Dottorato di Ricerca in Informatica
Università di Bologna, Padova

Secure Gossiping Techniques and Components

Gian Paolo Jesi

March 2007

Coordinatore:	Tutore:
Prof. Özalp Babaoğlu	Prof. Özalp Babaoğlu

This thesis is dedicated to my beloved Mum

Abstract

Gossip protocols have proved to be a viable solution to set-up and manage large-scale P2P services or applications in a fully decentralised scenario.

The gossip or epidemic communication scheme is heavily based on stochastic behaviors and it is the fundamental idea behind many large-scale P2P protocols. It provides many remarkable features, such as scalability, robustness to failures, emergent load balancing capabilities, fast spreading, and redundancy of information. In some sense, these services or protocols mimic natural system behaviors in order to achieve their goals.

The key idea of this work is that the remarkable properties of gossip hold when all the participants follow the rules dictated by the actual protocols. If one or more malicious nodes join the network and start cheating according to some strategy, the result can be catastrophic.

In order to study how serious the threat posed by malicious nodes can be and what can be done to prevent attackers from cheating, we focused on a general attack model aimed to defeat a key service in gossip overlay networks (the *Peer Sampling Service* [JGKvS04]). We also focused on the problem of protecting against forged information exchanged in gossip services.

We propose a solution technique for each problem; both techniques are general enough to be applied to distinct service implementations. As gossip protocols, our solutions are based on stochastic behavior and are fully decentralized.

In addition, each technique's behaviour is abstracted by a general primitive function extending the basic gossip scheme; this approach allows the adoptions of our solutions with minimal changes in different scenarios.

We provide an extensive experimental evaluation to support the effectiveness of our techniques. Basically, these techniques aim to be *building blocks* or *P2P architecture guidelines* in building more resilient and more secure P2P services.

Acknowledgements

There are many people to thank for their support and throughout my Ph.D.

Firstly, I would like to thank my supervisor, Prof. Özalp Babaoğlu, who gave me the chance to discover the world of research.

A special thanks to Alberto Montresor for his help and patience.

I owe a special debt of gratitude to Maarten van Steen (Vrije University of Amsterdam, The Netherlands) for his patience, encouragement, precious suggestions and advices concerning my work. I also wish to thank all the nice people I met during my internship at the Vrije and especially Daniela Gavidia and Chandana Gamage.

Thanks also to David Hales for the fruitful discussions we had and to Lorenzo Alvisi for his comments and suggestions.

Last, but by no means least, thanks to my Dad and my grandparents for their constant presence and to my Ph.D colleagues with whom I have shared the dark and unhealthy rooms of the underground lab!

Contents

List of Figures

Chapter 1

Introduction

This work aims to study the effect that malicious attacks can have on a gossip-based networks. In particular, we first concentrate on a generic attack model designed to defeat a very fundamental service, the Peer Sampling Service (PSS), in gossip-based networks. In brief, the PSS is a topology manager that builds and maintains by gossiping a random graph-like overlay. The overlay is continuously rewired over time and therefore each node has a fresh, random sample of other node references stored in its local cache. The PSS ensures strong connectivity and a high resilience to benign failures (crashes).

The contribution of this thesis is then to show how the aforementioned service can be easily damaged or defeated and how this threat can also affect other services relying on the PSS. Then, we propose a general solution to counter-measure the attack and to limit the damages to a negligible level. Secondly, we propose an efficient technique targeting a certain class of protocols relying on the PSS. This technique is aimed at preventing the diffusion of forged information by malicious attackers on behalf of well-behaving peers.

The motivations of this work are summarised as follows.

1.1 Motivations

Recent years have witnessed a growing interest in the area of application-layer overlay protocols and peer-to-peer (P2P) systems. Examples include popular file-sharing applications [Gnu, kaz, The, bit], information dissemination [JGJ+00, CRSZ01, EGKM04b, EGH+03], multimedia streaming applications [CDK+03, KRAV03] as well as publish/subscribe systems [CRW01, CDKR02, PB02].

The interest towards the P2P paradigm is motivated by its intrinsic decentralisation; informally, the P2P paradigm has introduced a sort of "democracy" in distributed systems, in which each peer has equal importance. Essentially, each peer can play both the consumer (client) and the producer (server) of information. This idea brings many advantages compared to the classic client-server paradigm, in which central servers play a substantial role. In fact, the load (messages) sustained by each peer is in general much lower than the one sustained by a central server and the failure of any peer is not an issue; therefore, P2P systems leads to an uniform usage of resources and do not present a single point of failure.

The interesting features we previously discussed comes at a cost; in general P2P systems are much harder to design, to deploy and to maintain. P2P systems are known to be very dynamic: peers join and leave the overlay continuously (a so called *churning* process). This dynamism adds further complexity over the usual client-server model, in which dynamism is restricted to the clients.

The inspiration given by other disciplines (e.g., biology or social science), can help to achieve the properties needed by P2P systems. In fact, a new "breed" of algorithms and protocols is growing rapidly. These new protocols are called *epidemic* or *gossip-based*. To communicate in a scalable manner, they mimic how epidemics spread in natural systems; their behavior is not strictly deterministic, but relies heavily on stochastic processes.

The communication in P2P systems is performed over an *overlay network*, superimposed over a routed network, such as the Internet. In order to achieve the

best possible results and according to the nature of the actual running application, the peers are arranged according to a particular *overlay-topology*. The topology provides the relation "who knows whom" and has a crucial impact on the performance of an application. In addition, each peer can know about only a small subset of the nodes in the system because of scalability reasons. Distinct topologies have distinct properties suitable for specific task. We will see a brief taxonomy in the following section.

The attractive properties of gossip-based protocols we stated previously, are achieved under the assumption that each peer follows the rules dictated by the protocol. In fact, it is crucial to note that the robustness to failures and the natural P2P friendliness to dynamic environment conditions, should not be confused with resilience to *malicious attacks*.

Our fundamental question is *how do gossip systems react to malicious actions or attacks*? This will be the central focus of our work.

Starting from this generic question, many others arise, such as: how easily can a network be damaged? What are the consequences of such damage? Do they lead to transient problems or to more severe problems? What amount of damage is tolerable by a specific protocol? How long does it take to successfully perform an attack?

Of course, these questions are indeed very generic, since they depend on the actual protocol or application under attack and on how the attack is designed and performed (e.g., the attacker's goal).

We present a simple example to show that we are not dealing with just an academic issue, but with a practical problem. To illustrate the problem, consider a gossiping network in which each node maintains a list of 20 neighbors, called its *partial view* (e.g., it provides the relation "who knows whom"). Elements of these views are continuously updated and exchanged between nodes. Essentially, the peers are wired in a random graph (topology) fashion with out-degree 20 and the graph is continuously rewired over time. 20 nodes in the system start behaving maliciously exchanging forged partial views; then, after a short amount of

time, they leave the network. Figure 1.1(a) shows the overlay in normal conditions; Figure 1.1(b) instead, shows the same graph after the malicious nodes exit: within a very short time the original overlay is completely disrupted. This example refers to a small, 1000 nodes network, but as we show in Chapter 3, no matter what the size of the network, a successful attack can be carried out swiftly. We will re-discuss this example with more detail in Chapter 3, when the particular nature of the gossip protocols involved will be clearly defined.

1.2 Topology taxonomy

As we previously stated, P2P systems build an overlay topology on top of the usual IP-level network. The overlay plays the important task of ensuring connectivity (e.g., neighbors to contact over the overlay) to the system nodes; in general, due to the massive size a P2P system can reach, each participant can know about only a small subset of the overlay, which we refer to as its *neighborhood*. Usually, the size of the neighborhood is fixed.

Informally, the *topology* tells us how the nodes are wired together. The actual wiring rule may vary depending on the goals to achieve (e.g., file sharing, searching, multimedia streaming, etc.). The topology may thus be a fundamental design choice to efficiently perform a certain function.

