contribute to each protocol's miss ratio will prove otherwise. As we will see, RINGCAST's miss ratio is almost entirely due to its poor performance at reaching newly joined nodes, while it provides good dissemination guarantees to all older

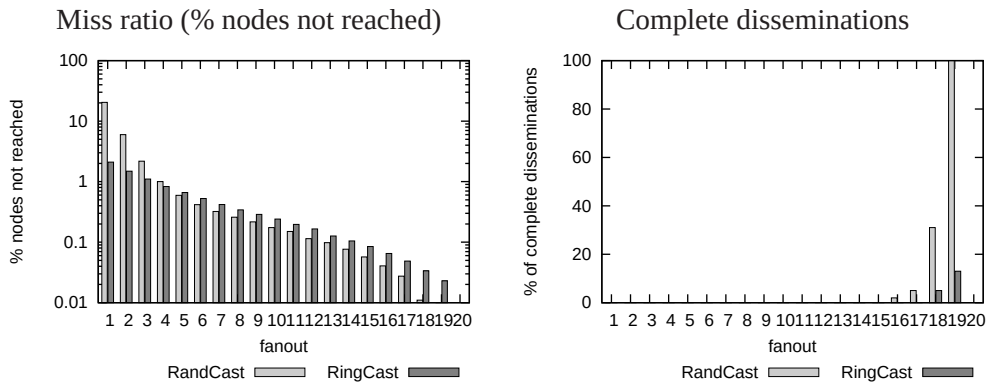


Figure 6.12: Dissemination effectiveness as a function of the fanout, in the presence of node churn. In each cycle, a randomly selected 0.2% of the nodes was removed, and replaced by an equal number of newly joined nodes.

nodes.

Along these lines, we now investigate the relation between a node's *lifetime*[2], that is, the number of cycles since it joined the network, and its chance of receiving a disseminated message. Figure 6.13 presents the distribution of node lifetimes after the execution of several thousand cycles, when every node has been removed and reinserted at least once. In fact, Figure 6.13 plots the exact count of nodes having a given lifetime, aggregated over 100 experiments, in log-log scale. Given that the network consists of 10,000 nodes and the churn rate is 0.2%, at each cycle 20 random nodes are evicted and 20 new are added. Therefore, the number of nodes having a given lifetime cannot exceed 20. For all 100 experiments together, the number of nodes of a given lifetime ranges from 0 to 2000, hence the range of the vertical axis.

The distribution of lifetimes of nodes that *were not notified* during dissemination, is presented in Figure 6.14. The distributions for two fanouts are shown, 3 (top) and 6 (bottom). It is clear that in all cases newly joined nodes (i.e., ones that joined up to 20 or 30 cycles ago) experience significantly higher miss ratio than other, older nodes. RINGCAST, in particular, results in quite more misses (notice the log scale) than RANDCAST for these nodes. Nevertheless, for nodes that have been in the network for at least 20 or 30 cycles, it demonstrates a substantially lower miss ratio, almost negligible compared to that of RANDCAST. For instance, let us take a look at dissemination with fanout 6. Although RINGCAST appears to have a higher overall miss ratio than RANDCAST (Fig. 6.12), it hardly suffers any

[2]not to be confused with a node's *age* field, as defined and used in CYCLON

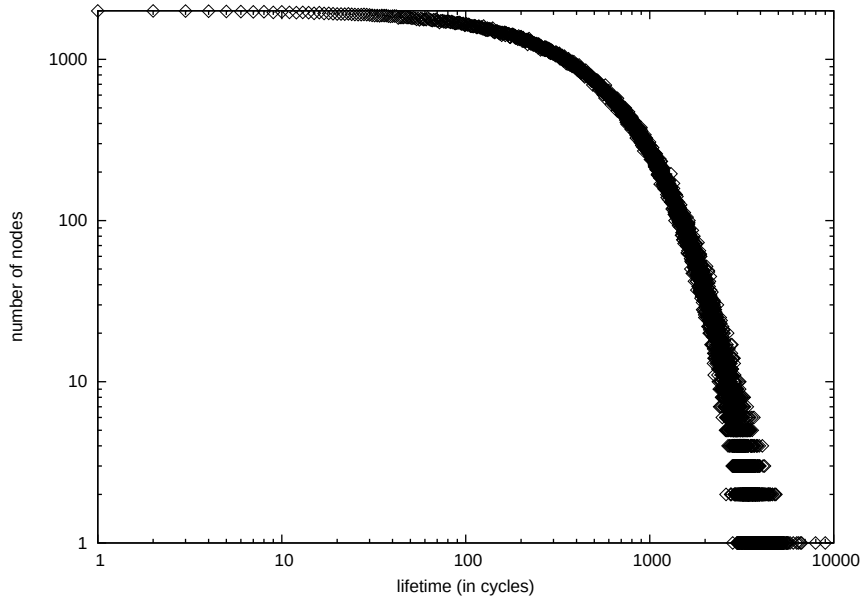


Figure 6.13: Distribution of node lifetimes, summed over 100 experiments.

misses for nodes that joined at least 20-30 cycles earlier, contrary to RANDCAST. Its miss ratio is entirely attributed to misses in newly joined nodes.

The implication behind this observation is worth noting. RINGCAST proves to be a better dissemination tool, except for the first few cycles after a node's join. Once a warm-up period of a few cycles has elapsed, a node receives all disseminated messages with very high probability. For a gossiping period of 10 seconds and a view length $\ell_{cyc} = 20$, the warm-up phase amounts to a bit over 3 minutes. In applications where faster node joins is vital, new nodes can gossip at an arbitrarily higher rate for the first few cycles, to complete their warm-up phase correspondingly fast. However, this is a mere optimization and will not be considered further in this dissertation.

At this point, it is interesting to understand why new nodes experience more misses, and why this phenomenon is more intense in RINGCAST. Nodes are notified through their incoming links. Their probability of being notified is tightly related to how well they are known by other nodes. A new node joins the network with zero indegree, and gradually increases it. Until a node's indegree reaches the average indegree of the network, it has less chance to receive a message than older, better connected nodes. This shows clearly in the aforementioned graphs (Fig. 6.14).

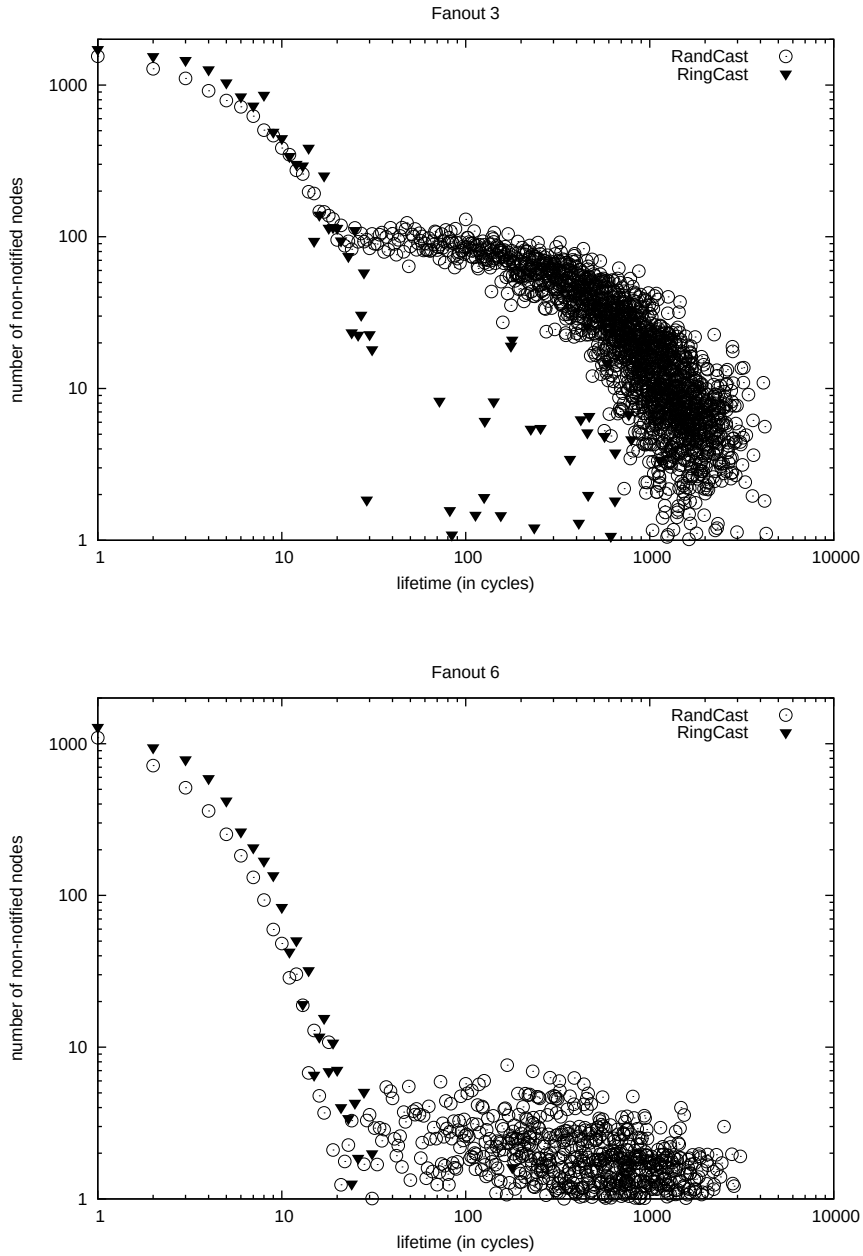


Figure 6.14: Distribution of lifetimes of nodes that were not notified, summed over 100 experiments.

More specifically, a new node's r-link indegree increases by one in each of its first few cycles, and takes approximately ℓ_{cyc} (here $\ell_{cyc} = 20$) cycles to stabilize to the average indegree of the network (which is ℓ_{cyc} too). This is a property of CYCLON, which manages r-links. So, for RANDCAST, which depends solely on CYCLON, we observe a steep decrease in misses for nodes of lifetimes 1 through 20, followed by an immediate stabilization thereafter.

On the other hand, RINGCAST also depends on VICINITY to form the d-links (i.e., the edges of the ring). However, a node does not benefit from incoming VICINITY links until the appropriate incoming d-links are formed, that is, until it eventually becomes known by its two direct ring neighbors. Generally this does not happen instantly, but may require an undefined—yet small—number of cycles. Until then, a newly joined node relies only on its incoming r-links to receive messages. During that phase, it is clear that newly joined nodes have better chances to receive messages in RANDCAST, where messages are forwarded to F r-links, as opposed to only $F-2$ r-links in RINGCAST. This explains why RINGCAST exhibits more misses than RANDCAST for nodes that joined roughly in the last 20 cycles (Fig. 6.14).

Note that the further curve in misses for lifetimes greater than 100 simply follows the lifetime distribution of the general node population (Fig. 6.13).

6.7. DISCUSSION AND FUTURE WORK

In this chapter we explored how gossiping can be applied to form overlays for information dissemination. Our goal was to introduce and demonstrate the benefits of hybrid dissemination systems, in comparison to probabilistic ones. However, our work can be extended in many directions.

Some applications may require higher reliability in dynamic environments. Recall from Section 6.3 that a bidirectional ring is a Harary graph of minimal cut two. One way to increase reliability, would be to design gossiping protocols that form Harary graphs of higher connectivity. Another, simpler way, is to organize nodes in multiple rings, assigning them a different random ID per ring. In both cases, reliability would be improved at the cost of increased gossip traffic.

Another potential optimization is proximity-based dissemination. Proximity can have many faces, e.g., geographic distance, domain name, network hops, etc. In the protocols examined in this chapter, proximity is not taken into consideration. For instance, a message originating in the Netherlands could follow a path such as Netherlands → Australia → Switzerland → Canada → Greece → Uruguay → New Zealand. Obviously, such a path is far from optimal.

A straightforward way to partially deal with domain name proximity in RING-

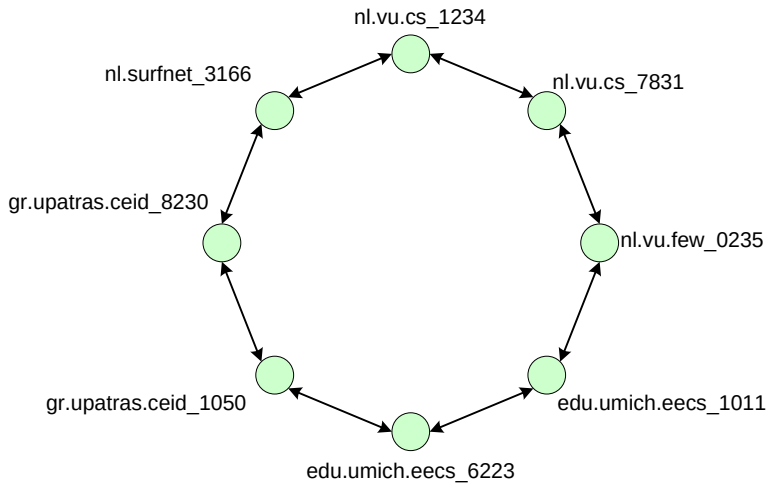


Figure 6.15: Nodes organized in a ring based on domain name proximity.

CAST, is to incorporate domain names in the VICINITY similarity function. In this version of RINGCAST, a node forms its ID by reversing its domain name (country domain first) and appending a randomly chosen number. I.e., the ID of a node at the `.cs.vu.nl` domain of the Vrije Universiteit in Amsterdam could be `nl.vu.cs.1234`. Without any additional modifications, nodes naturally organize themselves in a ring sorted by domain name, and domains sorted by country. An example can be seen in Figure 6.15.

Finally, it should be noted that the protocols discussed in this chapter are perfectly suitable for *application-layer multicasting* too. In multicasting, a number of *multicast groups* are defined, and each message is associated with one of them. All messages associated with a multicast group should be delivered to all nodes subscribed to that multicast group. The usage of dissemination protocols such as RANDCAST and RINGCAST for multicasting is straightforward. Each multicast group forms its own, separate dissemination overlay. Nodes subscribe by simply joining the overlay(s) of the multicast group(s) of their choice. Finally, messages are multicast by disseminating them in the appropriate dissemination overlay.

In this research we have not considered *pull-based* dissemination. We expect it to significantly improve the efficiency of the protocol in terms of reliability. However, additional issues have to be taken into account, such as the pull frequency, the duration for which nodes maintain old messages, the size of buffers on nodes, etc. Pull-based dissemination is left as future work, as it constitutes a natural extension of our current research.

CHAPTER 7

Semantic Overlay Networks

A lot of recent research on content-based P2P searching for file-sharing has focused on exploiting semantic relations between peers to facilitate searching. To the best of our knowledge, all methods proposed to date suggest *reactive* ways to seize peers' semantic relations. That is, they rely on the usage of the underlying search mechanism, and infer semantic relations based on the queries placed and the corresponding replies received.

In this chapter we introduce a *proactive* method to build semantic overlays, by means of the VICINITY/CYCLON topology construction framework presented in Chapter 4. By applying the appropriate selection function, peers with similar content are clustered together. It is worth noting that this peer clustering is done in a completely implicit way, that is, without requiring the user to specify his preferences or to characterize the content of files he shares.

As a historical note, the VICINITY/CYCLON topology construction framework presented in Chapter 4, was first conceived and materialized in the context of the research presented in this chapter.

7.1. OVERVIEW

File sharing peer-to-peer (P2P) systems have gained enormous popularity in recent years. This has stimulated significant research activity in the area of content-based searching. Sparkled by the legal adventures of Napster, and challenged to defeat the inherent limitations concerning the scalability and failure resilience of centralized systems, research has focused on *decentralized* solutions for content-based searching, which by now has resulted in a wealth of proposals for peer-to-peer networks.

In this chapter, we are interested in those groups of networks in which search-

ing is based on grouping semantically related nodes. In these networks, a node first queries its semantically close peers before resorting to search methods that span the entire network. In particular, we are interested in solutions where semantic relationships between nodes are captured implicitly. This capturing is generally achieved through analysis of query results, leading to the construction of a local *semantic list* at each peer, consisting of references to other, semantically close peers.

Only very recently has an extensive study been published on search methods in peer-to-peer networks, be they structured, unstructured, or of a hybrid form [Risson and Moors 2006]. This study reveals that virtually all peer-to-peer search methods in semantic overlay networks follow an integrated approach towards the construction of the semantic lists, while at the same time accounting for changes occurring in the whole set of nodes. These changes involve the joining and leaving of nodes, as well as changes in a node's preferences.

The construction of semantic lists should result in highly clustered overlay networks, reflecting users' interests. These networks excel for searching content when nothing changes. However, in realistic, dynamically changing networks, the discovery and propagation of changes that may happen *anywhere* in the network is of vital importance. For this reason, overlay networks should also reflect desirable properties of random graphs and complex networks in general [Albert and Barabási 2002; Newman 2002]. These two conflicting demands generally lead to complexity when integrating solutions into a single protocol.

Protocols for content-based searching in peer-to-peer networks should separate these concerns. In particular, we advocate that when it comes to constructing and using semantic lists, these lists should be optimized for search only, regardless of any other desirable property of the resulting overlay. Instead, a separate protocol should be used to handle network dynamics, and provide up-to-date information that will allow proper adjustments in the semantic lists (and thus leading to adjustments in the semantic overlay network itself).

Along these lines, we employ the VICINITY/CYCLON framework, which is designed to separate the two aforementioned issues. The VICINITY layer optimizes the semantic lists for searching only. The CYCLON layer, offers a fully decentralized service for delivering, in an unbiased fashion, information on new events. We demonstrate the efficiency of our framework through extensive simulations using traces collected from the eDonkey file-sharing network [Fessant et al. 2004].

7.2. MODEL OUTLINE

In our model each node maintains a dynamic list of semantic neighbors, called its *semantic list*, of small fixed size ℓ_{sem}. A node searches for a file in two phases. First, by querying its semantic neighbors. Second, and only if no results were returned from the first phase, the node resorts to the default search mechanism.