Essentially, the distributed systems literature distinguishes between two main overlay network classes: *structured* and *unstructured*. In the former class, nodes are organised in hierarchical structures that can grow or shrink according to strict mathematical organisation rules; it is essentially composed by *Distributed Hash Tables* (DHT) such as Chord [DBK+01], Pastry [RD01, fre], CAN [RFH+01] or other derivatives. As conventional hash tables, DHTs provide an efficient distributed solution for storing and retrieving items. They are suited to store items that can be mapped to a unique key value (e.g, the hash value of a file name in a file sharing scenario).

In the latter class, the rule used to build the actual overlay encourages the

emergence of a (pseudo) random graph. This kind of approach has scalability problems [JAB01] for some kinds of tasks (e.g., locating resources)due to the tendency to flood the overlay with queries, but it has minimal maintenance overhead in comparison to the structured approach.

Our simple taxonomy also include *superpeer* topologies [Mon04, JMB06], but as a sub category of structured overlays. This third kind of topology is motivated by the fact that original P2P systems were based on a complete "democracy" among nodes: "everyone is a peer". But physical hosts running P2P software are usually very heterogeneous in terms of computing, storage and communication resources, ranging from high-end servers to low-end desktop machines. The superpeer topology addresses this fact and assigns to the most powerful nodes (according to some rule or to their actual resources) some extra tasks or server capabilities (e.g., indexing or query routing) for other (weaker) nodes. Recent versions of the Gnutella [Gnu] file sharing software and Kazaa [kaz] have been the first examples of such approaches available to the public. The Skype [Sky] IP-telephony software is claimed to use the same kind of topology. However, this topology flavour is usually considered as structured.

The *topology management* protocol is the application or protocol suite component that maintains the topological properties of the overlay. These properties must hold in a real-world P2P scenario, such as when nodes continuously join and leave the system (*churning*) and when crashes occur. Other issues can complicate further the situation, such as the coexistence of multiple versions of the same software.

It is clear that the efficiency and the ratio of successful operations performed by an application heavily relies on topology management. If the topology is subverted, the topology manager should be able to recover its structure as, the topology is only *perception of the environment* from a peer point of view. A wrong perception of the environment may have severe consequences.

1.2.1 Basic requirements for topology management

In order to study the effect of malicious attacks we need to identify a basic core functionality among gossip-based protocols. We need a simple and manageable, but still realistic common denominator to work with.

We decided to dedicate our attention to the unstructured approach, because much of the attention of the literature has been focused on structured security approaches and because in [DKK+05] the authors suggest that unstructured systems are more vulnerable than DHTs.

The first action a peer must accomplish to enter in a P2P system is joining the desired overlay. The first step needed by a peer is obtaining a neighborhood list; for example, the first time a Skype [Sky, BS06] application runs, it asks for a neighbor list to one randomly chosen server (a high availability Skype node) among a fixed, pre-assigned set. This node will provide a fresh list of available Skype nodes to the newcomer. When the newcomer leaves (voluntarily) the system, he saves his last neighborhood list to reuse it next time.

The need to set up a reliable neighborhood for reliable *connectivity* is a primary concern not only for Skype software, but for any P2P system.

Basically, the *connectivity* concept (e.g., the nodes' capacity to interact together in the overlay) is given by the combination of the network routing facility and the overlay neighborhood. This functionality can be achieved with distinct strategies, but it is always the first step to accomplish.

The connectivity properties are directly related to the actual graph structure characteristics (degree, clustering coefficient, average path length, etc.).

We have identified a particular service, the *Peer Sampling Service* (PSS) [JGKvS04] as a gossip-based facility that provides strong connectivity. This service provides each node with a list of neighbors picked from the current network population; the list changes at regular time intervals. The neighbors in its list appear as a uniform random sample of the current network participants.

When designing gossip-based protocols, the PSS is a valuable component as it provides any node with random neighbor selection facility and connectivity

(neighborhood list). In fact, many other gossip protocols are explicitly based on its presence [JMB05, JB05, VGvS05, Mon04, JMB06, CJ05].

The PSS has a family of distinct fully decentralised implementations [JKvS03, VGvS05]. All implementations share the fact that nodes are arranged in a random graph-like fashion. In other words, the PSS graph shares the same important properties of a random graph (e.g., strong network connectivity, low clustering of nodes, uniform node degree distribution). This ensures robustness to benign failures and a fast spreading of the information through the overlay.

As in the vast majority of P2P protocols, the PSS relies on the assumption that all the participants follow the rules dictated by the protocol. No explicit trust or security measures are adopted. We will show that poisoning the PSS functionality can have a dramatic effect on the whole P2P system.

We adopt the PSS to test the effect of malicious behavior in P2P gossiping environments. In fact, attacking the PSS would cause not only trouble to the PSS itself, but also to the services that rely on it.

1.3 Cheating and attack principles

It is unfeasible to define secure solutions for a service without defining which are the threats we intend to address.

As a first constraint, we consider that cheating techniques or attacks are based on message exchange of malicious information aimed to trigger some unexpected side effect in well-behaving peers. Essentially, any malicious action is performed at the overlay level only.

The goal and the nature of an attack in a P2P gossip network can be similar to the standard counterpart for traditional distributed systems. In a denial of service (DOS) attack for example, an attacker may try to overload one or more target nodes in order to defeat them. This scenario may happen in a malicious DHT in which ordinary nodes, cheated by an attacker, can forward any message

to the victim(s). Essentially, a non negligible number of nodes start behaving maliciously without the voluntary intention of being malicious (see Chapter 7).

We consider that an attacker can basically aspire to two distinct **goals**: (i) achieving an advantage at the expense of the rest of the system or (ii) achieving a massive DoS leading to the overlay destruction.

The former behavior is highly dependent on the nature of the actual running application, as the concept of "advantage" can assume many distinct meanings. In a resource-sharing application for example, it is the equivalent of *free-riding*[1]. In other contexts, the attackers may be interested in obtaining a leader position in the underlying topology; this position may help a higher-layer application to spread their information messages faster, while dropping potential competitor messages. It is important to note that, *in a healthy gossip overlay, the message drop performed by few attackers would have no effect* because of the high degree of the overlay graph. The high degree provides redundancy in the spreading of the information. This fact emphasises the importance of securing the lower connectivity layer.

In the context of the scenario we are modeling, both attacker's goals can be achieved by modifying the current PSS topology. By gossiping malicious information, the attackers can silently modify the relation "who knows whom" in order to acquire an advantage for their upper level services (or protocols) or to destroy the network.

To complicate further the scenario, the attackers may run not only malicious PSS instances, but also malicious upper-level services.

The attack model we are going to present is Chapter 3 can affect also the protocol layers relying on the PSS, but of course securing the PSS is insufficient to secure the other layers. Essentially, securing the PSS means ensuring that the basic connectivity layer has the topology that a regular PSS is supposed to have (or maintaining the PSS topological properties).

[1]When free-riding, a node tries to consume as much resources as he can, but without sharing its own resources.

Other services distinct from topology managers, but relying on the PSS, may use it to gossip maliciously forged information as well. The effect of the attackers may vary, but for example, in an advertisement distribution application the attackers may spread their advertisement, while forging the other peer advertisements in order to make the latter much less interesting than the former. In Chapter 5, we propose an efficient technique to detect and limit to a minimum the spreading of this kind of forged information. Securing higher layers involves a totally different approach than securing the PSS.

1.4 Roadmap

The central focus of this thesis is the design of fully decentralised techniques and components for secure gossiping. In particular, our aim is to prevent a generic attack that mutates and possibly destroys the overlay-level topology provided by a PSS.

Along the way, we describe the attack model and we show its effectiveness against P2P systems. Then we present our attack countermeasure and another component aimed at preventing the spreading of forged information.