Our aim is to organize the semantic lists so as to maximize the hit ratio of the first phase of the search. We will call this the *semantic hit ratio*. We anticipate that the probability of a neighbor satisfying a peer's query is proportional to the semantic proximity between the peer and its neighbor. We aim, therefore, at filling a peer's semantic list with its ℓ_{sem} semantically closest peers out of the whole network.

We define the semantic proximity between two nodes as the number of common files they have. More formally, given two nodes P and Q, with file lists F_P and F_Q, respectively, their semantic proximity is defined as:

$$SemProx(P,Q) = |F_P \bigcap F_Q|$$

The more common files two nodes have—therefore, the semantically closer they are—the higher the value of *SemProx*. Essentially, for each node P we are seeking peers $Q_1, Q_2, \ldots, Q_{\ell_{sem}}$ for its semantic list, that maximize the sum:

$$\sum_{i=1}^{\ell_{sem}} SemProx(P,Q_i)$$

The VICINITY *selection function* $S(k,P,\mathcal{D})$ (see Section 4.2.2) is consequently, defined based on the proximity function *SemProx*. It simply sorts the nodes in $\mathcal{D}$ based on their semantic proximity to node P, and selects the k closest ones.

7.3. GOSSIPING FRAMEWORK

In the context of the VICINITY/CYCLON framework, each node runs two protocols, VICINITY and CYCLON. Each protocol maintains a separate view of size ℓ_{vic} and ℓ_{cyc}, respectively, and exchanges a different number of descriptors per gossip exchange (i.e., has a different gossip length), g_{vic} and g_{cyc}, respectively. The semantic list defined earlier, consists of the ℓ_{sem} neighbors of closest semantic proximity among the ones in the VICINITY view.

For the sake of demonstrating the importance of certain points in VICINITY's design, we consider the following three versions of VICINITY:

RANDOM VICINITY This is the most naive of the three versions. When gossiping, a node selects a *random* subset of descriptors from its VICINITY view to send to the other peer.

SELECTIVE VICINITY In this—better—version, a node selects and sends the subset of descriptors from its VICINITY view that is *optimal* for the recipient.

COMPLETE VICINITY This is essentially the fully-fledged VICINITY protocol, as defined in Chapter 4. The sender selects and sends the descriptors that are *optimal* for the recipient, considering also descriptors from its CYCLON—in addition to its VICINITY—view.

In terms of the generic gossiping skeleton that VICINITY follows (reproduced in Fig. 7.1 for convenience), Figure 7.2 presents the formal description of the actions taken by the three *hooks*. The only hook in which the aforementioned versions differ is `selectToSend()`.

In addition to these three VICINITY versions, we also examine a fourth configuration, namely RANDOM VICINITY *without* CYCLON. This is the only single-layer configuration that we consider.

```
do forever {
  wait(T time units)
  incr. all descriptors' age by 1
  Q ← selectPeer()
  remove Q from view
  buf_send ← selectToSend()
  send buf_send to Q
  receive buf_recv from Q
  view ← selectToKeep()
}
```

Active thread

```
do forever {
  receive buf_recv from Q
  buf_send ← selectToSend()
  send buf_send to Q
  view ← selectToKeep()
}
```

Passive thread

Figure 7.1: The generic gossiping skeleton for CYCLON and VICINITY.

7.4. EXPERIMENTAL ENVIRONMENT AND SETTINGS

All experiments presented here have been carried out with PeerSim, an open source simulator for P2P protocols, developed in Java at the University of Bologna [Peer-

Hook	Action taken
`selectPeer()`	Select descriptor with the oldest age
`selectToSend()`	
RANDOM	Make a copy of the VICINITY view. Add own descriptor with own profile and age 0. Randomly select g_{vic} descriptors.
SELECTIVE	Make a copy of the VICINITY view. Add own descriptor with own profile and age 0. Select the best g_{vic} descriptors for Q.
COMPLETE	Merge the VICINITY and CYCLON views. Add own descriptor with own profile and age 0. Select the best g_{vic} descriptors for Q.
`selectToKeep()`	Merge the VICINITY, CYCLON, and received views. Select the best ℓ_{vic} descriptors for P.

Figure 7.2: Implementation of the generic gossiping skeleton hooks, for the 3 VICINITY versions.

Sim].

To evaluate the construction of semantic overlays, we used real world traces from the eDonkey file sharing system [eDonkey], collected by Le Fessant et al. in November 2003 [Fessant et al. 2004]. A set of 11,872 world-wide distributed peers along with the files each one shares is logged in these traces. A total number of 923,000 unique files is being collectively shared by these peers.

In order to simplify the analysis of our system's emergent behavior, we determined equal gossiping periods for both layers. More specifically, once every T time units each node initiates first a gossip exchange with respect to its bottom (CYCLON) layer, immediately followed by a gossip exchange at its top (VICINITY) layer. Note that even though nodes initiate gossiping at universally fixed intervals, they are not synchronized with each other. Similarly to previous chapters, we study the evolutionary behavior in terms of cycles, where one cycle is a period of T time units.

A number of parameters had to be set for these experiments, listed here.

Semantic list size (ℓ_{sem}) In all experiments the semantic list consisted of the 10 semantically closest peers in the VICINITY view. As shown in [Handurukande et al. 2004], a semantic list size of $\ell_{sem} = 10$ provides a good tradeoff between the number of nodes contacted in the semantic search phase and the expected semantic hit ratio.

View size (ℓ_{vic}, ℓ_{cyc}) For the view size selection, we are faced with the following tradeoff for both protocols. A large view size provides higher chances of making better neighbor selections, and therefore accelerate the construction of (near-)optimal semantic lists. On the other hand, the larger the view size, the longer it takes to contact all peers in it, resulting in the accumulation of older—and therefore more likely to be invalid—links. Of course, a larger view also takes up more memory, although this is generally not a significant constraint nowadays.

Considering this tradeoff, and based on experiments not further described here, we fixed the view size to 100 as a basis to compare different configurations. When both VICINITY and CYCLON are used, they are allocated 50 view entries each.

Gossip length (g_{vic}, g_{cyc}) The gossip length, that is, the number of descriptors gossiped per gossip exchange per protocol, is a crucial factor for the amount of bandwidth used. This becomes of greater consequence in this application, considering that a descriptor carries the file list of its respective node. So, even though exchanging more descriptors per gossip exchange allows information to disseminate faster, we are inclined to keep the gossip lengths as low as possible, as long as the system's performance is reasonable.

Again, for the sake of comparison, we fixed the total gossip length to 6 descriptors. When both VICINITY and CYCLON are used, each one is assigned a gossip length of 3.

Gossip period (T) The gossip period is a parameter that does not affect the protocol's behavior. The protocol evolves as a function of the number of messages exchanged, or, consequently, of the number of cycles elapsed. The gossip period only affects how fast the protocol's evolution will take place in time. The single constraint is that the gossip period T should be adequately longer than the worse latency throughout the network, so that gossip exchanges are not favored or hindered due to latency heterogeneity. A typical gossip period would be 1 minute, even though this does not affect the following analysis.

Figure 7.3 summarizes the settings the four configurations we experiment with.

	ℓ_{vic}	g_{vic}	ℓ_{cyc}	g_{cyc}
RANDOM VICINITY	100	6	-	-
RANDOM VICINITY + CYCLON	50	3	50	3
SELECTIVE VICINITY + CYCLON	50	3	50	3
COMPLETE VICINITY + CYCLON	50	3	50	3

Figure 7.3: The four configurations we compare.

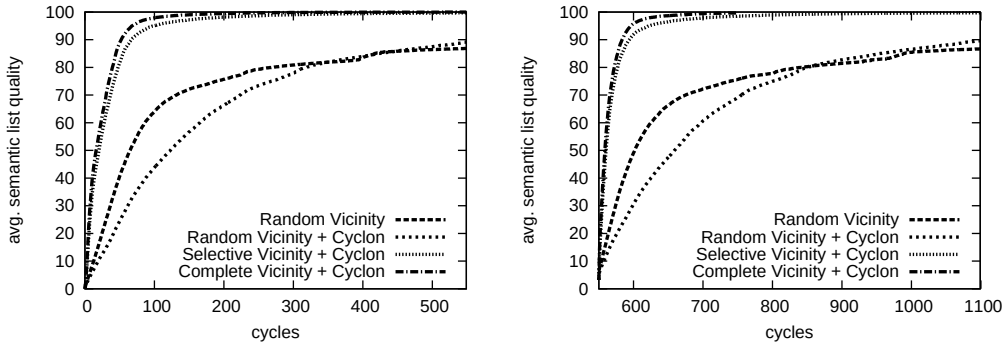


Figure 7.4: (a) Convergence of sem. views' quality. (b) Evolution of semantic lists' quality for a sudden change in all users' interests at cycle 550.

7.5. PERFORMANCE EVALUATION

In this section, we evaluate the performance of our framework in five different settings.

7.5.1. Convergence Speed on Cold Start

To evaluate the convergence speed of our algorithm, we first test how quickly it groups semantically related peers, when starting with a semantically-unaware network.

The objective, as imposed by the proximity function, is for each node to discover the ℓ_{sem} peers that have the most common files with it. We define a node's *semantic list quality* to be the ratio of the number of common files shared with its current ℓ_{sem} semantic neighbors, over the number of common files it would share with its ℓ_{sem} optimal semantic neighbors.

Figure 7.4(a) shows the average semantic list quality as a function of the cycle for four distinct configurations. In favor of comparison fairness, the view size and

gossip length are 50 and 3, respectively, in each layer, for all configurations. The only exception is the first configuration, which has a single layer. In this case, the view size and gossip length are 100 and 6, respectively. All experiments start with each node knowing 5 random other ones, simply to ensure initial connectivity in a single connected cluster.

In the first configuration, RANDOM VICINITY is running stand-alone. The progress of the semantic lists' quality is rather steep in the first 100 cycles, but as nodes gradually concentrate on their very own neighborhood, getting to know new, possibly better peers becomes rare, and progress slows down.

In the second configuration, a two-layered approach consisting of RANDOM VICINITY and CYCLON is running. The slow start compared to stand-alone VICINITY is a reflection of the smaller VICINITY view (3 as opposed to 6). However, the two-layered approach's advantage becomes apparent later, when CYCLON keeps feeding the RANDOM VICINITY layer with new, uniformly randomly selected nodes, maintaining a higher progress rate, and outperforming stand-alone VICINITY in the long run.

In the third configuration, SELECTIVE VICINITY demonstrates its contribution, as progress is significantly faster in the initial phase of the experiment. This is to be expected, since the descriptors sent over in each SELECTIVE VICINITY communication, are the ones that have been selected as the semantically closest to the recipient.

Finally, in the fourth configuration, COMPLETE VICINITY keeps the progress rate high even when the semantic lists are very close to their optimal state. This is due to the broad random sampling achieved by this version. In every communication, a node is exposed to the best peers out of 50 *random* ones, in addition to 50 peers from its neighbor. In this way, semantically related peers that belong to separate semantic clusters quickly discover each other, and subsequently the two clans merge into a single cluster in practically no time.

7.5.2. Adaptivity to Changes of User Interests

In order to test our protocol's adaptivity to dynamic user interests, we ran experiments where the interests of some users changed. We simulated the interest change by picking a random pair of nodes and swapping their file lists in the middle of the experiment. At that point, these two nodes found themselves with semantic lists unrelated to their (new) file lists, and therefore had to gradually climb their way up to their new semantic vicinity, and replace their useless links by new, useful ones.

Once again, we present the worst case—practically unrealistic—scenario, of *all* nodes changing interests at once, at cycle 550 of the experiment of figure 7.4(a). The evolution of the average semantic list quality from the moment when

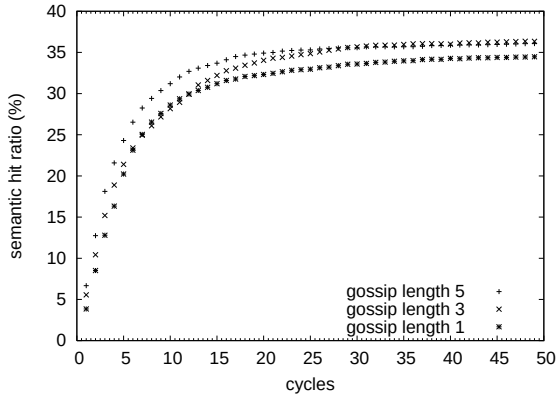


Figure 7.5: Semantic hit ratio, for gossip lengths 1, 3, and 5 in each layer.

all nodes change interests, is presented in figure 7.4(b). The faster convergence compared to figure 7.4(a) is due to the fact that views are already fully filled up at cycle 550, so nodes have more choices to start looking for good candidate neighbors.

Even though this scenario is very unrealistic, it demonstrates the power of our protocol in adapting to even massive scale changes. This adaptiveness is due to the priority given to newer descriptors in `selectToKeep()`, which allows a node's descriptors with updated semantic information to replace older descriptors of that node fast.

7.5.3. Effect on Semantic Hit Ratio

In order to further substantiate our claim that semantic based clustering endorses P2P searching, we conducted the following experiments. A randomly selected file was removed from *each* node, and the system was run considering proximity based on the remaining files. Then, each node did a search on the file it was missing. We measured the semantic hit ratio to be over 36% for a semantic list of size 10.

Figure 7.5 presents the semantic hit ratio as a function of the cycle. Three experiments are shown, with gossip lengths for *both* layers set to 1, 3, and 5. Note that computation of the hit ratio for each cycle was made offline, without affecting the mainstream experiment's state.

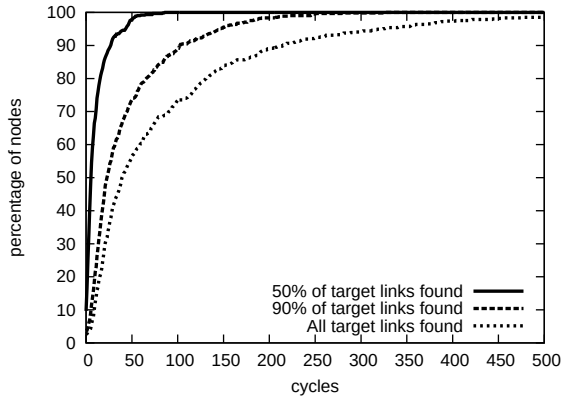


Figure 7.6: CDF of the speed by which the semantic list of a joining node is filled with optimal neighbors.

7.5.4. Single Node Joins

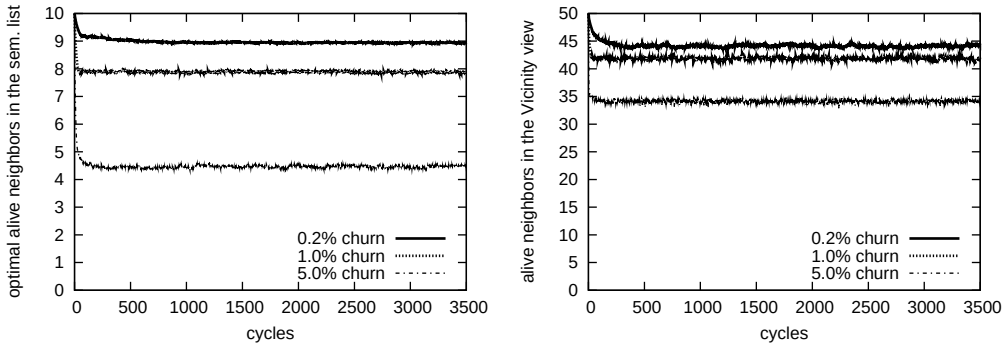
To examine how fast new nodes join an already semantically clustered network, we conducted a series of experiments with COMPLETE VICINITY. Each experiment started with a network from which one randomly selected node had been removed. After the network converged, we added the missing node and measured the number of cycles it took to fill, respectively 50%, 90%, and 100% of its semantic list with *optimal* neighbors. The respective cumulative distribution function graphs are shown in Figure 7.6.

These experiments clearly show that the semantic list is rapidly filled with optimal neighbors for the vast majority of the nodes, although some may take considerably more time. It can also be seen that, although discovering *all* best neighbors may take arguably long for some nodes, it takes significantly fewer cycles to discover *most* best neighbors (CDFs for finding 50% and 90% of best neighbors shown).

7.5.5. Behavior under Node Churn

To investigate the behavior of our algorithm as nodes regularly join and leave the network, we evaluated COMPLETE VICINITY under different levels of node churn.

For these experiments, we considered an initial converged network of 10,000 nodes. During each cycle we removed n nodes and replaced them with n other ones. Every node in the system corresponded to one node in the eDonkey traces (i.e., stores the same files), which contained a total of 11,872 nodes. Therefore, at



(a) (b)

Figure 7.7: (a) The average number of optimal alive neighbors in the semantic list. (b) The average number of alive neighbors in the VICINITY view.

any moment in time a random subset of 10,000 out of 11,872 nodes was active, and the remaining 1872 nodes were down. The analysis of churn rates from real-world Gnutella traces [Saroiu et al. 2002, 2003] shows that the set of active nodes changes by approximately 0.2% every 10 seconds. In other words, if we assume a cycle length of 10 seconds, a realistic value for n is 20. We have also experimented with larger churn rates, namely 1% and 5%, which correspond to a cycle duration of 50 and 250 seconds, respectively.

The results of these experiments are shown in Figure 7.7(a) in which the average number of optimal alive neighbors in semantic lists under different churn rates is plotted. We see that as the churn rate increases, the semantic lists generally remain polluted with links to non-optimal neighbors. This can be easily explained by considering the VICINITY view (from which the neighbors for the semantic lists are extracted). In Figure 7.7(b), we see that under high churn the view contains a relatively large fraction of links to dead nodes. As these inactive links may refer to nodes with a large number of common files, they prevent establishing optimal links with nodes that are alive but share a smaller number of files. Moreover, when a node is reborn, it can take some time for other nodes, which now may have non-optimal semantic lists, to establish links to it (see Fig. 7.6).