The remainder of this work is organised as follows:

- **Chapter 2:** we provide an overview of the PSS motivations and properties. Two distinct PSS implementations are briefly described; these implementations have been adopted for our work on gossip security.
- **Chapter 3:** we introduce our generic attack model, the "hub attack"; we motivate its novelty and real-world relevance in the P2P area. We present experimental results, obtained by extensive simulations, supporting our claim.
- **Chapter 4:** we present how the hub attack affects other gossip protocols relying on the PSS. In particular, we present and describe three distinct gossip protocols and we test the impact of having a poisoned PSS for such

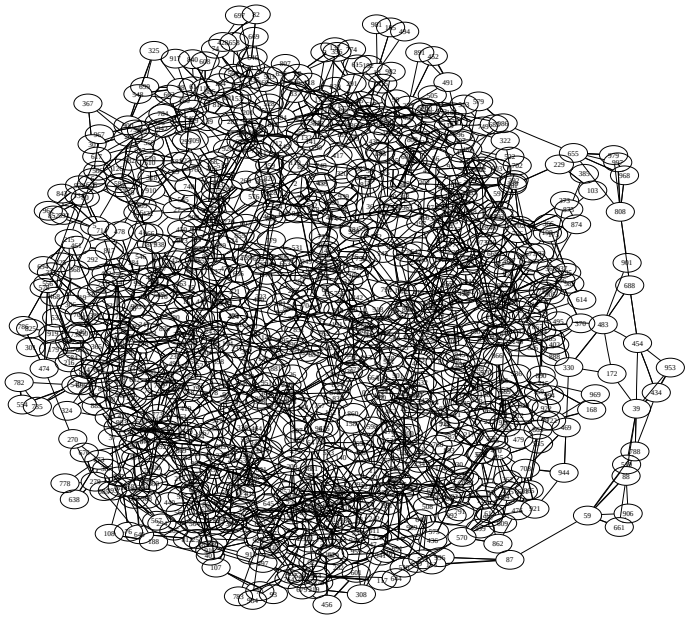
protocols. Then, due to the generality of our attack model, we test directly the effect of the attack over the protocols themselves instead of poisoning the PSS layer.

- **Chapter 5:** we present our first secure gossiping component, the SPSS, to prevent the hub attack and we perform an extensive analysis. The analysis reveals the efficiency of the solution and its independence from the actual PSS implementation. Our solution requires the presence of one or more trusted nodes for maximum security (e.g., used between several organisations) or can be configured in a fully distributed fashion for maximum scalability.

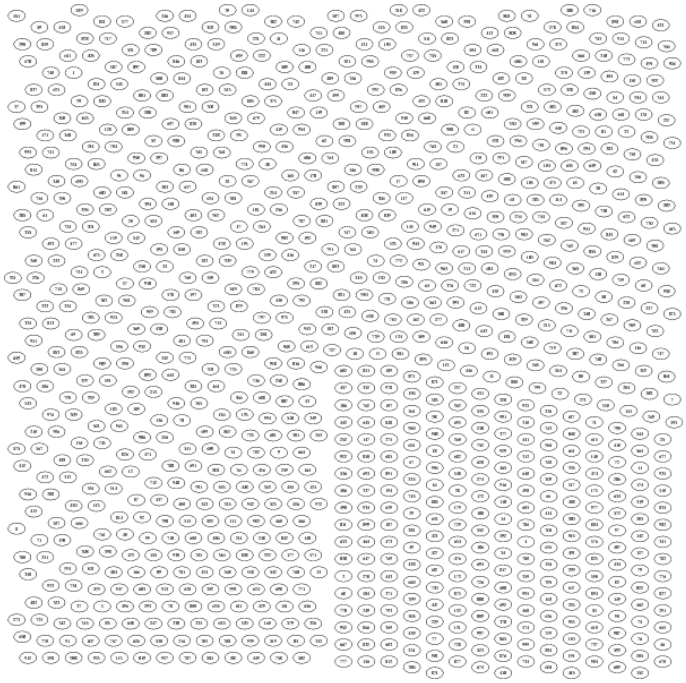
- **Chapter 6:** since the SPSS focuses on securing the lower overlay management layer, we introduce another technique that allows a node to detect any forged information item and to ban the malicious sender. The information items can be spread by a generic gossip spreading protocol. Again, we support our claims with extensive simulation results.

- **Chapter 7:** we discuss the current existing attacks and cheating techniques and their respective solutions, if any.

- **Chapter 8:** we summarise our main contributions and we discuss future research.



(a) A healthy random graph topology



(b) The previous random graph after an attack

Figure 1.1: Overlay mesh status before and after a simple attack: the random graph depicted in (a) becomes fully disconnected. The graph out-degree (constant) is set to 20, but only 3 links per node are printed. Less than 20 gossiping cycles are required to disrupt the graph. Network size is 1000 nodes.

Chapter 2

The Peer Sampling Service

In this chapter we introduce the reader to the Peer Sampling Service (PSS). We address its motivations, characteristics and we briefly review two specific implementations. We also describe the general system model in which we operate.

2.1 Introduction to the PSS

The gossip based communication model in large-scale distributed systems has been successfully applied in many areas, such as: information dissemination [EGKM04a], aggregation [JM04, JMB05], load balancing [JMB04] and synchronisation [MJB04].

The common key point of these approaches is that, periodically, every node in the system performs an information exchange with some of its peers. The underlying service that provides each need with a list of other peers is a fundamental component for these kind of systems. In general, this service is usually referred as the *Peer Sampling Service* (PSS) [JGKvS04] and it is assumed to be implemented in such a way that any node can exchange information with uniformly random selected peers; these peers are selected among all the currently available participants in the network.

To achieve this random selection assumption, some implementers proposed a solution where every node knows all the other participants in the system [KMG03].

do forever
wait(Δt)
neighbour = SELECTPEER()
SENDSTATE(neighbour)
n_state = RECEIVESTATE()
my_state.UPDATE(n_state)

(a) Active Thread

do forever
n_state = RECEIVESTATE()
SENDSTATE(n_state.sender)
my_state.UPDATE(n_state)

(b) Passive Thread

Figure 2.1: The epidemic or gossip paradigm.

In other words, each node maintains a dynamic list of nodes, usually called view or cache, which can grow with the size of the system. The maintenance cost of this structure is non negligible, especially in a dynamic system where nodes can join or leave at any time. For modern hardware, the memory footprint of this approach is not the main concern, but the quality of the information stored in the lists becomes problematic. It is interesting to note that while gossip systems are known to be scalable, the PSS implementation on which they are based may not.

A better idea to build scalable PSS implementations is to use the gossip-based paradigm, depicted in Figure 2.1, to diffuse the membership information, while keeping constant in size each peer's local list of nodes. The continuos gossip of the membership information enables the emergence of dynamic, unstructured overlay networks that express the dynamic nature of P2P systems. In addition, the overlay dynamism ensures good connectivity when nodes crash or disconnect.

The very generic nature of the gossip paradigm allows the existence of many variants of the membership dissemination strategy.

In [JGKvS04], the authors presents a taxonomy of possible PSS implementations according to three criterions: (i) peer selection (ps), (ii) cache selection (cs) and (iii) cache propagation (cp). These criterions correspond to distinct implementation behaviours of the SELECTPEER(), RECEIVESTATE() / SEND-

Peer selection (ps)	
random	Uniformly random selection of a node from the node's local cache
head	Select the first node in the local cache
tail	Select the last node in the local cache
Cache selection (cs)	
random	Uniformly random selection of c nodes without replacement from the (local) cache
head	Select the first c nodes from the (local) cache
tail	Select the last c nodes from the (local) cache
Cache propagation (cp)	
push	The current node sends its cache to the selected peer
pull	The current node asks for the selected peer cache
push-pull	Both nodes exchange their caches

Table 2.1: Distinct behaviours for each considered feature. A short, semantic explanation is given for each option. These options lead to 27 distinct PSS implementations.

STATE() [1] and UPDATE() methods. In Table 2.1 is shown the semantic of the considered options for each method. These options lead to 27 distinct PSS implementations; each one can be represented by the following tuple: (ps, cs, cp), where each element represents one of the three possible options for the corresponding policy.

The PSS API also requires the presence of an INIT() method (not shown in Figure 2.1). This method is responsible for node's cache initialisation; in other words, it has to *bootstrap* the node by filling its cache with (valid) node references. The bootstrapping problem can be solved by out-of-band techniques, for example

[1] The method signature refers to a generic "state" entity, but the actual information exchanged is the node's cache.

using a set of well-known nodes or a central service publishing node identifiers. Essentially, these are the approaches adopted by some well known P2P application, such as Skype [Sky] or many Gnutella [Gnu] clients.

The authors define an experimental methodology to evaluate these distinct protocols. The methodology focuses on the emergent overlay induced by the gossip interactions among peers. In particular, the convergence to the desirable uniform random model has been evaluated; other graph-like properties (e.g., the degree distribution, the average path length and the clustering coefficient) and the system reliability, in terms of self-healing, have been checked.

The key aspect highlighted by the evaluation is that the examined protocols, sharing the feature of building an unstructured overlay using partial views, can lead to different emergent overlays, *none of which resembles a random graph*. Instead, the overlays seems to belong to the family of *small-world graphs*, in which a small diameter and a large clustering are the typical characteristics.

However, whether from a system-wide point of view the overlay is far from being a random graph, from a node's point of view instead, the peer selection from the local cache can still be considered (almost) random. This property is a cornerstone for many gossip protocols and, in particular, for the protocols discussed in this work (see Chapters 4, 5 and 6).

We considered two distinct PSS implementations called NEWSCAST and *basic-shuffling*. Both are heavily inspired by the prototypal gossip scheme depicted in Figure 2.1, and, using the tuple notation, they correspond to (*random, head, push-pull*) and (*random, random, push-pull*) respectively. We obtained similar results with both implementations. Before introducing our attack model in the next chapter, we provide our general system model and a background of these implementations.

2.2 System model

The system model presented in this section holds in all the other sections of this work; it will be incrementally extended when required by the introduction of a specific protocol.

We consider a network consisting of a large collection of *nodes* that can join or leave at any time. Leaving the network can be voluntary or due to a *crash*. We assume the presence of a routed network (e.g., the Internet) in which any node can, in principle, contact any other party.

Any node in the network must be addressable by a *unique node identifier* (ID), such as an ⟨IP-address, port⟩ pair. Notice that we have chosen this simple form of ID as a more sophisticated one would add unnecessary details to our general model. However, an ID suitable for a real-world protocol must address many issues. For example, a single host may run several instances of an application under distinct user's domains and hence the ID must distinguish among different instances and users. A further complication is represented by the presence of firewalls between peers and by NAT routing. These issues suggest that a real ID is a complex structure and needs a careful design.

Because of scalability constraints, a node knows about only a small subset of other nodes. This subset, which may change, is stored in a *local cache*, while the node IDs it holds are called *neighbors*. This set provides the connectivity for a node in the overlay; the absence of items in this set or the presence of incorrect or bogus IDs leads to an unrecoverable situation. In this case, a new initialisation or *bootstrap* is required. In general, P2P applications provide a set of well-known, highly available nodes in order to be used as a bootstrap facility and hence as the initial neighborhood set.

In the cache, a *timestamp* is associated with each distinct node ID in order to eventually purge "old" ID references according to an ageing policy [JKvS03].

The notion of time in our model is not strict because our gossip protocols need not be synchronised. We measure time in generic *time units* or *cycles* during which

each node has the possibility to initiate a gossip exchange with another randomly selected node from its local cache.

We use the following terminology: the *pollution* is the presence of IDs of malicious nodes in a peer's cache. A *node is defeated* if all the entries in its cache refer to malicious nodes (i.e., 100% polluted) and an *overlay is defeated* or *destroyed* if every node in the overlay is defeated, or in other words, if it is completely partitioned (e.g., each peer has no more neighbors).

2.2.1 PSS implementation: Newscast

Newscast [JKvS03] is a gossip-based protocol that builds and maintains a continuously changing random graph (or *overlay*). The generated topology is very stable and provides robust connectivity. This protocol has been a successful building block to implement several P2P protocols [JMB05, JB05, CJ05]. The NEWSCAST implementation of the PSS corresponds to the (*random, head, push-pull*) tuple, using the notation introduced in Section 2.1.

In NEWSCAST, each node maintains a cache containing c IDs extended with a logical timestamp (*ts*) representing its creation time. The protocol behavior follows strictly the gossip scheme; periodically, a node A does the following: (i) it selects a random peer B from its local cache; (ii) then updates its local timestamp; and (iii) performs a *cache exchange* with B. The exchange involves sending A's cache along with its own ID and receiving B's cache and ID.

After the exchange, each party merges the received cache with its current one and keeps the c "freshest" IDs as measured by the timestamp associated with each ID. No multiple copies of the same ID in a single cache are allowed. Each peer always puts the other party's ID in first position in its own cache after the exchange. This sequence of actions is depicted in Figure 2.2.

This exchange mechanism has three effects:

1. caches are continuously shuffled, creating a topology with a low diameter that is close to a random graph with out-degree c. Experimental results (see

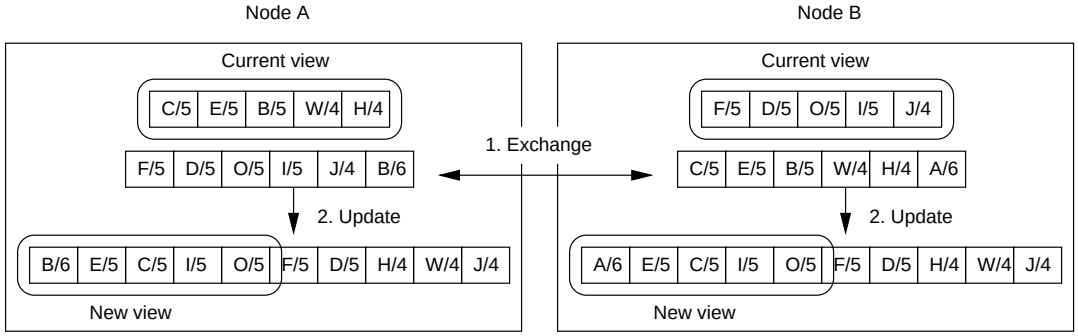


Figure 2.2: A NEWSCAST gossip-exchange between node A and B. Cache size $c = 5$. An ID is represented by a capital letter along with its timestamp. The exchange time is cycle 6.

[JKvS03]) proved that a small 20 elements local cache is already sufficient for a very stable and robust connectivity.

2. the resulting topology is strongly connected.

3. the overlay is self-repairing, since crashed nodes cannot inject new descriptors any more, so their information quickly disappears from the system because of the timestamp ageing policy.

Newscast is also cheap in terms of network communication. The traffic generated by the protocol is estimated in [JKvS03]. Summarising very briefly, the number of exchanges per cycle can be modelled by the random variable $1+\phi$, where ϕ has a Poisson distribution with parameter 1. Thus, on average, we expect two exchanges per cycle (one active and one passive, see Figure 2.1). This involves the exchange of a few hundred bytes per cycle for each peer.

2.2.2 PSS implementation: basic-shuffling

The basic shuffling is the foundation algorithm of the more sophisticated Cyclon [VGvS05] protocol. The basic-shuffling implementation of the PSS corre-

sponds to the (*random, random, push-pull*) tuple, using the notation introduced in Section 2.1.

We will be brief about the protocol and, for any further detail, refer to [VGvS05]. Each peer P periodically performs the following steps:

1. select a random, non-empty subset of l node IDs from its cache, where l is a system wide parameter; then pick a random neighbor Q from this set
2. replace Q ID with P's ID
3. send the updated subset to Q
4. receive from Q a subset ($\neq \emptyset$) of no more than l ID entries regarding Q's neighbors
5. discard entries pointing to P and any entry already in P's cache
6. update P's cache to include all remaining entries, by firstly using empty cache slots (if any), and secondly replacing entries among the ones originally sent to Q

When Q receives an exchange request, it randomly selects a subset of its own neighbors, of size no more than l, sends it to the initiating node (P), and executes steps 5 and 6 to update its own cache accordingly.

In contrast to NEWSCAST, the topology emerged by basic-shuffling exhibits a lower clustering coefficient, resulting in a mesh closer to a real random graph. However, a drawback is the longer time required to purge the IDs of nodes which have left the network.

Chapter 3

Attack model and analysis

In this chapter we introduce our attack model. We show its effectiveness in destroying or causing other severe damages to the *Peer Sampling Service* (PSS) overlay.