7.6. BANDWIDTH CONSIDERATIONS

Due to the periodic behavior of gossiping, the price of having rapidly converging and accurate protocols may inhibit a high usage of network resources

(i.e., bandwidth).

In each cycle, a node gossips on average twice (exactly once as an initiator, and on average once as a responder). In each gossip $2 \cdot (g_{vic} + g_{cyc})$ descriptors are transferred to and from the node, resulting in a total traffic of $4 \cdot (g_{vic} + g_{cyc})$ descriptors for a node per cycle. An descriptor's size is dominated by the file list it carries. A single file is identified by its 128-bit (16-byte) MD4 hash value. Analysis of the eDonkey traces [Fessant et al. 2004] revealed an average number of 100 files per node (more accurately, 99.35). Therefore, a node's file list takes on average 1,600 bytes. So, in each cycle, the total number of bytes transferred *to* and *from* the node is $6,400 \cdot (g_{vic} + g_{cyc})$.

For $g_{vic} = g_{cyc} = 3$, the average amount of data transferred to and from a node in one cycle is 38,400 bytes, while for $g_{vic} = g_{cyc} = 1$, it is just 12,800. Maintaining almost optimal semantic lists requires frequent gossiping to account for the churn. Based on the traces, to achieve 90% optimality of the semantic list, the churn rate must be limited to approximately 0.2%. Consequently, the gossip period T must be equal to 10 seconds, which translates to an average bandwidth of 3840 bytes per second for $g_{vic} = g_{cyc} = 3$, and 1280 bytes per second for $g_{vic} = g_{cyc} = 1$. However, if 80% optimality is acceptable, the gossip period can be reduced to 50 seconds, which yields 768 and 256 bytes per second, respectively, a factor of 5 improvement traded for only 10% quality degradation.

We consider such a bandwidth consumption to be rather small, if not negligible compared to the bandwidth used for the actual file downloads. It is, in fact, a small price to pay for relieving the default search mechanism from about 35% of the search load, which is often significantly higher (e.g., flooding or random-walk search). Moreover, the bandwidth can be further reduced by employing techniques such as Bloom filters [Bloom 1970] and on-demand fetching of file lists, instead of associating a full file list with each node descriptor.

7.7. DISCUSSION AND RELATED WORK

To the best of our knowledge, all earlier work on implicit building of semantic overlays relies on using heuristics to decide *which* of the peers that served a node recently are likely to be useful again in future queries [Sripanidkulchai et al. 2003; Voulgaris et al. 2001; Handurukande et al. 2004].

However, all these techniques inhibit a weakness that challenges their applicability to the real world. They all assume a *static* network, free of node departures, which is a rather strong assumption considering the highly dynamic nature of file-sharing communities. Also, it is not clear how they perform in the presence of dynamic user preferences.

Moreover, as opposed to the existing solutions, our algorithm can, to some extent, help against so-called free-riders in the P2P file sharing networks [Adar and Huberman 2000]. Free-riders provide no files to be downloaded by other users, but still use the network to obtain files that are interesting for themselves. Because of a lack of shared files, VICINITY does not create any meaningful semantic lists for such misbehaving nodes. This, combined with heuristics forbidding frequent usage of the backup search algorithms, minimizes the number of successful searches for free-riders, and consequently discourages the free-riding practice.

Regarding proximity-based P2P clustering, our work comes close to the T-Man protocol [Jelasity and Babaoglu 2005], which has been developed independently. Although there were significant differences with the original T-Man protocol, the most recent version shows a strong similarity with our work. An important difference remains that the VICINITY/CYCLON framework offers higher flexibility in controlling bandwidth, by arbitrarily deciding on the number of descriptors that need to be exchanged. This becomes important in this application, where a node descriptor contains the node's file list. More important, however, is that we show that the VICINITY/CYCLON topology construction framework can be successfully applied to forming semantic overlays for searching in peer-to-peer file-sharing systems. Such an evaluation has not yet been done before.

Note that we chose a rather simple, yet intuitive proximity function to test our protocol with. Our goal was to demonstrate the power of the VICINITY/CYCLON framework in forming a semantic overlay network based on *a* proximity function. Even though much richer proximity functions could have been applied, it was out of the scope of this research.

Concluding, in this work we introduced the idea of applying epidemics to proactively build and dynamically maintain semantic lists in a large-scale file-sharing system. Specifically, we showed that the VICINITY/CYCLON two-layered approach is the appropriate way to build such a service.

ACKNOWLEDGEMENTS

We would like to thank Fabrice Le Fessant for providing us with the eDonkey2000 traces [Fessant et al. 2004] he gathered in November 2003.

CHAPTER 8

SUB-2-SUB: Purely P2P Publish/Subscribe

Publish/Subscribe systems are designed to disseminate messages (*events*), from nodes issuing events (*publishers*), to nodes interested in receiving events (*subscribers*). Subscribers typically register with the system the type of events they are interested in. Publishers simply post events. Subsequently, it is the system's responsibility to deliver the right events to the right places. That is, an event should be delivered to *all* its subscribers and *no one else*.

Consider, for instance, a house rental service. Prospective renters *subscribe* to the service by setting their search criteria. Such criteria may include their preferred neighborhood(s), the price range they are willing to pay, the suitable number of bedrooms, sufficient area, etc. Some subscribers may even set only *some* of the aforementioned criteria, should the rest be irrelevant to their decision. Home owners, on the other hand, *publish* announcements (*events* in publish/subscribe terminology) advertising their property for rent. The system is responsible for delivering *each* rental announcement to *all* prospective renters whose criteria classify it as a potentially suitable choice.

In this work we address the problem of constructing scalable content-based publish/subscribe systems. Subscriptions can range from a simple specification of merely the type of an event to a specification of the value ranges that an event's attributes can have. Notably the latter poses potential scalability problems.

Structured peer-to-peer systems can provide scalable solutions to publish/subscribe systems with simple subscription patterns. For complex subscription types their applicability is less obvious. In this chapter, we present SUB-2-SUB, a collaborative self-organizing publish/subscribe system deploying an unstructured overlay network. SUB-2-SUB relies on an epidemic-based algorithm in which peers continuously exchange subscription information to get clustered to similar peers.

In contrast to many existing approaches, SUB-2-SUB supports both value-based and interval-based subscriptions. Simulations of SUB-2-SUB on synthetic and reusable workloads convey its good properties in terms of routing efficiency, fairness, accuracy and efficiency.

A shorter version of this chapter appears in [Voulgaris et al. 2006].

8.1. OVERVIEW

Peer-to-peer (P2P) systems have been identified as the key to scalability and their self-organizing properties make them natural candidates for large-scale publish/subscribe systems design. Several efficient implementations of P2P topic-based publish/subscribe systems have been proposed [Castro et al. 2002; Banerjee et al. 2002]. Unfortunately, it is not obvious how to devise a scalable P2P solution for content-based publish/subscribe systems.

Structured P2P overlays [Rowstron and Druschel 2001a; Ratnasamy et al. 2001a; Stoica et al. 2001] have often been favored over unstructured ones to implement content-based publish/subscribe systems. The idea is to map the attribute space of the latter to the identifier space of the former. At one extreme, each attribute is associated with one specific peer. Although this provides efficient routing to interested subscribers, peers hosting popular attributes are quickly overloaded. At the other end of the spectrum, in an attempt to discretize the ranges of attributes, a peer is made responsible for a specific *(attribute, value)* pair. In this case, attaining a scalable implementation for range subscriptions becomes problematic.

In this work, we step away from structured overlays and propose a fully decentralized and self-organizing approach based on unstructured overlays to deal efficiently with both exact and range subscriptions. Key to our approach, which is called SUB-2-SUB, is that subscribers to the same events are automatically clustered. SUB-2-SUB leverages the overlapping intervals of range subscriptions and creates an unstructured overlay reflecting the structure of the attribute space *and* that of the set of subscriptions. Once subscriptions are clustered, events are directly posted to the proper cluster where they are efficiently disseminated.

A key issue is that SUB-2-SUB is highly reactive to changes in the set of subscriptions. To this end, it deploys an enhanced epidemic protocol based on the VICINITY and PEER SAMPLING SERVICE topology construction framework, to continuously cluster subscribers based on their subscriptions. Clustering is based on a proximity metric in the attribute space, which results in nodes of similar subscriptions clustering with each other. A similar process is followed to navigate publishers to clusters of matching subscriptions. Publishers progress greedily

across the network according to the same algorithm and proximity metric, eventually reaching the cluster that contains *exactly* the subscribers which the event should be delivered to. Moreover, within such a cluster, subscribers are loosely organized into a distributed data structure that enables efficient event dissemination.

8.2. ISSUES IN PUBLISH / SUBSCRIBE SYSTEMS

A number of issues are of interest when designing a publish/subscribe system. It is important for the remaining of the chapter to list the most significant issues that characterize the quality of a publish/subscribe system.

Hit ratio The hit ratio is defined as the percentage of subscribers that receive an event, over the total number of subscribers interested in it. It is a metric of the penetration of events. Ideally, an event should be delivered to *all* nodes interested in it (hit ratio 100%), or at least to as many of them as possible.

Spam ratio The spam ratio is defined as the percentage of subscribers that receive an event although it was *not* matching their subscriptions, over the total number of subscribers interested in the event. Delivering an event to subscribers not interested in it wastes network and processing resources. Therefore, the spam ratio should be kept as low as possible, ideally 0%.

Dissemination speed The speed at which an event dissemination completes. The two main factors governing dissemination speed are (a) the average delay in forwarding messages (processing delay on nodes plus network latency), and (b) the number of hops messages take to reach the most distant nodes. In our evaluation we focus on the latter factor.

8.3. SYSTEM MODEL

In brief, we consider a conjunctive attribute-based publish/subscribe system with N floating-point attributes that supports subscriptions on both *exact* attribute values and *ranges*.

More formally, we assume a fixed number N of attributes, $A_1, \ldots, A_N$, with values in $\mathbb{R}$ (the set of real numbers). Attributes can alternatively be assigned values of any type that can be directly mapped to $\mathbb{R}$, such as integers, enumerations, boolean values, or strings.

Subscriptions are conjunctions of predicates on one or more attributes. A predicate can denote either an exact value (e.g., $A_i = v$), or a continuous range

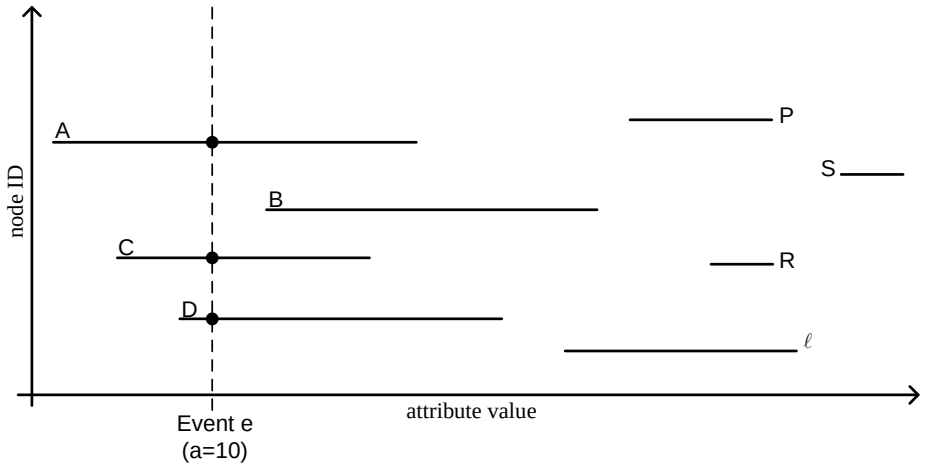


Figure 8.1: A set of subscriptions and an event.

of values (e.g., $A_i \in [v_{min}, v_{max}]$). A subscription can have at most one predicate per attribute. Multiple exact values or multiple noncontinuous subranges on a single attribute can be modeled as multiple separate subscriptions. Attributes not referred to in a subscription (wildcards) are assumed to cover the whole attribute space, that is, their value is indifferent to the subscriber. An example subscription for a 5-attribute system is $S: \langle A_1, A_2, A_3, A_4, A_5\rangle = \langle *, 30, *, *, [2.2, 2.7]\rangle$

Events are N-sized vectors specifying exact values for *all* attributes. An example of an event for a 5-attribute system is $E: \langle A_1, A_2, A_3, A_4, A_5\rangle = \langle 5, 3, -2.5, 20, 1.87\rangle$.

In the remainder of this chapter, we will consider range subscriptions, i.e., subscriptions composed of a set of predicates, each specifying a range of values. Exact-value predicates are considered as a special, and simpler, case of a range subscription.

Also, nodes whose subscriptions match a particular event will be referred to as the event's *matching subscribers*.

8.4. SUB-2-SUB IN A NUTSHELL

SUB-2-SUB is an autonomous, self-organizing P2P event-notification system that supports multi-attribute subscriptions. *Autonomous* implies that the dissemination of events to all interested nodes is accomplished by the cooperation of interested nodes themselves, eliminating any dependency on relay servers or ded-

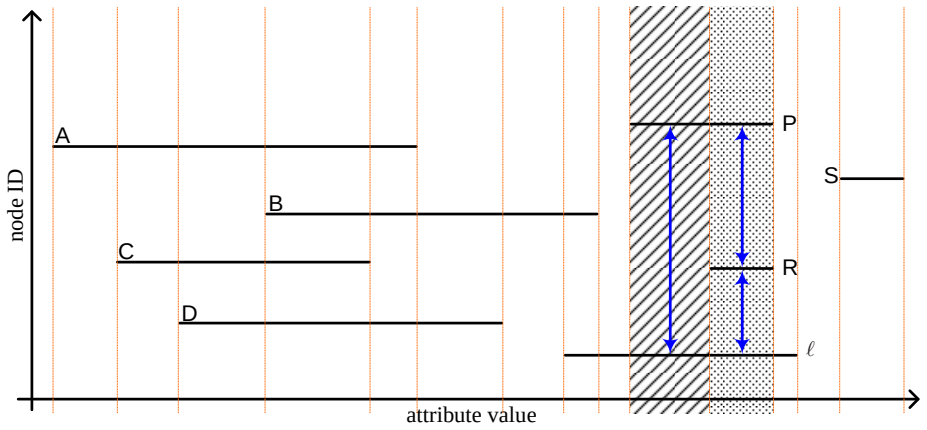


Figure 8.2: Partitioning of an one-dimensional event space in homogeneous subspaces $\mathcal{H}_i$.

icated elements. *Self-organizing* refers to the fact that nodes organize themselves in a structure that enables their cooperation for event dissemination in a completely decentralized manner. The self-organizing property of SUB-2-SUB relies on the use of the VICINITY/CYCLON topology construction framework to cluster peers of similar subscriptions. Its efficiency relies on the fact that overlapping subscriptions are leveraged so that *(i)* only interested subscribers are reached by an event and *(ii)* subscribers do not miss any event matching their subscription.

Clustering overlapping subscriptions SUB-2-SUB forms an unstructured overlay network in which each peer is associated with one subscription. Multiple subscriptions are handled by running multiple virtual peers on a single physical node. In the context of a customized version of the VICINITY/CYCLON framework, peers periodically exchange information to discover peers with similar subscriptions to form clusters with. Note that the resulting clusters do not have explicit boundaries. Figure 8.1 depicts an example of a set of subscriptions for a single attribute scheme. Each line represents a range subscription. The epidemic algorithm ensures that peers are automatically clustered so that when an event specifying a value for attribute a (e.g., $e : \langle a = 10 \rangle$ in Figure 8.1) is published, all interested subscribers (A, C, and D in Figure 8.1) get it.

Partitioning the event space A key observation underlying SUB-2-SUB's design is that every peer P's subscription specifies an N-dimensional *rectangular* subspace $\mathcal{S}_P \subseteq \mathbb{R}^N$, which we refer to as P's *subscription subspace*. As a conse-

quence, we are interested only in those events that fall into $\mathcal{S} = \bigcup \mathcal{S}_P$. To simplify the SUB-2-SUB model, we *conceptually* partition $\mathcal{S}$ into disjoint subspaces $\mathcal{H}_1, \mathcal{H}_2, \ldots$, which we name *homogeneous subspaces*:

$$\bigcup \mathcal{H}_i = \mathcal{S}$$

and

$$\forall\, i \neq j: \quad \mathcal{H}_i \cap \mathcal{H}_j = \emptyset$$

All events belonging to a given homogeneous subspace $\mathcal{H}_i$ have the exact same set of matching subscribers (hence the name *homogeneous* for these subspaces). More formally:

$$\forall\, i, P: \quad [\mathcal{H}_i \cap \mathcal{S}_P \neq \emptyset] \Rightarrow [\mathcal{H}_i \subseteq \mathcal{S}_P]$$

Furthermore, we demand that the set of homogeneous subspaces $\mathcal{H}_i$ is minimal: there is no partitioning with fewer parts that can satisfy the aforementioned constraint.

The notion of homogeneous subspaces provides a level of abstraction: it permits us to think in terms of groups of events—that have the same set of matching subscribers—rather than for single events individually. Figure 8.2 shows how the event space of the example presented in Figure 8.1 is partitioned in homogeneous subspaces.

The ultimate goal in SUB-2-SUB is, for each homogeneous subspace $\mathcal{H}_i$, to cluster its nodes in such a way that an event $e \in \mathcal{H}_i$ can be efficiently disseminated to all of them. Essentially, we want to maintain a connected overlay (as we will see, a bidirectional ring) for every homogeneous subspace $\mathcal{H}_i$. If that holds, then any possible event in the system can be efficiently disseminated to all its matching subscribers. To this end, we let peers periodically exchange their subscriptions. If two peers P and Q note that $\mathcal{S}_{PQ} \equiv \mathcal{S}_P \cap \mathcal{S}_Q \neq \emptyset$, they will record this fact and maintain references to each other (how this is done is described below). For example in Figure 8.2, peers P and Q satisfy the above condition for a given range and get connected. When discovering a third peer, R, with $\mathcal{S}_{PQR} \equiv \mathcal{S}_P \cap \mathcal{S}_Q \cap \mathcal{S}_R \neq \emptyset$, peers P, Q and R will further organize into a structure associated with $\mathcal{S}_{PQR}$ such that an event $e \in \mathcal{S}_{PQR}$ will be efficiently disseminated to the three peers. Both P and Q will still maintain references to each other, but now for the subspace $\mathcal{S}_{PQ} - \mathcal{S}_{PQR}$, if not empty. Figure 8.2 illustrates this process: when R joins the network it gets connected to P and Q for the shaded range while P and Q remain connected for the hatched range.

The publisher of an event e joins the overlay identically to subscribers, and will eventually find the set $\mathcal{H}_i$ where e belongs to. At that point, e is disseminated to the members associated with $\mathcal{H}_i$. Note that, provided $\mathcal{S}$ is indeed partitioned along the lines we just described, e will reach *only* the nodes that are interested

in it, and no others. This method of letting publishers locate the relevant subscribers is primarily elegant, but not necessarily efficient. Higher efficiency can be achieved through, for example, greedy routing algorithms. We do not consider such alternatives in this dissertation.

8.5. THE SUB-2-SUB DISSEMINATION OVERLAY

In Chapter 6 we explored epidemic schemes for group communication. Such schemes provide strong probabilistic guarantees for the dissemination of messages to *all* members of an overlay, demonstrating tolerance to churn and node failures. Here we are looking at employing one of these techniques, namely *hybrid dissemination* and the RINGCAST algorithm (see Section 6.5), in *selectively* disseminating events to all their matching subscribers.

What differentiates dissemination in publish/subscribe systems from the generic dissemination model, is that there is no *single* set of target nodes. Instead, each event needs to be disseminated to a different set of nodes, consisting of the event's matching subscribers. Group communication, therefore, should be applicable to any possible set of matching subscribers.

This is exactly what lies in the core of SUB-2-SUB: A *dissemination overlay* that enables group communication among *exactly* the matching subscribers of *any* given event, independently. Considering that events within a homogeneous subspace have the same set of matching subscribers, the SUB-2-SUB dissemination overlay should enable group communication among nodes with subscriptions intersecting $\mathcal{H}_i$, for every $\mathcal{H}_i$ independently.

Let $\mathcal{N}_i$ denote the set of nodes whose subscriptions intersect homogeneous subspace $\mathcal{H}_i$. In Section 6.5.1 we saw that organizing a group of nodes in a RINGCAST overlay, that is, a bidirectional ring augmented by random shortcuts, enables inexpensive and efficient dissemination of messages among them. The major issue that SUB-2-SUB needs to solve is to organize the nodes in *each* set $\mathcal{N}_i$ in a RINGCAST overlay. Along these lines, each node is equipped with a random *sequence ID* uniformly drawn from a large identifier space. If nodes P and Q are both in $\mathcal{N}_i$, and there is no other node in $\mathcal{N}_i$ whose sequence ID lies between those of P and Q (using modulo arithmetic), they will keep a link to each other. These links, known as *ring links*, organize nodes of any given homogeneous subspace in a bidirectional ring. In addition to ring links, nodes also maintain links to random other nodes in the same set $\mathcal{N}_i$, known as *(random) overlapping-interest* links. Figure 8.3 shows the links forming a ring for each homogeneous subspace. Random overlapping-interest links are omitted for clarity.

Often, the subscriptions of two nodes may overlap in multiple homogeneous

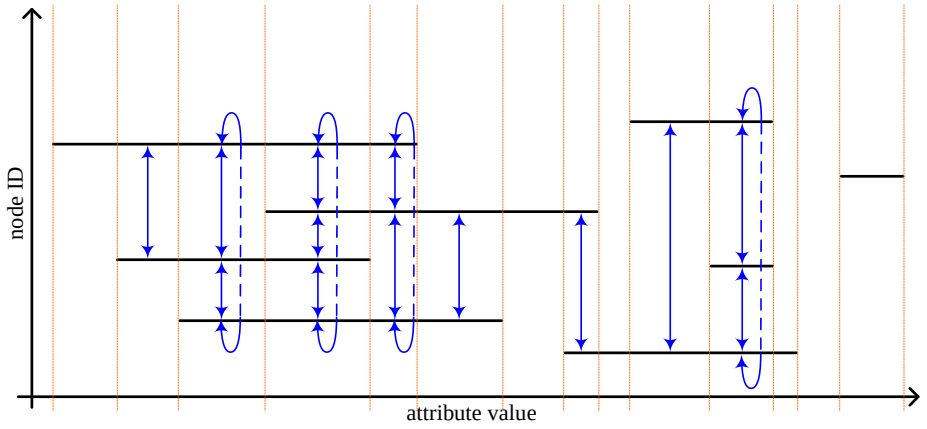


Figure 8.3: The conceptual SUB-2-SUB dissemination overlay. For each homogeneous subspace nodes are linked in a ring structure. Only the *ring links* are shown. *Random overlapping-interest* links are omitted for clarity.

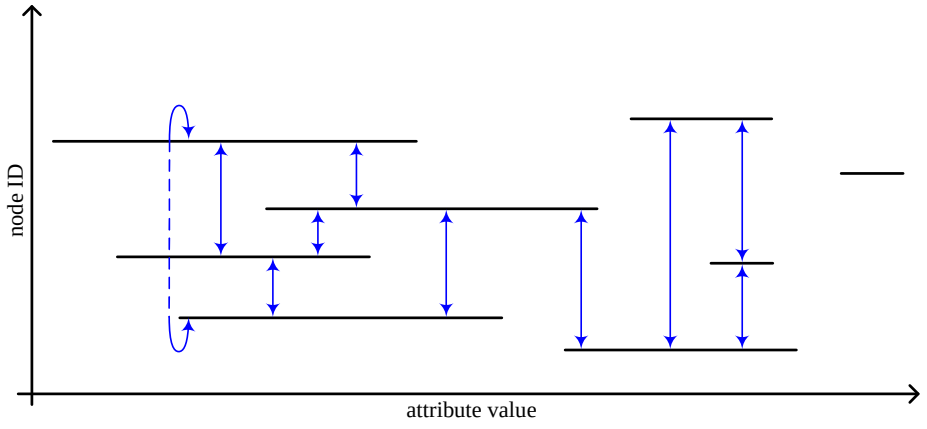
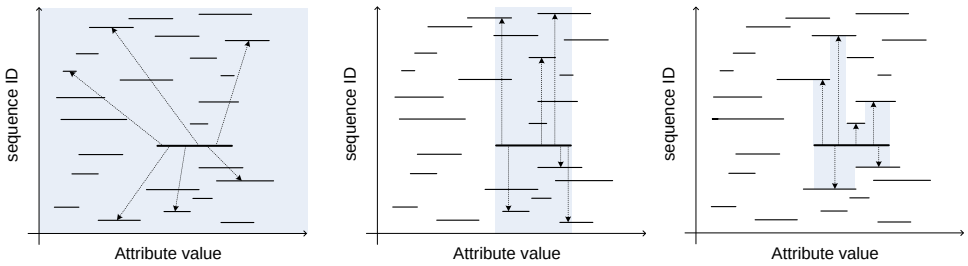


Figure 8.4: The actual SUB-2-SUB dissemination overlay. For any possible event all matching subscribers are linked in a ring structure. However, no multiple links between the same two nodes are kept. Only the *ring links* are shown. *Random overlapping-interest* links are omitted for clarity.



(a) Random links (PEER SAMPLING SERVICE)

(b) Overlapping-interest links (VICINITY)

(c) Ring links (VICINITY)

Figure 8.5: The three sets of links each subscriber should maintain. Shaded areas denote the areas where links of the respective type are appropriate (for the given subscriber).

subspaces, leading to multiple links between them, as shown in Figure 8.3. In reality, no multiple links are allowed between two nodes. This results in the *actual* SUB-2-SUB dissemination overlay for our example, shown in Figure 8.4.

These observations lead to the following three types of links:

Random links These are links to randomly selected peers from the whole network. They are needed to maintain the network connected, so that publishers and new subscribers can navigate to their appropriate homogeneous subspace irrespectively of the node they join the network through.

Overlapping-interest links They reflect the similarities between subscriptions and are used to send published events to random other interested peers (and to speed up event dissemination).

Ring links These links organize nodes of each homogeneous subspace $\mathcal{H}_i$ in a ring, ensuring that events belonging to $\mathcal{H}_i$ reach all matching subscribers by means of the respective ring.

Figure 8.5 depicts these three types of links, for a set of subscriptions over one attribute.

8.5.1. Spreading Events

Given the dissemination overlay described above, publishing events is a simple task. All a publisher has to do is locate *any* one matching subscriber for its potential event(s), and deliver the event(s) to it. From that point on, dissemination is taken care of by the matching subscribers themselves.

Subscribers disseminate events by means of the RINGCAST algorithm, presented in Section 6.5.1. In particular, we assume that each subscriber is running a daemon thread listening to incoming events and forwarding them accordingly. The daemon thread is activated when a node receives an event. It first looks up the node's recent event history. Previously seen events are ignored. New events are delivered to the application, and subsequently forwarded in two respects. First, an event is forwarded to the node's two adjacent neighbors (if any) along the event's ring. Second, it is forwarded along a small number (typically only one or two) of additional *random shortcuts*. Random shortcuts are links to matching subscribers, picked randomly among the overlapping-interest neighbors that match this particular event (if any). Obviously, a node does not send an event back along the same link it received it through.

Four facets of the dissemination algorithm are worth noting, namely its behavior with respect to *hit ratio*, *propagation speed*, *spam ratio*, and *load balancing*.

Hit ratio and propagation delay are dealt with by ring links and random shortcuts, respectively. In brief, forwarding along ring links guarantees that events are sequentially propagated to *all* corresponding matching subscribers, achieving a hit ratio of 100%. Random shortcuts to matching subscribers are followed only to boost propagation speed. Indeed, following the ring links alone requires linear time to cover all interested nodes. By each node forwarding incoming events to as few as one random matching node, dissemination completes in close to logarithmic time. A thorough evaluation of the RINGCAST algorithm's efficiency is presented in Section 6.6.

Spam is entirely out of the question in this dissemination algorithm. Clearly, no node forwards an event to another node, unless the latter is interested in that event. The only case a node may receive spam messages, is if it has recently changed its subscription, and its updated subscription has not yet spread enough.

Finally, with respect to load balancing, two points are worth emphasizing. First, no dissemination load is imposed on irrelevant subscribers. Second, load is evenly balanced across matching subscribers, as each of them receives an event once or a few more times, and upon first reception forwards it to the same small number of nodes: up to two adjacent neighbors, and a few random ones.

8.6. BUILDING THE DISSEMINATION OVERLAY

From our previous discussion, the publish/subscribe problem has evidently turned into a *topology construction* and *maintenance* problem. We handle it by means of the VICINITY/CYCLON framework for topology construction, presented in Chapter 4.

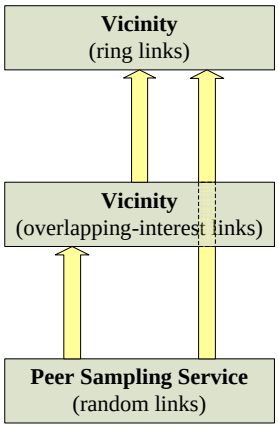


Figure 8.6: The SUB-2-SUB Architecture.

The SUB-2-SUB architecture explicitly sets a dual goal. Apart from random links to maintain connectivity, subscribers are required to dynamically maintain two types of links: overlapping-interest links, and ring links. Although this clearly suggests employing two instances of VICINITY, it is not the sole reason for doing so.

Discovering nodes with overlapping interests comprises a filtering over all participating nodes, based on a single criterion: relevance of subscriptions. As we will see, this filtering can be directly handled by VICINITY, given the appropriate selection function. Selecting a node's ring neighbors is more specific: they need to be of overlapping interest, but in addition we also need to consider their sequence ID. Selection of ring neighbors, thus, comprises a further filtering over a node's overlapping-interest neighbors. Consequently, discovering a node's overlapping-interest neighbors is a crucial step to discovering its ring neighbors too.

Given the aforementioned design considerations, we combine two instances of VICINITY on top of a single PEER SAMPLING SERVICE instance, resulting in the architecture depicted in Figure 8.6. Each layer is a gossiping protocol in itself, communicating directly with the respective layer of other nodes, as shown in Figure 8.7. The sole interaction between layers is by means of link recommendations, from lower to higher layers.

Note that ring links are indifferent to publishers. Indeed, publishers build views for only random and overlapping-interest links, and gossip greedily (as fast as they can) to reach *any* matching subscriber, independently of its sequence ID. As we will see in Section 8.7, this permits them to find a matching subscriber in a very small number of steps. If more steps than a small threshold elapse, they

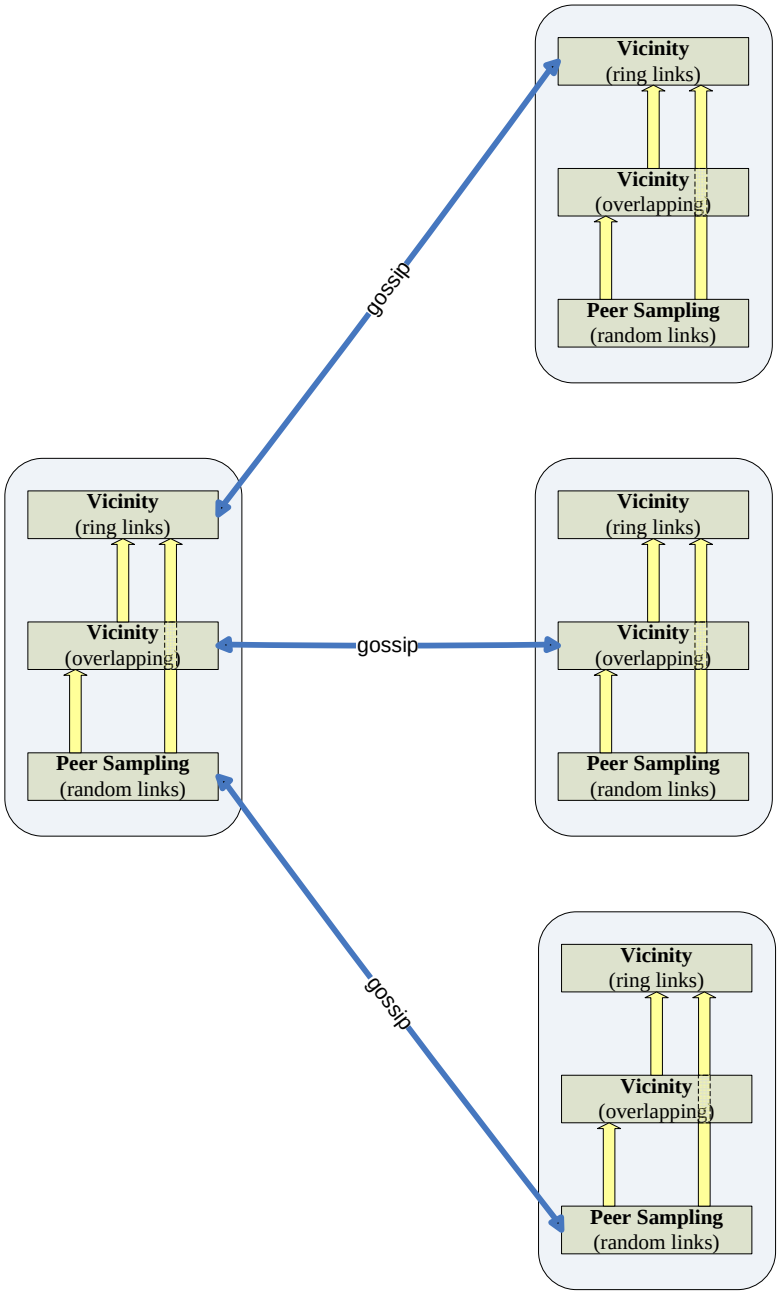


Figure 8.7: Layered communication. Each layer of a node gossips exclusively to the respective layer of other nodes. Interaction between layers is restricted to passing around links within a single node.

can safely assume that no subscriber in the whole network is interested in their event(s).

In the following three sections we will explain in detail the operation of each layer and the way they interact.

8.6.1. Building Random Links

Random links are handled by the bottom layer in our design, which consists of the PEER SAMPLING SERVICE, studied in Chapter 3. Its purpose is twofold. First, it keeps the whole set of subscribers connected in a single partition, even in the presence of churn or large scale failures. Connectivity is crucial to let publishers and new subscribers find their way to their appropriate neighborhood sets, irrespectively of *where* they initially joined the network. Second, it constitutes a source of links selected uniformly at random from the whole network. Random links do not play a direct role in event dissemination, but are fundamental for VICINITY (in this case for both instances of it), as we have seen in Chapter 4.

8.6.2. Building Overlapping-Interest Links

Overlapping-interest links are discovered by the middle layer, consisting of the VICINITY protocol. The operation of VICINITY has been studied in Chapter 4.

Of particular interest is the selection function we apply here. It is based on the notion of *subscription distance*, which we introduce as a measure of subscription similarity. The subscription of a node defines an N-dimensional rectangle, which we will refer to as its *hyper-rectangle*. We define the distance between two subscriptions to be 0 (zero) if they intersect (overlapping-interest nodes), or the Euclidean distance between their hyper-rectangles otherwise.