Before dealing with the details of the attack and introducing our attack model, we first start with an example scenario we have already seen briefly in Section 1.1. Then, we describe the actual strategy and algorithm played by the attackers and we evaluate the effects of the attack over the PSS topology. In this chapter, we do not adopt any countermeasure against the attack.

3.1 Attack scenario

Figure 3.1 shows the impact that our attack model can have on a network, in order to give a quick understanding of its relevance. The attack leads to a completely partitioned network. The time require to achieve this massive (DoS) destruction is just less than 20 cycles. The network size is 1,000 nodes, but the same result holds for larger networks (see Section 3.3).

In this scenario, the cache size is set to 20 IDs, as well as the number of malicious nodes in the system. Malicious nodes know each other and execute the standard PSS algorithm, NEWSCAST in this particular scenario. However, when a malicious node initiates a gossip exchange, it fills its cache with the IDs of the

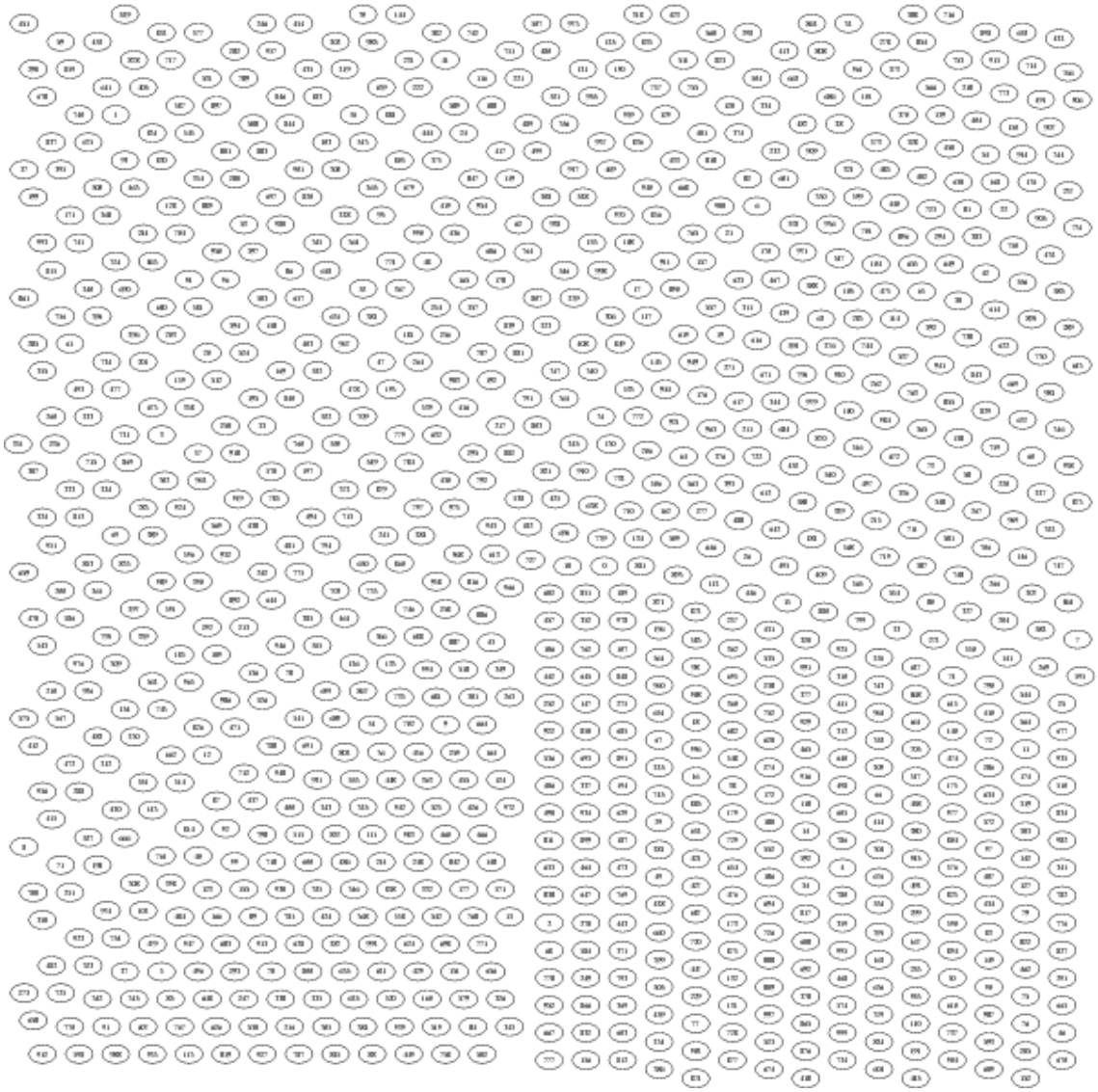


Figure 3.1: Overlay mesh status after a hub-attack: each node is fully disconnected. The PSS cache size is fixed to 20 IDs. Less than 20 gossiping cycles are required to disrupt the PSS. Network size is 1000 nodes.

other malicious nodes. In addition, it forges each ID by giving it a fresh timestamp. A malicious node always provides this information to a non-malicious node. As a consequence of using fresh timestamps, the latter will always replace its entire cache with the one sent by the malicious node, effectively immediately

isolating it from other non-malicious nodes, at which point it is defeated.

Only a nondefeated, nonmalicious node B can help a defeated one, say A, provided B has A's ID in its local cache. However, when B contacts A, we can expect that after the exchange half of the local caches of A and B, respectively, will be polluted with IDs of malicious nodes. As a consequence, there is a 50% chance that each will contact such a node, in turn, being defeated. But even contacting a nonmalicious node will generally also spread the pollution and increase the chance that a nondefeated node contacts a malicious one. The pollution and defeat of nodes spreads very fast, completely in accordance to what can be expected from a gossiping protocol.

There is no way for a nonmalicious node to identify a malicious one as they seem to play fairly; however, as the attackers always pass on the same cache, it is easy for any node to keep track of the last cache provided by a neighbor in order to detect the malicious nodes. Sadly, when a non-malicious node detects the bogus cache replayed by the same neighbor, it is too late to react since the node cache is completely filled with malicious IDs.

In less than 20 cycles, all nonmalicious peers have their cache completely polluted and become defeated. At that point, the malicious nodes may decide to leave the network, leaving it in completely disrupted state without any hope of recovery. It is interesting to note that many gossiping protocols may easily suffer from such an attack. In fact, many distinct gossip protocols differ in just a few details. For example, QuickPeer [CJ05] (a T-Man [JB05] family protocol) may suffer as well.

In this sense, it is somewhat surprising to see how little attention this topic has yet received.

3.2 Attack model

In the following, we will define in detail a generic attack model that is *independent* of specific PSS implementations. The attackers purposefully deviate from the

standard PSS protocol in order to disrupt the service. The reason for doing so can range from obtaining a leader position in order to manipulate applications relying on the PSS, to conducting a massive DoS attack as described in Section 3.1.

In both cases, the main objective of an attacker is to destroy the well-formed peer network topology and cause the formation of a hub set of attackers over which all peers are linked. For example, a successful single attacker can subvert the peer network topology to a star topology with itself as the hub. We decided to promote the formation of $k > 0$ hubs, because it allows the attackers to achieve a strategic control position over the network. Essentially, on the one hand a star topology still ensures connectivity of the overlay graph, while being robust to random failures. On the other hand, it is completely dependent on (vulnerable to) plans of the attackers; the hub topology fulfils the goals sketched in Section 1.3.

The principal method of attack in our generic attack model is injection of fabricated data through the messages gossiped among the participants resulting in the pollution of local caches, which induce the structure of the topology.

Our attack model expects the attackers to operate *intelligently* and not allow themselves to be trivially exposed as the sources of cache pollution. Therefore, the attackers will operate in such a manner so as to be indistinguishable from well-behaved peers under standard operating conditions. While the goal and principal method of attack are generic, actual attacks may use features specific to a particular protocol implementation.