More formally, the distance between two subscribers P and Q with subscriptions

$$\text{subscription of } P: \quad A_i \in [p_i^{min}, p_i^{max}], \quad i = 1 \ldots N$$

$$\text{subscription of } Q: \quad A_i \in [q_i^{min}, q_i^{max}], \quad i = 1 \ldots N$$

is given as follows:

$$distance(P,Q) = \begin{cases} 0 & \text{, overlapping;} \\ \sqrt{\sum_{i=1}^{N} \left(min\{p_i^{max}, q_i^{max}\} - max\{p_i^{min}, q_i^{min}\} \right)^2} & \text{, nonoverlapping.} \end{cases} \tag{8.1}$$

Let us now take a look at the incentives behind this distance function. By applying it in selecting neighbors, nodes concentrate on acquiring links to nodes of gradually closer, and eventually overlapping interests. However, no preference is made among different nodes of overlapping interests, making all of them equally likely to be kept or discarded from the VICINITY view during gossiping. This way, a subscriber gets to eventually "see" all nodes with some overlapping interests.

8.6.3. Building Ring Links

The top layer in our architecture (Figure 8.6) is in charge of building ring links. It consists of the VICINITY protocol too, albeit with a small modification: it adopts a view of *variable* length, as opposed to fixed length in the standard version of the protocol. This reflects the variable number of ring links nodes maintain.

It is worth noting that the top layer receives input from both lower layers, as can be seen in Figure 8.6. Indeed, when two nodes gossip in the context of the top layer, each of them merges all its views (i.e., including the ones for the random and overlapping-interest links) into a single container. It subsequently applies the VICINITY selection function (described below) on this container to select the nodes that are eligible as ring neighbors for the other node, and sends them accordingly. Similarly, when receiving links from the other node, it merges them with its current views from all three layers, and applies the selection function again, this time to select ring links for itself.

An essential element for building ring links is the associated VICINITY selection function. The selection function, $S(k, P, \mathcal{D})$, goes through the node descriptors in set $\mathcal{D}$, and selects the ones that qualify as ring links for node P. More specifically, it goes through the nodes in $\mathcal{D}$ in increasing sequence ID order (starting from the ID of P and cycling when reaching the maximal sequence ID) and selects a node only if its subscription intersects P's subscription subspace at some region not yet covered by already selected subscribers. This process is then repeated, but now iterating in decreasing sequence ID order. Note that the argument k, which normally determines the number of nodes returned by S, is completely ignored by this selection function. The pseudocode of the selection function is shown in Figure 8.8.

Another point deserving illumination is the representation of the not-yet-covered subscription subspace of P, and the way we subtract hyper-rectangles from it. It is represented by means of a new type, called *hyperspace*. A hyperspace is, in turn, represented by a collection of hyper-rectangles, constituting its *components*. Subtracting a hyper-rectangle from a hyperspace consists in sequentially subtracting it from each—interesecting—component hyper-rectangle of the hyperspace. A hyper-rectangle intersects a hyperspace, if it intersects *any* of the hyperspace's component hyper-rectangles. The pseudocode for the hyperspace class, along with

```
function S(k,P,𝒟) // VICINITY selection function for ring links
    var uncovered: HyperSpace
    var selected: set of Node init ∅
    for direction in {ascending,descending}
        uncovered ← P.hyperRect
        foreach Q ∈ 𝒟 from P.id by direction
            if Q.hyperRect intersects uncovered then
                uncovered.subtractHyperRect(Q.hyperRect)
                selected ← selected + {Q}
            end if
        end foreach
    end for
    return selected
```

Figure 8.8: The VICINITY selection function for building ring links, in pseudocode. It selects ring links for node P, out of the set of node descriptors $\mathcal{D}$. Argument k, determining the number of nodes that should be returned in the standard version of VICINITY, is not taken into account.

its method for subtracting hyper-rectangles, is presented in Figure 8.9.

Let us, finally, explain how we subtract hyper-rectangles from each other. To subtract hyper-rectangle u from v, we sweep along every dimension, one at a time, extracting from v chunks (smaller hyper-rectangles) that do not intersect with u in that dimension. We stop after going through all dimensions. The outcome of the subtraction is the collection of chunks we have extracted, which essentially constitute a hyperspace, $v-u$. Subtracting another hyper-rectangle is done by subtracting it from each component hyper-rectangle of hyperspace $v-u$, as described in the previous paragraph. The process of hyper-rectangle subtraction is illustrated in a two-dimensional example in Figure 8.10.

8.7. EVALUATION

In this section, we evaluate SUB-2-SUB by simulation under synthetic subscription workloads. We focus on four key issues: overlay construction, hit ratio, propagation speed, and complexity for publisher and subscriber joins. Spam ratio is not considered, as it is fundamentally eliminated by our design.

We built and evaluated SUB-2-SUB on PeerSim, an open source Java simulation framework for P2P protocols [PeerSim].

```
class HyperSpace
    var components: set of HyperRect

    method subtractHyperRect(u)
        foreach v ∈ components
            if u intersects v then
                components ← components − v
                components ← components + v.subtract(u)
            end if
        end foreach
```

Figure 8.9: The HyperSpace class.

8.7.1. Experimental Setup

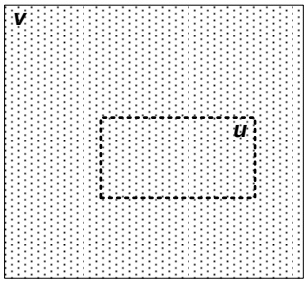
In lack of real-world subscription datasets, we generated synthetic ones as follows. N-attribute subscriptions were represented as N ranges in $[0 \ldots 1]$, one for each attribute. A range's center was chosen following the respective attribute's *interest distribution*. A range's width was determined by the attribute's *width distribution*.

In each experiment we applied the same interest distribution to all N attributes: either *uniform* or *power law*. The former represents a natural unbiased workload. The latter, known as *Zipf*, is admitted to be a good approximation of interest popularity and results in subscription sets closer to expected social behavior, exhibiting popular and rare values. It also results in more interesting experiments, as there is higher overlap around the "center" of the interest space, and lower towards its edges, resulting in rings of various lengths. The width distribution was fixed to *power law* centered at 0, with $\alpha = 4$, to account for both wide range and (nearly) exact subscriptions.

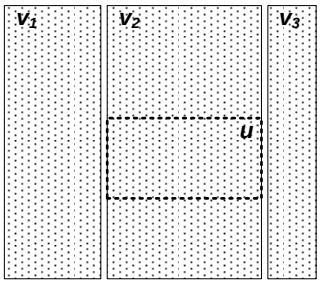
In evaluating SUB-2-SUB we considered schemes of up to five attributes. The number of subscribers was fixed to 10,000 for all experiments.

We tested each experiment's effectiveness by observing the dissemination of 10,000 test events. Test events were picked at random, ensuring each one had at least two matching subscribers, to make dissemination meaningful.

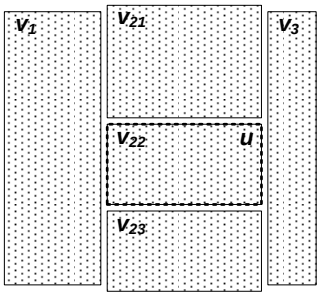
Finally, with respect to the RINGCAST dissemination algorithm, nodes forwarded events to their *two* ring neighbors, and to *one* matching subscriber (if any) chosen from their overlapping-interest neighbors. That is, events were disseminated with a fanout of three.



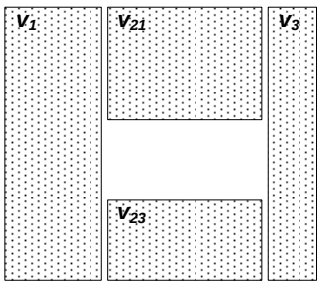
(a) We want to subtract rectangle u (small, with dashed outline) from rectangle v (large, shaded).



(b) We first sweep along the x dimension, detaching parts of v not intersecting with u. Rectangle v is split in rectangles v_1, v_2, and v_3. Note that the overlap with u is isolated in a single new rectangle, v_2, which matches u in the x dimension.



(c) We, then, concentrate on v_2, the only rectangle that still intersects u. We sweep along the y dimension, splitting v_2 in rectangles v_{21}, v_{22}, and v_{23}. The overlap with u has now been isolated to rectangle v_{22}. Note that at this point we have looped through *all* dimensions. Consequently, the rectangle still intersecting u (v_{22}) constitutes an exact overlap of u. We, therefore, remove it altogether.



(d) The set of rectangles we are left with, constitute the outcome of the subtraction $v - u$.

Figure 8.10: Example of rectangle removal in a two-dimensional space. First sweeping along the x and then along the y dimension. Generally, in an N-dimensional space, we would carry on the same process for all N dimensions.

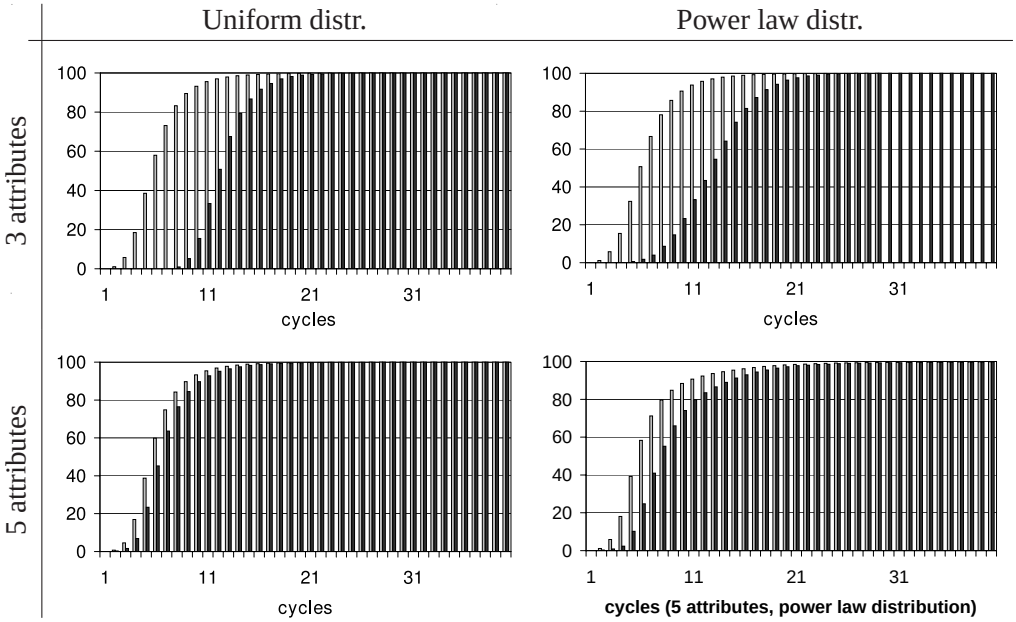


Figure 8.11: Construction of the rings in time. Light bars show the percentage of ring links already in place. Dark bars show the percentage of rings that are complete. 10K nodes.

8.7.2. Jump-starting SUB-2-SUB

We first test the efficiency of our algorithm in jump-starting a SUB-2-SUB overlay from scratch. Nodes started gossiping at the same time, having been initiated with a single random link in their CYCLON views, ensuring the overlay formed a connected graph. We recorded the topology evolution by keeping statistics over the ring links associated with each of the 10,000 test events.

Figure 8.11 shows the evolution of ring construction per cycle, for four experiments. We can see that after 40 cycles, all rings are fully set up.

Having confirmed that rings are constructed in a small number of cycles in all cases, the remaining evaluation focuses on a single experiment, namely the one with three attributes and power law interest distribution. This experiment is the most interesting one for testing event dissemination and propagation speed, as the rings it involves range from very small (2 subscribers) to quite large (246 subscribers). In five-attribute schemes, rings are trivially short (2-3 subscribers) due to the very large subscription space. The distribution of ring lengths for all experiments is depicted in Figure 8.12.

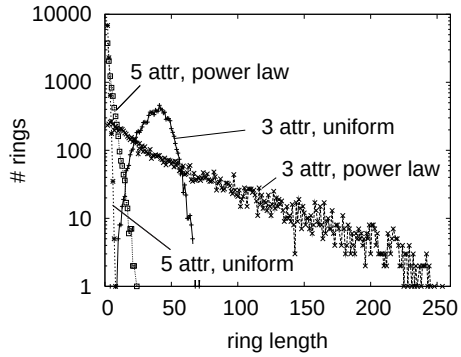


Figure 8.12: Distribution of ring lengths.

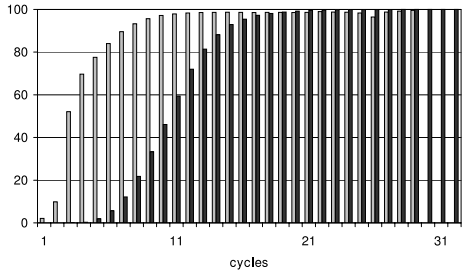


Figure 8.13: Event dissemination. Light bars show the hit ratio for *non-complete* disseminations. Dark bars show the percentage of disseminations that were complete (events delivered to all their matching subscribers). 10K nodes; 3 attributes; power law interest distribution.

8.7.3. Event Dissemination

As mentioned earlier, events are disseminated by means of the RINGCAST dissemination algorithm (Section 6.5.1). Nodes forward events to their *two* ring neighbors, and to *one* matching subscriber picked randomly from the view of overlapping-interest neighbors.

Figure 8.13 presents the performance of SUB-2-SUB with respect to event dissemination. It is worth noting that, by comparison to Figure 8.11(upper-right), complete dissemination is achieved even *before* all rings are in place. This comes as a result of (also) forwarding across random overlapping-interest links.

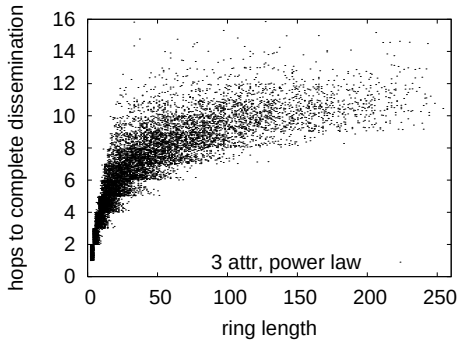


Figure 8.14: Hops to complete event dissemination, as a function of the number of matching subscribers (ring length). 10K nodes; 3 attributes; power law interest distribution.

8.7.4. Propagation Speed

We now examine the speed, in terms of the number of hops at which events spread. We are specifically interested in the number of hops for *complete dissemination*, that is, the number of hops elapsed from the moment a publisher delivers an event to some matching subscriber, until the event reaches the last one of them.

Figure 8.14(b) shows the number of dissemination hops as a function of the number of subscribers matching the respective events. Clearly, the number of hops increases with the number of matching subscribers. However, as a result of short-cutting the rings in disseminating events, this relation is of logarithmic fashion.

8.7.5. Single Node Joins

Jump-starting SUB-2-SUB comprises a worst-case scenario, as the whole overlay starts from a completely non-clustered state. We now take a look at the other end of the spectrum, measuring the number of cycles it takes a single node to join an already converged overlay.

Starting from the converged state of our experiments, each subscriber was individually wiped out of the network. That is, it was removed from the network along with all links to it. Then it was let to rejoin, and the number of cycles to fully rejoin was recorded. Joining the network involves (a) building appropriate ring and overlapping-interest links to other nodes, and (b) becoming known by other nodes. A node was considered to have fully rejoined the overlay when *all* its ring links (i.e., outgoing and incoming ones) were in place. The number of cycles it took to rejoin the overlay gives the distribution shown in Figure 8.15.

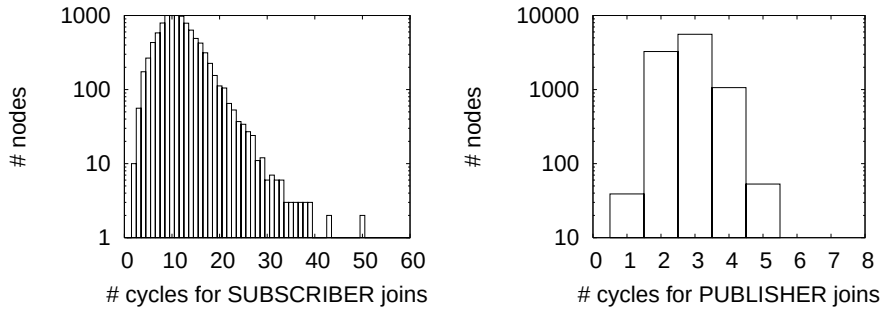


Figure 8.15: Distribution of cycles it takes subscribers and publishers to join.

For publishers, on the other hand, joining is a simpler task, as they are interested only in reaching *any* matching subscriber, independently of its sequence IDs. Figure 8.15 also shows the distribution of cycles it takes publishers to join, starting from a random node. It is worth noting that all 10,000 publishers we tested joined in five or less cycles. This is important, as a publisher can safely assume there is no subscriber matching its event(s) after a low threshold of cycles (i.e., in the order of 10 or 20).

8.8. RELATED WORK AND CONCLUSIONS

Scalability of peer-to-peer systems makes them natural candidates to implement large-scale publish/subscribe systems. In this chapter, we presented the design and evaluation of SUB-2-SUB, a scalable, self-organizing peer-to-peer approach for content-based publish/subscribe in collaborative environments. SUB-2-SUB deploys an unstructured overlay where subscribers are clustered in efficient dissemination structures, based on shared interests. To the best of our knowledge, SUB-2-SUB is the first attempt to build publish/subscribe overlays using epidemic-based algorithms, thus exploiting their ability to handle dynamic environments.

Unlike SUB-2-SUB, previous peer-to-peer approaches for content-based publish/subscribe have mainly focused on structured overlays. Among them, Meghdoot [Gupta et al. 2004] uses an extension of the CAN DHT [Ratnasamy et al. 2001b]. It maps subscriptions to a $2 \times k$-Euclidean space, where k is the number of attributes. Each attribute is represented by two dimensions, corresponding to its minimum and maximum allowed values respectively, allowing for range subscriptions. Unlike SUB-2-SUB's autonomous and self-contained operation, Meghdoot

employs a separate set of dedicated nodes for storing subscriptions and disseminating events. Meghdoot deals with sparse interest distribution with CAN zone replication, which may be computationally expensive to maintain in highly populated parts of the space. Terpstra et al. [Terpstra et al. 2003] proposed to leverage the properties of the Chord DHT [Stoica et al. 2003] to implement an event filtering system. Distributed nodes are dynamic brokers organized in a graph. They use subscription merging and covering to provide scalability, acting similarly to traditional content-based filtering systems based on a dedicated set of brokers. However, such a set of brokers may be hard to maintain efficiently in highly dynamic environments. Finally, Costa et al. proposed to use epidemic-based algorithms to enhance reliability of existing publish/subscribe systems [Costa et al. 2003].