In this setting, the set of actions that an attacker may carry out are as follows:

- **Dropping of identifiers**: attackers may maliciously drop node references from their caches. However, this attack does not pose a serious threat to the peer network whether attackers select dropped references randomly or in collusion with other attackers. The resilience of the peer network to this attack is due to the gossiping scheme used for communication that provides inherent redundancy to the system. As the neighborhood set for each node

continuously changes, an attacker cannot prevent its neighbors from receiving a node reference for more than one cycle.

- **Replay of identifiers**: an attacker can violate protocol-specific constraints (e.g., timestamped-related rules for identifiers in NEWSCAST) in the message exchanges of node references and attempt to diffuse invalid information throughout the overlay. However, this attack also cannot cause serious damage to the network due to reasons given earlier for dropped identifiers attack.

- **Corruption of identifiers**: an attacker may corrupt node identifiers (IDs) to influence maliciously the operation of the specific PSS implementation. The attacker may corrupt selected elements of an ID to achieve a specific anomalous behavior of the protocol. This is a serious attack as gossiping protocols can diffuse these corrupt identifiers rapidly throughout the overlay thus potentially leading to a massive denial-of-service (DOS) attack.

- **Forging of identifiers**: an attacker forges one or more node references to insert into its cache. The difference among the previous kind of action is that the ID is created from scratch, while in the previous case the ID is modified according to a malicious intent. These identifiers could be of actual nodes of the PSS nodes that are other attackers, nodes that are present but not part of the PSS or even fake nodes that do not exist in the overlay. Although the semantics of a PSS require that exchanged caches must contain distinct IDs, such rules are of no use in the absence of active checks on peers (such as pinging to detect liveness). This is a serious attack as well, as gossiping protocols can diffuse these corrupt identifiers rapidly throughout the overlay thus potentially leading to a massive denial-of-service (DOS) attack.

Because of the highly dynamic nature of a PSS the above actions will not cause a serious disruption to the overlay if they are carried out occasionally by a few attackers. While a large number of attackers can destroy the topology with any of

the above attacks, the presence of such a large number of attackers is an unlikely attack scenario. Instead, we consider a practical attack scenario in which there are k malicious peers attacking a PSS with a cache size of c where $k \leq c$. We consider very difficult for an attacker to acquire distinct fake identities, therefore we exclude the threat of a Sybil [Dou02] attack in our model.

Our attack algorithm, which we call the *hub attack*, uses a *combination* of *replay* and *forgery* attacks simultaneously.

We consider the attack scenario in which attackers can cause maximum damage to the peer network utilising the minimum amount of resources for the attack. For this, we assume that malicious peers collude and co-operate. However, this assumption can be removed without any effect on the attack algorithm as it affects only the time needed to complete the attack and not its outcome.

3.2.1 Hub attack algorithm

The basic idea to always replay a message holding the forged ID's of the other malicious peers in the network is perfectly valid from a PSS point of view, as the only mandatory constraint is that cache entries must be distinct. This intrinsic weak constraint complies to reality. For example, in real world P2P file sharing [LNR06, NCW05, RD01], when a peer receives an advertisement for an item, the peer does not check whether the item is available at the advertised location or not; as noted earlier, this allows a malicious peer to appear exactly as any other peer in the network and not arouse suspicions on its behavior.

In the hub attack, each malicious peer maintains a StealthCache structure in which it collects the IDs of well-behaved peers only. This hidden cache structure has no size limit and it is allowed, if necessary, to grow to the size of the network. The StealthCache is initialised with the IDs in the public standard PSS cache at the start of the attack. Then the attacker runs the standard PSS algorithm but with maliciously created messages that contain the IDs of all the other malicious peers. The neighbor nodes to exchange these messages are picked from the

StealthCache. The hub attack algorithm executed by a malicious peer P_m is as follows:

1. P_m pollutes its cache with the IDs of other malicious peers and its own ID. If $k < c$ then P_m fills the remaining entries with nonmalicious IDs taken from its StealthCache.

2. P_m next executes the standard PSS algorithm but randomly selects a non-malicious neighbor (P_n) from the StealthCache.

3. P_m receives the cache of P_n and stores the records in its StealthCache.

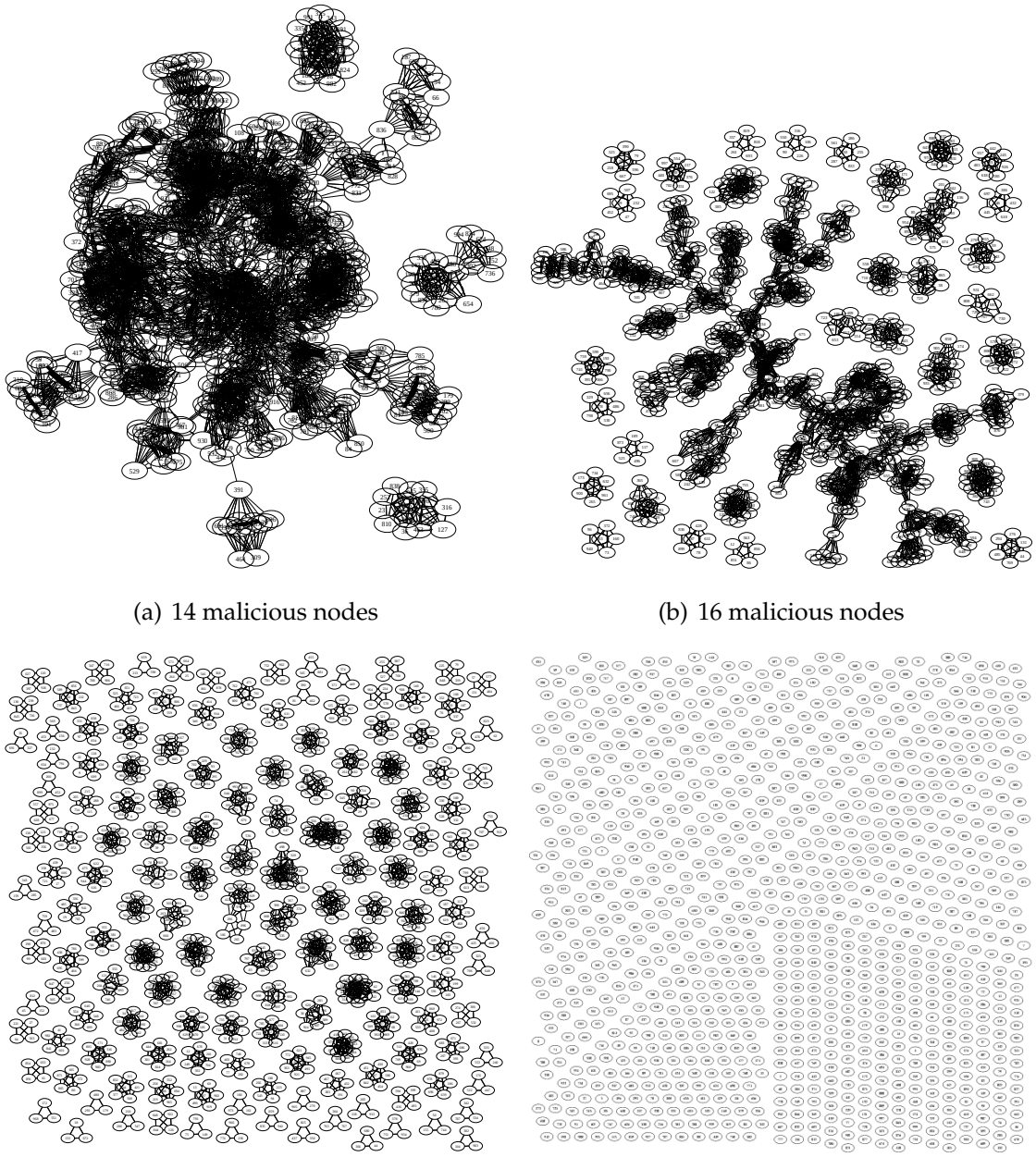
While malicious peers execute the hub attack algorithm, the nonmalicious peers execute the standard PSS algorithm in each cycle. In the above algorithm, a $P_m$will pollute its cache with other malicious peer IDs (MN variant). A variant of the hub attack algorithm has P_m fill up its cache with randomly generated fake IDs (FN variant) when $k < c$.