We conclude that SUB-2-SUB is an appealing alternative to existing solutions for content-based publish/subscribe. It offers a scalable, autonomous, and self-organizing system, combining the resilience of epidemic-based overlays with the expressiveness of the content-based model.

CHAPTER 9

Conclusions

In this dissertation we explored gossiping protocols for self-organization, and demonstrated their power in a number of different settings. This final chapter presents our observations from the research conducted, grouped in three categories. First, we discuss conclusions regarding randomized overlays, second, conclusions regarding structured overlays, and, third, conclusions from our gossip-based applications. Finally, in Section 9.4 we discuss future directions for our research.

9.1. RANDOMIZED OVERLAYS

We present our high-level observations first, followed by detailed conclusions afterwards.

9.1.1. High-level Observations

In Chapters 2 and 3 we explored a class of gossiping protocols in which peers gossip membership information in a more or less random way. Our main conclusion from these chapters is that such protocols constitute an excellent choice for the decentralized construction of unstructured overlay networks that share a lot of properties with random graphs.

This class of gossiping protocols is particularly appealing to massive-scale distributed applications. The reasons can be found in our high-level observations from Chapters 2 and 3:

Decentralization Gossiping protocols are by nature fully decentralized. They operate in a completely distributed way, by each node maintaining only a very small, partial view of the network, and acting based on local decisions exclusively.

Self-organization and Adaptivity The gossiping protocols we explored in Chapters 2 and 3 are self-organizing. The behavior of nodes relies entirely on local decisions, and the communication graph (i.e., the overlay) is defined exclusively by local knowledge of the nodes, yet the system converges as a whole towards certain global properties. In the case of network changes, gossiping protocols have shown to be very adaptive. It is worth noting that in our experiments, overlays converge to the same global properties irrespectively of the bootstrap method used. Self-organization renders these protocols attractive for internet-scale systems, for which explicit control is either infeasible or, at least, very expensive.

No need for administration As a result of their self-organizing nature, these gossiping protocols are also administration-free. Global properties are not imposed through any central—and possibly expensive—administration and control, but come "for free" as emergent properties of gossip-based overlays.

Robustness We observed that randomized overlays formed by our gossiping protocols demonstrate remarkable resilience to large-scale failures and high node churn, even when each node maintains only a couple of dozen outgoing links. This property is particularly desirable for massive-scale distributed applications, as it keeps all nodes connected in a single cluster despite failures and node churn, prohibiting (irrecoverable) partitioning of the set of nodes. Gossiping protocols are, therefore, valuable as a fundamental substrate taking care of membership management.

Self-healing behavior In addition to sustaining large-scale failures and high churn, the gossiping protocols we explored also demonstrate strong self-healing behavior. That is, after a—possibly severe—network change disturbs the overlay, nodes swiftly reorganize themselves and converge as of new to global properties. In the case of continuous churn, the overlay engages in a continuous self-healing process, maintaining its global properties close (depending on the churn rate) to the ones in a stable network.

Scalability The gossiping protocols we explored in Chapters 2 and 3 are inherently scalable. Irrespectively of the network size, each node maintains only a small, fixed sized, partial view of the network, and exchanges messages with other nodes at a fixed rate. The size of the network does not have any effect on the memory, processing, or network load of nodes. Essentially, gossiping protocols are infinitely scalable with respect to the load of nodes.

Local randomness We concluded that the protocols explored in Chapter 3 constitute a source of "random link generator", providing each node with a

stream of links to randomly chosen other nodes, in a way similar to a random number generator providing a program with random numbers. This is why we describe them collectively as the PEER SAMPLING SERVICE. This feature is important to a number of applications, some of which we explored in Part II of this dissertation.

Simplicity Last but not least, gossiping protocols are very simple in nature. This is particularly important for systems of so large scale, where things can easily get out of hand.

9.1.2. Detailed Observations

Apart from these high-level observations, by conducting the research presented in Chapters 2 and 3 we came across a number of specific, fine-grained questions. Delving deeper into the details, we pinpointed a number of design issues that need to be addressed, the most important of which are laid out here:

View length Our protocols define a very small view length, consisting of only a couple of dozen links out of possibly hundreds of thousand. Definitely, the reason for imposing such a short view length is not to save memory, a very cheap commodity nowadays. In fact, some applications, principally the ones related to content searching, would benefit from a larger view length. However, there is a tradeoff between the chosen view length and the validity of links in dynamic environments. We observed that the larger the view length, the higher the percentage of dead links in node views. Small views allow nodes to cycle through their neighbors faster, removing dead links in a more timely manner.

Peer selection Peer selection addresses the following question: "Which of its neighbors should a node select to initiate gossiping with?" Should it prefer a new (*head peer selection*), an old (*tail peer selection*), or a random (*random peer selection*) link from its view? Head peer selection is a bad idea: a new link points to a recently contacted neighbor, therefore it has little new useful information to offer. Especially selecting *the* newest link is the worst option, as it points to the last contacted neighbor, which means that a node keeps gossiping with that same neighbor continuously. We have concluded that peer selection should be set either to tail or random. In fact, these two options have very comparable effects, with tail peer selection slightly outperforming random in most cases.

Directionality of communication When node P initiates gossiping with node Q, who should send links to the other one? It can be either P (*push-only*), Q

(*pull-only*), or both *P* and *Q* (*push-pull*). Our experimental analysis showed that the only policy that results in adaptive overlays is push-pull, that is, communication should be bidirectional. Using the pull-only (push-only) policy, a node having very low indegree (outdegree) has low chances of advertising itself to other nodes, and, therefore, higher chances to get disconnected. Such scenarios are common. For instance, a node may find itself having a low indegree because it just joined, or a low outdegree because some of its neighbors died.

Garbage collection policy After exchanging views, a node typically finds itself with more links than its view size, so it should discard some of them. Which ones should it discard? The newest links, the oldest ones, or a random sample? And how many? In our generic gossiping framework of Chapter 3 we are modeling this with the *healing parameter H*. We conclude that discarding the *H* oldest links after each gossip exchange helps the network get rid of invalid links (links pointing at disconnected nodes) soon. The higher the value of *H*, the faster dead links are removed, therefore, the better the self-healing behavior of the network. However, there is a tradeoff here. The higher the value of *H* the more skewed the indegree distribution becomes, which, in turn, is a disadvantage with respect to balanced load distribution.

Copy links or swap links? This question addresses the way nodes exchange views. When gossiping, what should a node do with the links it sends to its gossiping counterpart? Should it still keep them (*copy* links), discard them (*swap* links), or something in-between? Swapping links results in lower clustering, resembling random graphs. This, in turn, has three advantages. First, it provides nodes with highly uncorrelated neighbors, leading to better local and global randomness. Second, it increases the resilience of the network to catastrophic failures. Third, it results in more concentrated indegree distributions, having a positive effect on load distribution. On the other hand, copying—as opposed to swapping—links results in higher clustering, forming overlays that resemble small worlds. The load is less evenly distributed across nodes. However, there is an abundance of links after each gossip exchange, which permits us to increase the healing parameter *H*, making the network more resilient to high churn.

Apparently there is no single best solution for all design issues. Our research depicts the tradeoffs imposed by different policies. Depending on the priorities in a particular network, the designer can handle the respective tradeoffs accordingly.

9.2. STRUCTURED OVERLAYS

In Chapter 4 we presented the framework consisting of VICINITY and the PEER SAMPLING SERVICE, that allows networks to self-organize into a given target topology. The target topology is expressed by means of a selection function that determines which neighbors are optimal for every node.

The overall conclusion from this part of our research is that our topology construction framework is flexible enough to build a wide variety of topologies. In particular, the framework excels for topologies demonstrating high correlation of nodes that share common neighbors, embodying the principle "the friend of my friend is also my friend". That is, topologies where two nodes sharing a common neighbor have a high chance of being direct neighbors as well. For networks with lower correlation between nodes, our framework still manages to construct the target topology, although not as efficiently, as the discovery of appropriate neighbors relies on random encounters.

The major conclusion regarding topology construction is that it is crucial to combine two layers. One layer, namely the VICINITY protocol, strives at locating neighbors of close proximity in the target topology. The importance of this layer is straightforward: it establishes the target topology links. It continuously improves these links by probing neighbors for links to new, even "closer" neighbors, better approximating the ideal neighbors in the target topology.

The other layer, in our case implemented by the PEER SAMPLING SERVICE, keeps nodes connected in a randomized overlay. This is crucial for two reasons. First, it prohibits partitioning of the set of nodes, which would occur in no time if nodes concentrated only on their "close" neighbors. In a partitioned overlay, nodes that happen to join in the "wrong" partition have no way to traverse the network to discover their appropriate neighbors. Second, it provides nodes with random links all over the network. This is vital for newly joined nodes to reach fast their topological vicinity in an already converged overlay, rather than traversing the network in numerous small steps. Random links serve in a way similar to long-range links in small-world networks.

Concluding, the principal advantage of our topology construction framework is that it is simple and generic. Its simplicity stems from its self-organizing nature. Each node autonomously carried out a sequence of simple steps and operates based on local decisions exclusively, eliminating the need of any deployed infrastructure. Its generic character is derived from its flexibility to form a wide variety of target topologies, given the appropriate selection function. It constitutes, therefore, a very convenient—yet efficient—way to form arbitrary topologies, either for prototyping or for the working version of massive-scale decentralized systems. It is particularly suitable for systems where the exact final topology is not known be-

fore deployment, as it permits fine tuning, or even radical changes of the topology at run-time.

9.3. APPLICATIONS

The second part of this dissertation dealt with applications based on the protocols presented in the first part. The applications included routing table management (Chapter 5), information dissemination (Chapter 6), semantic overlay network construction (Chapter 7), and a publish/subscribe system (Chapter 8). All four applications employed gossiping protocols to organize the nodes in certain topologies in a fully decentralized and autonomous manner. With the exception of the first application, routing table management, which constitutes our first efforts towards gossip-based structuring, all other applications are based on the combination of VICINITY with the PEER SAMPLING SERVICE.

Chapter 5 demonstrated the potential of the PEER SAMPLING SERVICE in forming a structured overlay. As mentioned already, this was our first system to derive structure from randomness. The main conclusion drawn from this chapter is that randomized gossiping protocols can be harnessed to create structure, in a totally decentralized, self-organizing fashion. We also observed that the robustness and self-healing properties common in gossip protocols are retained in building a structure. Although the research presented in this chapter is now outdated by the VICINITY protocol, the conclusion it offered was valuable in continuing our work towards that direction, that is, gossip-based topology construction.

Chapter 6 employed VICINITY and the PEER SAMPLING SERVICE for information dissemination. Gossiping protocols have been applied previously for *probabilistic* information dissemination [Birman et al. 1999; Kermarrec et al. 2003]. Our research extends previous work by complementing probabilistic with *deterministic* dissemination. The chief conclusion of this chapter is that the combination of probabilistic with deterministic—together called *hybrid*—dissemination constitutes a very attractive framework for broadcasting messages efficiently and inexpensively, in a completely decentralized way. Probabilistic dissemination (i.e., forwarding a message at random) is achieved through random links established by the PEER SAMPLING SERVICE, and results in fast diffusion of messages all over the network. Deterministic dissemination (i.e., forwarding a message across well defined links) is carried out over a structured overlay formed by VICINITY, and ensures that every message reaches all nodes.

In Chapter 7 we employ gossiping to enhance distributed content-based searching. We employ the topology construction framework of VICINITY and the PEER SAMPLING SERVICE to build a semantic overlay network linking nodes of related

interests. When node issues a query, it first asks a few of its semantically closest peers, and if the query is not satisfied it resorts to the underlying search mechanism (such as flooding, random-walks, super-peer indexing, etc.). Our simulations, based on real-world traces, show that about 30% of the queries are satisfied this way.

The idea of building a semantic overlay network to enhance content-based queries is not new. However, all previous systems depended on heuristics to guess which peers out of the ones that recently served a node might be useful in serving the same node again. As a consequence, such systems start being effective only after a node has issued a number of queries. Also, they (implicitly) assume a static network, and no changes in user interests, otherwise links collected in the present are likely to be useless in the near future.

Unlike these systems, our protocol is—to the best of our knowledge—the first system that handles node churn and dynamic user interest. It is a highly adaptive, fast converging, yet lightweight epidemic-style solution, that builds and maintains semantic overlay networks *proactively*.

Finally, Chapter 8 introduces SUB-2-SUB, a fully decentralized, autonomous solution for attribute-based publish/subscribe supporting exact and range queries. SUB-2-SUB is based on a self-organizing overlay, that groups nodes of overlapping subscriptions together, so that all nodes interested in a given event can disseminate it among themselves in a completely autonomous way.

Our major conclusion from this chapter is that building a collaborative publish/subscribe system can benefit from the scalability and adaptivity inherent in P2P systems. In addition, SUB-2-SUB demonstrates the power of gossiping protocols in building intricate structures capturing relations between peers. Finally, we conclude that unstructured overlays can be very promising for publish/subscribe systems, a research area currently dominated by work on centralized, hierarchical, and structured P2P systems.

9.4. FUTURE DIRECTIONS

This dissertation investigated gossiping protocols and showed their power in a number of different applications. In effect, our work has opened some new tracks in the area of gossip-based self-organization for massive scale decentralized systems. Undoubtedly, this is a large area that cannot be fully covered in a single dissertation. There are a number of directions in which our research can be complemented and extended.

An important next step in our research is the replacement of periodic view exchanges with a reactive exchange mechanism. With such a protocol, nodes

would locally determine and dynamically adjust their gossiping frequency based on observed system dynamics, such as node churn, failure rates, and information dissemination needs. We envisage that this replacement will lead to a better utilization of network resources, and incur only minimal costs for detecting failed nodes and keeping membership information up to date. A first step towards reactive adaptation of the gossiping frequency can be found in our recent work on a gossip-based clock synchronization protocol [Iwanicki et al. 2006].

Expanding on adaptation, it will be interesting to build an adaptive version of the PEER SAMPLING SERVICE, that dynamically adjusts the *swapping parameter* (*S*) and the *healing parameter* (*H*), to optimize system behavior given the current network conditions. Such a system would switch to a more fault tolerant mode of operation in the face of high churn, and would gradually shift to an overlay with more balanced degree distribution and lower clustering as the system stabilizes.

To further limit overlay maintenance costs, network proximity should be taken into account. Network proximity estimation can be provided through a decentralized network coordination system such as Skole [Szymaniak et al. 2004], or Vivaldi [Dabek et al. 2004]. In a proximity-aware gossiping protocol, view exchanges among nearby peers should be favored—that is, be more frequent—than among distant ones. This, however, would have an impact on the connectivity and robustness properties of the emerged overlays. Determining the right balance between proximity-aware gossiping and desired overlay robustness constitutes a challenging research topic.

Overlay management is not the only service proximity-aware gossiping can be useful to. Applications can benefit from proximity-aware overlays too, lowering communication costs and enhancing application efficiency. More specifically, input from network coordination systems, such as Skole and Vivaldi, can be incorporated in the VICINITY selection function of an overlay-based application, resulting in proximity-aware overlays. The implications of such overlays, as opposed to a proximity-unaware ones, in efficient broadcasting of information are clear.

Another research direction worth following is the class of gossiping protocols featuring variable view lengths. Our gossiping protocols (with the exception of the top layer in SUB-2-SUB) impose a fixed length on node views. The underlying motivation is to keep the number of neighbors per node low and controllable, so that dead links are discarded promptly. An alternative is to impose a maximum lifetime for each link and discard it after it expires. This alternative is employed by the Kelips protocol [Gupta et al. 2003]. It is not known, however, how overlay properties, such as the degree distribution, the percentage of dead links, clustering, etc., are affected by node churn and failures. This family of gossiping protocols certainly deserves further research.

All research presented in this dissertation is experimental. Gossiping protocols exhibit particularly chaotic behavior that is hard to model analytically. Theoretical analysis has not been in the focus of this work. Nevertheless, it is interesting to mathematically analyze the behavior of gossiping protocols, and formally validate their properties. Our recent work in [Bonnet et al. 2006], analyzing a simplified version of CYCLON without the age field, constitutes a first attempt in that direction.

With respect to information dissemination, this dissertation has focused on push-based approaches. This is the case in both protocols dealing with information dissemination, namely the RINGCAST protocol (Chapter 6) and the SUB-2-SUB publish/subscribe system (Chapter 8). Combining push-based with pull-based dissemination should yield a significant improvement in dissemination efficiency. However, the details of such an approach have been deferred to future work.

Some applications may require higher reliability and performance guarantees. Gossiping protocols, and P2P systems in general, offer best-effort solutions. An interesting direction of research would be in designing hybrid systems, that consist of a combination of centralized and P2P components. In such a setting, centralized components will provide hard guarantees on reliability, while P2P components will add scalability to the system. Napster was a very successful example of such a hybrid system [Napster]. Its eventual failure was due solely to copyright violation issues.

Security has not been discussed in this dissertation. Nevertheless, security is a crucial aspect for any communication framework today. In the case of gossiping protocols, security comes in two different flavors. First, security at the overlay maintenance level. Malicious peers may deliberately try to cause damage to the overlay, e.g., by reporting fake neighbors, discarding legitimate neighbors, achieving a very high degree by gossiping at a very fast pace, etc. Mechanisms should be in place to shield gossip-based overlays from such malicious peers. Second, security at the application level. Malicious peers may refuse to forward messages, respond to other peers, and generally contribute to a P2P application. Security at this level is application specific, and should be dealt with by the respective applications. We believe that security is the main factor still hindering the massive deployment of peer-to-peer systems for critical applications nowadays. Its importance, therefore, is key to the commercial success of peer-to-peer protocols.

In a broader sense, our vision for the future is to exploit the PEER SAMPLING SERVICE, CYCLON, VICINITY, and possibly new gossiping protocols for a multitude of diverse peer-to-peer applications. We envision the protocols introduced in this dissertation as forming a basic background process, supporting, organizing, and managing overlay networks of *any* scale across the Internet in a fully

decentralized way.

SAMENVATTING

Epidemisch Gebaseerde Zelforganisatie in Peer-to-Peer Systemen

ACHTERGROND

Met rede kan gesteld worden dat we in het tijdperk van de communicatie revolutie leven. Communicatie is nog nooit zo aanwezig, massaal, snel en goedkoop geweest. Computernetwerken in het algemeen en het Internet in het bijzonder spelen een katalyserende rol in moderne samenlevingen. De dagen dat het Internet slechts een onderzoeksgereedschap was, of een communicatiemiddel voor slechts academische en militaire instellingen, liggen ver achter ons. Toegang tot het Internet heeft zich verbazingwekkend snel ontwikkeld voor een steeds breder wordend publiek.

Vroege Internet diensten werden ontwikkeld rondom twee principes: het zogeheten *client/server model* en centralisatie. Het client/server model maakt een expliciet onderscheid tussen computers ("knopen") die een dienst leveren (de zogeheten *servers*) en computers die diensten afnemen (de *clients*). Centralisatie behelst dat een dienst op slechts één centraal punt aangeboden wordt, veelal bovendien door slechts één computer.

De groei van het Internet heeft laten zien dat een gecentraliseerde architectuur niet langer houdbaar is voor een groot aantal diensten zodra het aantal afnemers (met andere woorden, clients) dramatisch toeneemt. Ten gevolge hiervan is veel onderzoek ontstaan naar gedistribueerde systemen, waarbij de kerngedachte is om het aanbod van een dienst te verspreiden over meerdere, samenwerkende computers. Een dergelijke verspreiding heeft potentieel veel voordelen, waaronder een grotere tolerantie tegen falende computers, verhoogde geaggregeerde capaciteit,

geografische verspreiding (met als gevolg dat een dienst dichterbij clients komt), enz. Clients blijven echter ook bij verspreiding van de servers relatief passief: het blijven slechts afnemers.

In de afgelopen jaren is de toegang tot het Internet sterk verbeterd, zowel in quantitatieve als qualitatieve zin, en is ook de reken- en opslagkracht van PCs indrukwekkend toegenomen. Het effect is dat de traditioneel relatief zwakke client computers nu zeer krachtige en goed verbonden apparaten zijn geworden. Een gevolg daarvan is dat in veel gevallen hun relatief passieve rol langzaam maar zeker veranderde naar die van actieve entititeiten in een massal gedistribueerd systeem. In dit nieuwe paradigma, ook wel *peer-to-peer computing* genoemd is het verschil tussen aanbieders en afnemers van diensten verdwenen. In plaats daarvan zijn knopen gelijkwaardig en werken samen om een specifieke dienst uit te voeren.

Het *peer-to-peer* (P2P) model brengt een aantal uitdagingen met zich mee. De belangrijkste daarvan is de potentiele schaal waarin P2P systemen dienen te opereren. De schaal van deze systemen is gegroeid van groot naar massaal, waarbij in sommige gevallen miljoenen computers verspreid over het hele Internet samenwerken, zonder dat er sprake is van enige centralisatie. Echter, met dergelijke groottes hebben we ook te maken met een hoge mate van instabiliteit doordat dergelijke systemen continu aan verandering onderhevig zijn. Knopen in een P2P systeem laten geen gegarandeerd patroon van samenwerking zien. In plaats daarvan zien we een komen en gaan van knopen en vallen er constant knopen en verbindingen uit. Daarbij komt dat de knopen in een groot P2P systeem erg kunnen verschillen met betrekking tot hun hardware architectuur, samengestelde software, en feitelijke kracht zoals uitgedrukt in o.a. de capaciteit van processor, werkgeheugen, permanente opslag en netwerkverbinding.

Dit geheel maakt de administratie en het beheer van P2P systemen extra kritish. De massale omvang in combinatie met de hoge mate van dynamiek in de samenstelling van een P2P systeem maken expliciete en centrale besturing vrijwel onmogelijk, of op z'n minst zeer lastig te realiseren. Gecentraliseerde oplossingen zijn zeer gelimiteerd als het aankomt om snel en effectief bij te houden welke computers deel uitmaken van een systeem. Deze schaalbaarheidsproblemen kunnen enisgzins verminderd worden door bijvoorbeeld hiërarchische oplossingen toe te passen. Echter, dergelijke oplossingen zijn dikwijls complex, introduceren een forse administratieve last en zijn afhankelijk van de beschikbaarheid van een klein groepje cruciale computers. Deze laatste afhankelijkheid kan aangepakt worden door het toepassen van replicatie, maar dergelijke oplossingen verhogen complexiteit en feitelijke beheersbaarheid van de besturing op zich. Daarbij komt dat de omvang van de benodigde besturing sterk afhankelijk is van de omvang van het systeem dat bestuurd dient te worden. Een schatting geven van die omvang is

moeilijk, zoniet dikwijls onmogelijk.

De toename van het aantal apparaten dat verbonden is met het Internet en de verdere verspreiding van netwerktechnologie laat zien dat de trend naar grote systemen van samewerkende computers alleen maar zal toenemen. Ook in ons onderzoek anticiperen we een sterke toename van zeer omvangrijke en gedistribueerde toepassingen. Derhalve zijn we afgestapt van deterministische en expliciete systeembesturing en zijn methoden gaan exploreren voor autonoom management in de vorm van zelfbestuur en -organisatie.

EPIDEMISCHE PROTOCOLLEN

Het doel van dit proefschrift is het verkennen van de uitdagingen in het besturen van zeer grote gedistribueerde systemen, het introduceren van nieuwe protocollen en het onderzoeken van hun effectiviteit in huidige en toekomstige toepassingen. Het was hierbij niet de bedoeling om de communicatiemodellen die ten grondslag liggen aan het Internet te veranderen daar vele toepassingen reeds uitstekend werken. In plaats daarvan hebben we gezocht naar alternatieve oplossingen waar huidige benaderingen te duur zijn, niet schaalbaar zijn, teveel administratieve last vergen, of gewoon niet toepasbaar zijn.

In het bijzonder heeft het onderzoek zich gericht op het toepassen van zogeheten *epidemische protocollen* voor de ontwikkeling en onderhoud van P2P systemen. Het basisprincipe van dergelijke protocollen is dat elke knoop regelmatig een willekeurig andere knoop selecteert om gegevens mee uit te wisselen. Op deze wijze zal data zich geleidelijk over alle knopen verspreiden totdat elke knoop dezelfde informatie heeft. Een probleem dat hierbij optreedt is dat er een willekeurige selectie uit alle andere knopen moet plaatsvinden. Echter, een dergelijke selectie is ronduit onmogelijk indien we te maken hebben met zeer grote systemen, waar globaal bekende *accurate* informatie over de huidige verzameling van knopen schier onmogelijk is.

In ons onderzoek hebben we gekeken naar epidemische protocollen die op twee manieren afweken van het oorspronkelijke basisprincipe. Ten eerste hebben we de aanname van globale kennis omtrent de verzameling van knopen laten vallen. In onze benadering hanteert elke knoop een kleine, maar veranderende lijst van andere knopen, ook wel zijn buren geheten. Deze lijsten brengen met zich mee dat een P2P systeem georganiseerd is als een *netwerk*. Ten tweede behelst de data die tussen twee knopen uitgewisseld wordt in elk geval referenties naar de respectievelijke buren. Met andere woorden, knopen informeren elkaar over hun buren. Na een uitwisseling worden de respectievelijke lijsten (gedeeltelijk) ververst met referenties naar nieuwe buren. Door exact vast te leggen welke refer-

enties uitgewisseld worden, en welke referenties in de lijst van buren opgenomen worden, blijkt het mogelijk om op volledige gedecentraliseerde wijze de *topologie* van het P2P netwerk vast te kunnen leggen.

ONZE PROTOCOLLEN

Een aantal ontwerpkriteria bepalen de eigenschappen van de resulterende P2P netwerken. Wordt de lijst van buren min of meer willekeurig samengesteld, dan blijkt een zogeheten *random netwerk* het resultaat te zijn. Wordt een ordening op de selectie van referenties gehanteerd, dan kunnen specifieke topologieën geconstrueerd worden. Deze keuzemogelijkheid heeft tot de volgende twee nieuwe protocollen geleid.

Cyclon In het Cyclon protocol *ruilen* twee knopen referenties met buren. Dit betekent dat de topologie van het P2P netwerk continu verandert. Echter, het blijkt dat macroscopische eigenschappen convergeren. Zo hebben we experimenteel aangetoond dat de gemiddelde afstand tussen twee knopen, alsook de gemiddelde clusterings coëfficient convergeren naar de waarden die men ziet bij random netwerken. Dat betekent dat op elk moment de lijst van buren feitelijk overeenkomt met een *willekeurige* selectie van knopen uit het P2P netwerk.

In het bijzonder blijkt dat Cyclon niet alleen overeenkomsten heeft met random netwerken, maar dat het bovendien uitstekend bestand is tegen snelle veranderingen in de samenstelling van P2P systeem, of deze nou opzettelijk tot stand zijn gekomen, of omdat er fouten zijn opgetreden. Forse veranderingen tasten doorgaans de toplogie van het netwerk aan, maar Cyclon blijkt hierbij altijd zeer snel te convergeren naar een stabiele situatie. Daarbij komt dat elke knoop doorgaans bij evenveel andere knopen bekend is, wat vervolgens weer leidt tot een evenwichtige balancering van het uit te voeren werk.

Cyclon maakte onderdeel uit van een uitgebreide studie naar de effecten van ontwerpbeslissingen voor epidemische protocollen die uitgaan van alleen lokale informatie. Hoewel de functionele gedragingen van de protocollen grote overeenkomsten laten zien, blijkt dat extra-funtionele eigenschappen, zoals balancering van werk, robuustheid, fouttolerantie en zelforganisatie gevoelig te zijn voor de wijze waarop lijsten van buren uiteindelijk samengesteld worden na een epidemische uitwisseling.

Vicinity In het Vicinity protocol besluit een knoop alleen die verkregen referenties te behouden die aan een bepaald voorkeurskriterium voldoen. Een

knoop zal op deze wijze geleidelijk buren vervangen door buren die steeds meer zijn voorkeur hebben, om uiteindelijk met de optimale buren te eindigen. Op deze wijze zal het P2P systeem zichzelf organiseren naar een specifieke topologie die volledig bepaald wordt door de selectiefunctie voor preferente buren. Het grootste voordeel van Vicinity is haar generieke karakter: door slechts de selectiefunctie te veranderen kan op eenvoudige wijze een andere topologie ontstaan. Ook blijkt dat *verandering* van de selectiefunctie al snel tot convergente leidt naar de nieuw beoogde organisatie.

Vicinity kan echter niet garanderen dat het P2P netwerk samenhangend blijft. Derhalve wordt het altijd in combinatie met Cyclon gebruikt. Deze combinatie heeft tevens het voordeel dat vergelegen buren uiteindelijk toch ontdekt worden en derhalve geselecteerd kunnen worden.

Deze twee protocollen zijn gebruikt voor ontwerp en evaluatie van vier verschillende applicaties: een systeem voor het onderhouden van routeringstabellen in zogeheten DHT netwerken; een raamwerk voor efficiënte informatieverspreiding; de constructie van semantische netwerken voor gedecentraliseerd zoeken; en tenslotte een collaboratief *publish-subscribe* systeem. De toepassing van Cyclon en Vicinity voor deze vier applicaties is telkens zeer verschillend en illustreert hun versatiliteit.

GEVOLGTREKKINGEN

Het toepassen van Cyclon en Vicinity leverde een aantal inzichten op. Ten eerste bleek uit onze experimenten dat de protocollen inherent schaalbaar waren. Schaalbaarheid is het gevolg van het feit dat elke knoop een vast aantal operaties uitvoert, op een tevoren bepaalde frequentie, en onafhankelijk van de grootte van het P2P systeem. Ten tweede bleek dat onze protocollen zich zeer goed konden aanpassen aan gewijzigde situaties in de samenstelling van het P2P systeem. De combinatie van schaalbaarheid met deze flexibiliteit maakt onze epidemische protocollen uitermate geschikt voor zeer grote zelforganiserende systemen. Merk op dat in geen geval er sprake is van welke vorm van centraal beheer dan ook. Globale eigenschappen worden dus niet centraal opgelegd, maar komen als vanzelf tevoorschijn bij de uitvoering van een protocol.

Uit onze experimenten bleek verder dat de epidemische protocollen bijzonder goed bestand waren tegen falende knopen en verbindingen tussen knopen. Deze eigenschappen komen voort uit het feit dat alle knopen een gelijkwaardige rol spelen, met als gevolg dat het effect van een falende knoop zich niet of nauwelijks verspreidt.

Bij massaal optreden van fouten zien we wel degelijk een effect op het globale gedrag van de rest van het P2P systeem. Het bleek echter dat de prestaties, functionaliteit en betrouwbaarheid op voorspelbare wijze afnamen bij het toenemen van het aantal falende knopen. Bovendien bleek dat bij een plotseling optredende gelijktijdige uitval van een groot aantal knopen, het overgebleven systeem zich snel herstelde door naar een nieuwe topologie te convergeren met de gewenste globale eigenschappen. Tot slot bleek dat snelle wisseling in de samenstelling van participerende knopen continu gecompenseerd werd door gewenste aanpassingen in de toplogie.

Al deze goede eigenschappen gaan gepaard met een soms verrassende eenvoud: zowel Vicinity als Cyclon zijn protocollen waarvan de details zich op feitelijke en op uitermate simpele wijze laten beschrijven. Eenvoud is essentieel voor grootschalige systemen die vanuit zichzelf de natuurlijke neiging hebben complex te zijn. De combinatie van eenvoudige principes en de opmerkelijke inherente eigenschappen van epidemische protocollen ligt ten grondslag aan onze motivatie voor het exploreren van hun toepassingen in grootschalige zelforganiserende systemen.

BIBLIOGRAPHY

Abdelzaher, T., Shaikh, A., Jahanian, F., and Shin, K. (1996). Rtcast: lightweight multicast for real-time process groups. In *RTAS '96: Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS '96)*, page 250, Washington, DC, USA. IEEE Computer Society.

Adar, E. and Huberman, B. A. (2000). Free riding on gnutella. *First Monday*, 5(10).

Albert, R. and Barabási, A.-L. (2002). Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, 74:47–97.

Albert, R., Jeong, H., and Barabási, A.-L. (2000). Error and attack tolerance of complex networks. *Nature*, 406:378–382.

Allavena, A., Demers, A., and Hopcroft, J. E. (2005). Correctness of a gossip based membership protocol. In *Proceedings of the 24th annual ACM symposium on principles of distributed computing (PODC'05)*, Las Vegas, Nevada, USA. ACM Press.

Babaoglu, O. (1987). On the reliability of consensus-based fault-tolerant distributed computing systems. *ACM Trans. Comput. Syst.*, 5(4):394–416.

Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2003). Looking up Data in P2P Systems. *Commun. ACM*, 46(2):43–48.

Banerjee, S., Bhattacharjee, B., and Kommareddy, C. (2002). Scalable application layer multicast. In *SIGCOMM '02*, pages 205–217, Pittsburgh, PA.

Barabási, A.-L. (2002). *Linked: the new science of networks*. Perseus, Cambridge, Mass.

Bhagwan, R., Savage, S., and Voelker, G. (2003). Understanding Availability. In *2nd International Workshop on Peer-to-Peer Systems*, Lecture Notes on Computer Science, Berlin. Springer-Verlag.

Birman, K. P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., and Minsky, Y. (1999). Bimodal multicast. *ACM Trans. Comp. Syst.*, 17(2):41–88.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Bollobas, B. (2001). *Random Graphs*. Cambridge University Press, Cambridge, UK, 2nd edition.

Bonnet, F., Tronel, F., and Voulgaris, S. (2006). Performance analysis of cyclon an inexpensive membership protocol for unstructured p2p overlays. In *20th International Symposium on Distributed Computing (DISC 2006)*.

Broder, A. Z., Frieze, A. M., and Upfal, E. (1994). Existence and construction of edge-disjoint paths on expander graphs. *SIAM J. Comput.*, 23(5):976–989.

Broder, A. Z., Frieze, A. M., and Upfal, E. (1997). Existence and construction of edge low congestion paths on expander graphs. In *29th ACM Symposium on Theory of Computing*, pages 531–539.

Broder, A. Z., Frieze, A. M., and Upfal, E. (1999). Static and dynamic path selection on expander graphs: a random walk approach. *Random Struct. Algorithms*, 14(1):87–109.

Castro, M., Druschel, P., Hu, Y. C., and Rowstron, A. (2003). Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft Research.

Castro, M., Druschel, P., Kermarrec, A.-M., and Rowstron, A. (2002). SCRIBE: A Large-scale and Decentralized Publish-Subscribe Infrastructure. *IEEE JSAC*, 20(8).

Chun, B.-G., Wu, P., Weatherspoon, H., and Kubiatowicz, J. (2006). Chunkcast: An anycast service for large content distribution. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS06)*, Santa Barbara, CA.

Clegg, M. and Marzullo, K. (1997). A low-cost processor group membership protocol for a hard real-time distributed system. In *RTSS '97: Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, page 90, Washington, DC, USA. IEEE Computer Society.

Cohen, B. (2003). Incentives build robustness in bittorrent. In *First Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA.

Costa, P., Migliavacca, M., Picco, G. P., and Cugola, G. (2003). Introducing Reliability in Content-based Publish-Subscribe through Epidemic Algorithms. In *DEBS '03: Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8, New York, NY, USA. ACM Press.