After executing the algorithm sufficiently for long (e.g., 20-30 gossip rounds in practice), the malicious peers will have formed a hub topology, and effectively now control the network. At that point, different scenarios are possible, including the control of higher level applications, and complete disruption of the PSS by exiting the network.

3.3 Attack evaluation

To validate the effectiveness of our hub attack algorithm described in Section 3.2, we performed a large number of simulation experiments with two goals: (1) to measure the speed of convergence of the network to a destroyed overlay topology and (2) to measure the extent of the damage according to the number of emergent clusters and the percentage of nodes residing outside the main cluster.

The simulations were done on three different overlay sizes of 1,000, 5,000 and 10,000 nodes with the number of malicious peers k ranging from 10 to 20 for a



(a) 14 malicious nodes

(b) 16 malicious nodes

(c) 18 malicious nodes

(d) 20 malicious nodes

Figure 3.2: Overlay mesh status after a hub-attack (MN variant). The PSS cache size is fixed to 20 IDs. When $k = c$ (see 3.2(d)), each node is fully disconnected. Less than 20 gossiping cycles are required to disrupt the PSS. Network size is 1000 nodes.

PSS cache size c set at 20 entries. In the evaluation, we consider both NEWSCAST and basic-shuffling implementations of the PSS. In order to achieve a fair comparison among them, we set $l = c$ in the case of basic shuffling (see Section 2.2.2); in this manner, the amount of information exchanged is equal at each cycle for both implementations. All the individual results described are averaged over 10 separate experiments.

The effects of our hub attack on a 1,000 node overlay are shown in the sequence of images in Figure 3.2. The images from left to right show the increasing damage to the overlay as a function of the number of malicious peers involved (respectively 14, 16, 18 and 20). Each image has been taken just after the exit of malicious peers from the network. As shown in the rightmost image, when $k = c$, the network is completely partitioned. As already discussed in Section 3.1, now for each remaining node, all its neighbors were the malicious peers that no longer exist in the network.

Even for hub attack scenarios where the number of malicious peers are less than the cache size (e.g., 70-75%), the overlay is still severely damaged as can be seen in Figure 3.2(b) with the network partitioned into many distinct clusters. As the PSS is incapable of merging clusters together, recovery of the network requires a reboot process that will restore a random wiring among the clusters after a few cycles.

The emergence of clusters when the caches are polluted to 70-65% of their capacity with malicious peers IDs is shown in Figure 3.3(a). A cache pollution percentage lower than these values leads to a single, very large and highly clustered component. For this kind of damage, the standard PSS can restore the original topology in a short amount of time. Thus, the hub attack in this case is not very serious with only a transient damage and the result holds for any network size. The simulation results also confirm that when $k = c$, the number of clusters is equal to the actual network size. When the number of malicious peers k is closer to the cache size c, say $18 - 20$, then the number of nodes that are located outside the main cluster decreases almost exponentially. This behavior is shown in

Figure 3.3(b). Figure 3.3(c) shows the important result from our experiments on network fragmentation due to a hub attack from a different perspective. Once the small set of malicious peers exit the network, the average size of the biggest cluster tends to be quite small. This behavior is completely opposite to the effect of a massive node crash failure in a PSS overlay in which a giant component is surrounded by few small satellite clusters (see [JGKvS04]).

These results also hold for the basic-shuffling PSS implementation depicted in Figures 3.3(d), 3.3(e) and 3.3(f). Apart from a small variance in the actual values, the results are almost coincident.

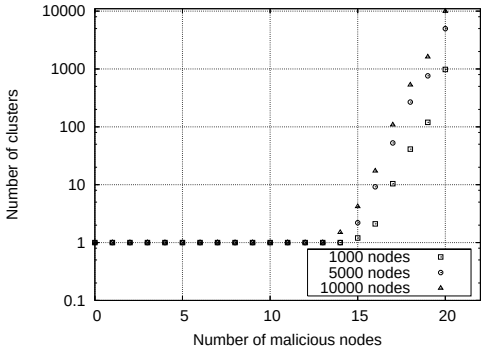
The graphs in Figure 3.4(a) shows the convergence of the overlay to a completely destroyed status in terms of the percentage of nodes that become defeated over time. The smaller 1,000 node network converges rapidly as the cache size of 20 is comparatively large for such a small network. However, the larger overlays also becomes completely destroyed in less than 45 cycles. Note that the y-axis percentage of defeated nodes does not reach 100% as the node count includes the malicious peers also.

However, the FN variant of the hub attack is much more destructive as it can destroy the network with just 4 or 5 malicious peers regardless of the network size or the cache size, as we discuss next. The performance of the FN variant of the hub attack is summarised in Figure 3.4(b) with the y-axis showing the time required to destroy the network and the x-axis indicating the network size. Each line represents a specific number of malicious peers k involved in the attack with remaining cache entries $(c - k)$ filled with fake IDs. As shown, even with only four malicious peers, the network can be defeated in 42 cycles for a 10,000 node network. The differences in time required to defeat the overlay for various network sizes is in general small. Therefore, it can be said that the attack speed is independent of the network size. However, when we use less than 4 malicious peers, the time required to defeat the network begins to increase noticeably. For example, with 2 malicious peers, the time is almost linear to the network size. The reason for this phenomenon is twofold: (1) the sources of infection become

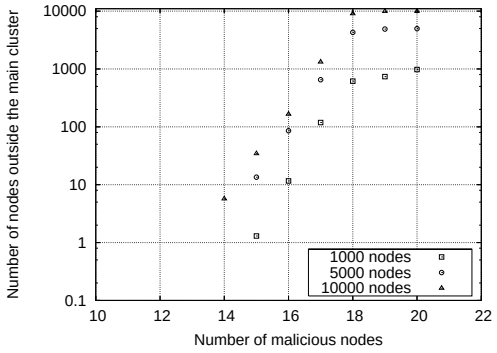
limited and (2) the fake IDs polluting the caches have the effect of reducing the node degree, limiting further chances of contacting a peer and thus slows down the infected PSS. It is interesting to note that the FN variant of the hub attack is able to destroy a network using just a single malicious peer; however, the time needed to complete the attack is too long for it to have any practical impact.

Figure 3.4(c) and 3.4(d) show the same scenario, but using the basic shuffling PSS implementation. Apart from a very small difference in the convergence speed (e.g., a few cycles), the observations we made previously still hold. This speed difference is probably due to the absence of a time-stamps bound to the node IDs in the basic-shuffling algorithm; this makes the pollution to spread a bit slower, as the mechanism with which a new ID is collected is random rather than deterministic (e.g., time-stamp based). However, the difference is almost negligible.

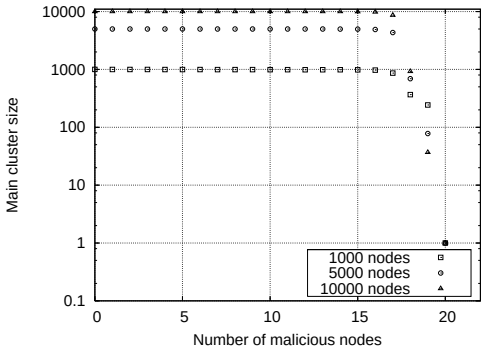
It is important to realise that this rapidly spreading attack has a very high infection rate, thus making it extremely difficult for peers to react in time. In other words, by the time a peer discovers that it is infected, it is already too late to obtain assistance from a neighbor peer, either because the node is already defeated or its neighbors are too polluted to be of any help.