Cristian, F. (1990). Synchronous atomic broadcast for redundant broadcast for redundant channels. *Real-Time Systems*, 2(3):195–212.

Dabek, F., Cox, R., Kaashoek, F., and Morris, R. (2004). Vivaldi: A decentralized network coordinate system. In *Proceedings of the ACM SIGCOMM '04 Conference*, Portland, Oregon.

Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J., and Stoica, I. (2003). Towards a common API for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA.

DAS-2 (no date). http://www.cs.vu.nl/das2/.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. (1987). Epidemic Algorithms for Replicated Database Maintenance. In *Sixth Symp. on Principles of Distributed Computing*, pages 1–12, New York, NY, USA. ACM Press.

Dorogovtsev, S. N. and Mendes, J. F. F. (2002). Evolution of networks. *Advances in Physics*, 51:1079–1187.

eDonkey (no date). http://www.edonkey2000.com.

El-Ansary, S., Alima, L. O., Brand, P., and Haridi, S. (2003). Efficient broadcast in structured p2p networks. In *IPTPS*, pages 304–314.

Eugster, P., Handurukande, S., Guerraoui, R., Kermarrec, A.-M., and Kouznetsov, P. (2001). Lightweight Probabilistic Broadcast. In *Int'l Conf. on Dependable Systems and Networks*. IEEE Computer Society.

Eugster, P. T., Felber, P., Guerraoui, R., and Kermarrec, A.-M. (2003a). The Many Faces of Publish/Subscribe. *ACM Comput. Surv.*, 35(2):114–131.

Eugster, P. T., Guerraoui, R., Handurukande, S. B., Kermarrec, A.-M., and Kouznetsov, P. (2003b). Lightweight probabilistic broadcast. *ACM Trans. Comp. Syst.*, 21(4):341–374.

Eugster, P. T., Guerraoui, R., Kermarrec, A.-M., and Massoulié, L. (2004). Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67.

Fessant, F. L., Handurukande, S., Kermarrec, A.-M., and Massoulié, L. (2004). Clustering in peer-to-peer file sharing workloads. In *Third Int'l Workshop on Peer-to-Peer Systems*, San Diego, USA.

Floyd, S., Jacobson, V., Liu, C.-G., McCanne, S., and Zhang, L. (1997). A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. Netw.*, 5(6):784–803.

Frieze, A. M. and Zhao, L. (1999). Optimal construction of edge-disjoint paths in random regular graphs. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 346–355, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Ganesh, A., Kermarrec, A.-M., and Massoulié, L. (2003). Peer-to-Peer Membership Management for Gossip-based Protocols. *IEEE Trans. Comp.*, 52(2):139–149.

Golding, R. A. and Taylor, K. (1992). Group membership in the epidemic style. Technical report, University of California at Santa Cruz, Santa Cruz, CA, USA.

Guo, K., Hayden, M., van Renesse, R., Vogels, W., and Birman, K. P. (1997). Gsgc: An efficient gossip-style garbage collection scheme for scalable reliable multicast. Technical Report TR97-1656, Cornell University, Ithaca, NY, USA.

Gupta, A., Sahin, O. D., Agrawal, D., and Abbadi, A. E. (2004). Meghdoot: Content-Based Publish/Subscribe over P2P Networks. In *Middleware*, pages 254–273.

Gupta, I., Birman, K., Linga, P., Demers, A., and van Renesse, R. (2003). Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*.

Gupta, I., Birman, K. P., and van Renesse, R. (2002). Fighting fire with fire: using randomized gossip to combat stochastic scalability limits. *Quality and Reliability Engineering International*, 18(3):165–184.

Gupta, I., Kermarrec, A.-M., and Ganesh, A. J. (2006). Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE Transactions on Parallel and Distributed Systems*, 17(7):593–605.

Gupta, I., van Renesse, R., and Birman, K. P. (2001). Scalable fault-tolerant aggregation in large process groups. In *DSN '01: Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*, pages 433–442, Washington, DC, USA. IEEE Computer Society.

Hadzilacos, V. and Toueg, S. (1993). *Fault-tolerant broadcasts and related problems*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Handurukande, S., Kermarrec, A.-M., Fessant, F. L., and Massoulié, L. (2004). Exploiting semantic clustering in the edonkey p2p network. In *11th ACM SIGOPS European Workshop (SIGOPS)*, Leuven, Belgium.

Harary, F. (1962). The maximum connectivity of a graph. In *Proceedings of the National Academy of Sciences*, volume 48, pages 1142–1146.

Hendrick, C. (1988). Routing information protocol. RFC 1058, IETF.

Iwanicki, K., van Steen, M., and Voulgaris, S. (2006). Gossip-based clock synchronization for large decentralized systems. In *Proceedings of the Second IEEE International Workshop on Self-Managed Networks, Systems & Services (SelfMan 2006)*, pages 28–42, Dublin, Ireland.

Jelasity, M. and Babaoglu, O. (2004). T-Man: Fast gossip-based construction of large-scale overlay topologies. Technical Report UBLCS-2004-7, University of Bologna, Department of Computer Science, Bologna, Italy.

Jelasity, M. and Babaoglu, O. (2005). T-Man: Gossip-based Overlay Topology Management. In *Proceedings of Engineering Self-Organising Applications (ESOA'05)*.

Jelasity, M. and Babaoglu, O. (2006). T-Man: Gossip-based overlay topology management. In Brueckner, S. A., Di Marzo Serugendo, G., Hales, D., and Zambonelli, F., editors, *Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Revised Selected Papers*, volume 3910 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag.

Jelasity, M., Guerraoui, R., Kermarrec, A.-M., and van Steen, M. (2004a). The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In *Fifth ACM/IFIP/USENIX International Middleware Conference*, pages 79–98, New York, NY, USA. Springer-Verlag New York, Inc.

Jelasity, M., Kowalczyk, W., and van Steen, M. (2003). Newscast Computing. Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands.

Jelasity, M., Kowalczyk, W., and van Steen, M. (2004b). An approach to massively distributed aggregate computing on peer-to-peer networks. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04)*, pages 200–207, A Coruna, Spain. IEEE Computer Society.

Jelasity, M. and Montresor, A. (2004). Epidemic-Style Proactive Aggregation in Large Overlay Networks. In *24th Int'l Conf. on Distributed Computing Systems*, pages 102–109.

Jelasity, M., Montresor, A., and Babaoglu, O. (2004c). A modular paradigm for building self-organizing peer-to-peer applications. In Di Marzo Serugendo, G., Karageorgos, A., Rana, O. F., and Zambonelli, F., editors, *Engineering Self-Organising Systems*, volume 2977 of *Lecture Notes in Artificial Intelligence*, pages 265–282. Springer. invited paper.

Jelasity, M., Montresor, A., and Babaoglu, O. (2005). Gossip-based aggregation in large dynamic networks. *ACM Trans. Comp. Syst.*, 23(3):219–252.

Jelasity, M., Montresor, A., and Babaoglu, O. (2006). The bootstrapping service. In *Proceedings of International ICDCS Workshop on Dynamic Distributed Systems (ICDCS-IWDDS'06)*, Lisboa, Portugal. IEEE Computer Society. To appear.

Jenkins, K. and Demers, A. (2001). Logarithmic harary graphs. *icdcsw*, 00:0043.

J.Pouwelse, P.Garbacki, J.Wang, A.Bakker, J.Yang, A.Iosup, D.Epema, M.Reinders, van Steen, M., and H.Sips (2006). Chitraka: A social-based peer-to-peer system. In *Proc. of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*.

Karp, R. M., Schindelhauer, C., Shenker, S., and Vöcking, B. (2000). Randomized Rumor Spreading. In *14th Symposium on the Foundations of Computer Science*, pages 565–574, Los Alamitos, CA. IEEE, IEEE Computer Society Press.

Keidar, I., Sussman, J., Marzullo, K., and Dolev, D. (2002). Moshe: A group membership service for wans. *ACM Trans. Comput. Syst.*, 20(3):191–238.

Kempe, D., Dobra, A., and Gehrke, J. (2003). Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003)*, pages 482–491. IEEE Computer Society.

Kempe, D., Kleinberg, J., and Demers, A. (2001). Spatial gossip and resource location protocols. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 163–172, New York, NY, USA. ACM Press.

Kermarrec, A.-M., Massoulié, L., and Ganesh, A. J. (2003). Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Trans. Par. Distr. Syst.*, 14(2):248–258.

King, V. and Saia, J. (2004). Choosing a random peer. In *Proceedings of the 23rd annual ACM symposium on principles of distributed computing (PODC'04)*, pages 125–130. ACM Press.

Kleinberg, J. (2000a). The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*.

Kleinberg, J. (2004). The small-world phenomenon and decentralized search. *Math Awareness Month 2004*, 37(3).

Kleinberg, J. and Rubinfeld, R. (1996). Short paths in expander graphs. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 86, Washington, DC, USA. IEEE Computer Society.

Kleinberg, J. and Tardos, E. (1995). Disjoint paths in densely embedded graphs. In *FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, page 52, Washington, DC, USA. IEEE Computer Society.

Kleinberg, J. M. (2000b). Navigation in a small world. *Nature*, 406(6798).

Kleinberg, J. M. (2001). Small-world phenomena and the dynamics of information. In *NIPS*, pages 431–438.

Kostić, D., Rodriguez, A., Albrecht, J., Bhirud, A., and Vahdat, A. (2003). Using random subsets to build scalable network services. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS 2003)*.

Kowalczyk, W. and Vlassis, N. (2004). Newscast EM. In *Advances in Neural Information Processing Systems (NIPS) 17*, Cambridge, MA. MIT Press.

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Dennis Geels, R. G., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. (2000). OceanStore: An Extremely Wide-Area Storage System. In *Nineth Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pages 190–201, Cambridge, MA. ACM.

Law, C. and Sui, K.-Y. (2003). Distributed Construction of Random Expander Networks. In *22nd INFOCOM Conf.*, Los Alamitos, CA. IEEE, IEEE Computer Society Press.

Li, M. and Vitányi, P. (1997). *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag, 2nd edition.

Lin, J. C.-H. and Paul, S. (1996). Rmtp: A reliable multicast transport protocol. In *INFOCOM*, pages 1414–1424.

Lin, M.-J. and Marzullo, K. (1999). Directional Gossip: Gossip in a Wide Area Network. In *European Dependable Computing Conference*, pages 364–379.

Lin, M.-J., Marzullo, K., and Masini, S. (2000). Gossip versus Deterministic Flooding: Low Message Overhead and High Reliability for Broadcasting on Small Networks. In *14th Int'l Symp. Distributed Computing (DISC)*, pages 253–267. University of California at San Diego.

Linga, P., Gupta, I., and Birman, K. (2004). Kache: Peer-to-peer web caching using kelips.

Loguinov, D., Kumar, A., Rai, V., and Ganesh, S. (2003). Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. In *Proceedings of ACM SIGCOMM 2003*, pages 395–406. ACM Press.

Marsaglia, G. (1995). *The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness*. Florida State University. http://www.stat.fsu.edu/pub/diehard.

Marsaglia, G. and Tsang, W. W. (2002). Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3):1–8.

Massoulie, L., Kermarrec, A.-M., and Ganesh, A. J. (2003). Network awareness and failure resilience in self-organising overlay networks. In *Symp. on Reliable Distributed Systems*, page 47, Los Alamitos, CA, USA. IEEE Computer Society.

Montresor, A., Jelasity, M., and Babaoglu, O. (2004). Robust Aggregation Protocols for Large-scale Overlay Networks. In *Int'l Conf. on Dependable Systems and Networks*, pages 19–28. IEEE Computer Society.

Montresor, A., Jelasity, M., and Babaoglu, O. (2005). Chord on demand. In *Peer-to-Peer Computing*, pages 87–94.

Moy, J. (1994). OSPF version 2. RFC 1583, IETF.

Napster (no date). http://www.napster.com.

Newman, M. (2002). Random Graphs as Models of Networks. In Bornholdt, S. and Schuster, H. G., editors, *Handbook of Graphs and Networks: From the Genome to the Internet*, chapter 2. John Wiley, New York.

Padmanabhan, V. N. and Sripanidkulchai, K. (2002). The case for cooperative networking. In *Peer-to-Peer Systems: First International Workshop, IPTPS 2002*, pages 178–190, Cambridge, MA, USA.

Pandurangan, G., Raghavan, P., and Upfal, E. (2003). Building low-diameter peer-to-peer networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 21(6):995–1002.

Pastor-Satorras, R. and Vespignani, A. (2001). Epidemic dynamics and endemic states in complex networks. *Physical Review E*, 63:066117.

PeerSim (no date). http://peersim.sourceforge.net.

Peleg, D. and Upfal, E. (1987). Constructing disjoint paths on expander graphs. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 264–273, New York, NY, USA. ACM Press.

Peleg, D. and Upfal, E. (1989). Constructing disjoint paths on expander graphs. *Combinatorica*, 9:289–313.

Piantoni, R. and Stancescu, C. (1997). Implementing the swiss exchange trading system. In *FTCS '97: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, page 309, Washington, DC, USA. IEEE Computer Society.

Pittel, B. (1987). On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. (2001a). A Scalable Content-Addressable Network. In *SIGCOMM*, pages 161–172, San Diego, CA. ACM.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001b). A Scalable Content-Addressable Network. In *SIGCOMM*, pages 161–172, San Diego, CA.

Renesse, R. V., Birman, K. P., and Vogels, W. (2003). Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206.

Risson, J. and Moors, T. (2006). Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*. To appear.

Rowstron, A. and Druschel, P. (2001a). Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware 2001*, Heidelberg, Germany.

Rowstron, A. and Druschel, P. (2001b). Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *18th Symp. Operating System Principles*, Banff, Canada. ACM.

Saroiu, S., Gummadi, P. K., and Gribble, S. D. (2002). A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA.

Saroiu, S., Gummadi, P. K., and Gribble, S. D. (2003). Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems Journal*, 9(2):170–184.

Sen, S. and Wang, J. (2004). Analyzing Peer-to-Peer Traffic Across Large Networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232.

Sherwood, R., Braud, R., and Bhattacharjee, B. (2004). Slurpie: A cooperative bulk data transfer protocol. In *INFOCOM*.

Sripanidkulchai, K., Maggs, B., and Zhang, H. (2003). Efficient content location using interest-based locality in peer-to-peer systems. In *INFOCOM Conference*.

Stavrou, A., Rubenstein, D., and Sahu, S. (2004). A Lightweight, Robust P2P System to Handle Flash Crowds. *IEEE Journal on Selected Areas in Communications*, 22(1):6–17.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, pages 149–160, San Diego, CA. ACM.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. (2003). Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. *ACM/IEEE Trans. Netw.*, 11(1):17–32.

Sun, Q. and Sturman, D. (2000). A Gossip-Based Reliable Multicast for Large-Scale High-Throughput Applications. In *International Conference on Dependable Systems and Networks*, pages 347–358, Los Alamitos, CA. IEEE, IEEE Computer Society Press.

Szymaniak, M., Pierre, G., and van Steen, M. (2004). Scalable Cooperative Latency Estimation. In *Tenth Int'l Conf. Parallel and Distributed Systems*, Los Alamitos, CA. IEEE, IEEE Computer Society Press.

Terpstra, W. W., Behnel, S., Fiege, L., Zeidler, A., and Buchmann, A. P. (2003). A Peer-to-Peer Approach to Content-Based Publish/Subscribe. In *DEBS*, pages 1–8, San Diego, CA.

van Renesse, R., Minsky, Y., and Hayden, M. (1998). A gossip-based failure detection service. In *Proc. of Middleware, the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 55–70, The Lake District, England. IFIP.

Venkataraman, V., Francisy, P., and Calandrino, J. (2006). Chunkyspread: Multi-tree unstructured peer-to-peer. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS06)*, Santa Barbara, CA.

Voulgaris, S., Gavidia, D., and van Steen, M. (2005). Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217.

Voulgaris, S., Jelasity, M., and van Steen, M. (2003). A Robust and Scalable Peer-to-Peer Gossiping Protocol. In *2nd International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2003)*, Melbourne, Australia.

Voulgaris, S., Kermarrec, A.-M., Massoulié, L., and van Steen, M. (2001). Exploiting semantic proximity in peer-to-peer content searching. In *10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004)*, Suzhu, China.

Voulgaris, S., Rivière, E., Kermarrec, A.-M., and van Steen, M. (2006). Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS06)*.

Voulgaris, S. and van Steen, M. (2003). An Epidemic Protocol for Managing Routing Tables in very large Peer-to-Peer Networks. In *14th IFIP/IEEEWorkshop on Distributed Systems: Operations and Management (DSOM 2003)*, volume 2867 of *Lecture Notes on Computer Science*, pages 41–54, Berlin. IFIP/IEEE, Springer-Verlag.

Voulgaris, S. and van Steen, M. (2005). Epidemic-style Management of Semantic Overlays for Content-Based Searching. In *EuroPar*, LNCS 3648, pages 1143–1152. Springer-Verlag.

Watts, D. J. (1999). *Small Worlds, The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, NJ.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393:440–442.

Wouhaybi, R. and Campbell, A. T. (2004). Supporting Resilient Low-Diameter Peer-to-Peer Topologies. In *23rd INFOCOM Conf.*, Los Alamitos, CA. IEEE, IEEE Computer Society Press.

Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., and Kubiatowicz, J. D. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53.

Zhao, B. Y., Kubiatowicz, J., and Joseph, A. D. (2001). Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report CSD-01-1141, Computer Science Division, University of California, Berkeley.

Zhong, M., Shen, K., and Seiferas, J. (2005). Non-uniform random membership management in peer-to-peer networks. In *Proc. of the IEEE INFOCOM*, Miami, FL.

Zhuang, S. Q., Zhao, B. Y., Joseph, A. D., Katz, R. H., and Kubiatowicz, J. D. (2001). Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20, New York, NY, USA. ACM Press.