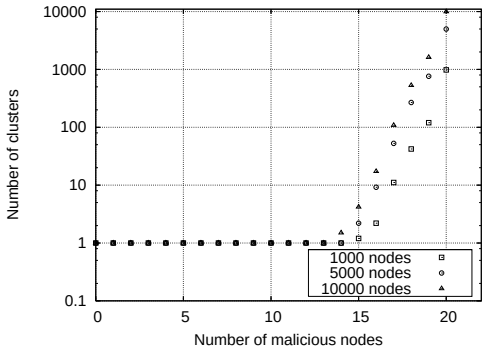
(a) Emergent clusters, NEWSCAST implementation



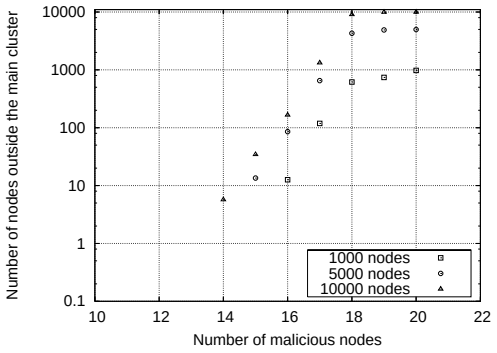
(b) Nodes outside the main cluster, NEWSCAST implementation



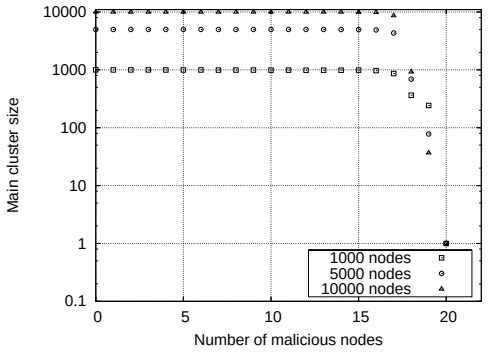
(c) Avg. biggest main cluster, NEWSCAST implementation



(d) Emergent clusters, basic-shuffling implementation

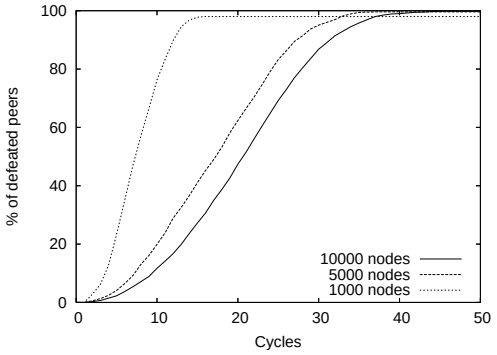


(e) Nodes outside the main cluster, basic-shuffling implementation

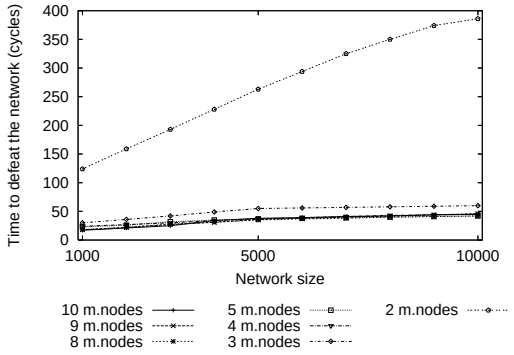


(f) Avg. biggest main cluster, basic-shuffling implementation

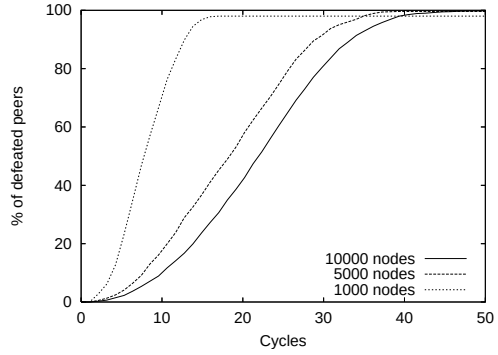
Figure 3.3: Cluster emergence after the exit of all malicious peers. The first three graphs represent the NEWSCAST implementation behaviour, while the others represents the basic-shuffling implementation.



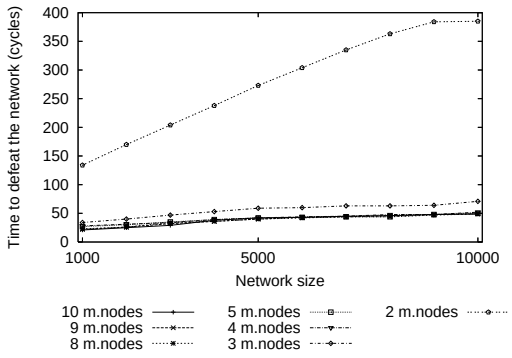
(a) NEWSCAST PSS: MN attack, 20 mal. nodes



(b) NEWSCAST PSS: FN attack, mal. nodes $\in [2:10]$



(c) Basic-shuffling PSS: MN attack, 20 mal. nodes



(d) Basic-shuffling PSS: FN attack, mal. nodes $\in [2:10]$

Figure 3.4: (a),(c) Convergence to the defeated network using 20 malicious nodes adopting the MN attack variant; using a number of malicious nodes lower than the cache size c, the nodes caches cannot be completely polluted and thus, not even a single node is defeated. The results are shown for NEWSCAST and basic-shifling respectively. (b), (d) The time required to defeat the network using the FN attack variant; distinct network sizes are shown and each line represents a specific

Chapter 4

Effects on protocols other than the PSS

From the previous chapter, it is clear that the hub attack has a devastating impact on the PSS and on its crucial functionality.

When the PSS becomes completely partitioned, any protocol relying on the PSS becomes completely isolated as well. Other protocols instead, using the PSS as a bootstrap facility to build their own overlay, may have less trouble; however the lack of the PSS functionality would produce problems in dynamic environments if, for example, the peers are periodically sampling the network - i.e. using the PSS - looking for newly joined peers.

Because of these reasons, we still consider the hub attack described in Section 3.2, but the attackers *do not leave the network*. Essentially, the attackers are interested in obtaining a *leader position* to influence and bias the performance or the behavior of a higher-layer protocol or application.

We consider the hub attack in light of three distinct gossip based protocols: (i) an aggregation protocol (see Section 4.1), (ii) the QuickPeer protocol (see Section 4.2) and (iii) the SG-2 management protocol for superpeer networks(see Section 4.3).

The motivation for our choice is the following: the first protocol relies completely on the overlay, provided by the PSS, while the other two are *overlay managers* protocols; they still rely on the PSS for bootstrapping, locating new nodes identifiers or diffusing critical information, but they build their own overlay (i.e.,

the relation "who knows whom") according to their specific constraints.

In the following sections, we will first describe each protocol along with its attack scenario.

4.1 Aggregation protocol

Aggregation [JM04] is a family of fast epidemic-style averaging protocols designed to compute any mean function on a numeric value held at each network node. Essentially, we suppose each node holds a value expressing any relevant characteristic and all nodes are interested into the global mean of this particular characteristic.

The mean function must be in the form:

$$\mathbf{m} = f^{-1}\left(\frac{f(a_1) + \ldots + f(a_n)}{n}\right)$$

where $f(x)$ can be:

$$\begin{aligned} f(x) &= x && \text{average} \\ f(x) &= x^2 && \text{quadratic} \\ f(x) &= \tfrac{1}{x} && \text{harmonic} \\ f(x) &= \ln x && \text{geometric} \end{aligned}$$

The averaging protocol can compute any aggregate function expressed as a function of some means. For example, the *variance* can be computed using the average and the average of squares, the *network size* can be estimated using $\frac{1}{\text{average}}$ and the *sum* can be obtained using the network size times the average.

Aggregation does not manage the overlay wiring, but every node relies on an underlying PSS (see Sections 2.2.1 and 2.2.2) which provides access to a neighbor list cache. Without imposing any particular requirements about the topology management protocol.

To understand how the aggregation works, we discuss how to calculate the averages.

The basic aggregation behavior fits exactly in the epidemic scheme (see Figure 2.1). The generic method UPDATE() returns $(a + b)/2$, where a and b are the values held by node A and B respectively. This computation step is performed by each node at regular intervals (at each simulation cycle). The global value average is not affected, but the variance over all the estimates decreases very quickly after a few interaction steps; the result is that each node reaches an almost exact estimate of the average of all the node's values in the system. The convergence rate decreases exponentially and it proves also highly scalable, in fact it is almost independent from the network size.

In addition, the aggregation protocol is also very robust in case of (massive) node failures. Failures can at most lower the speed of the aggregation process.

As stated in the beginning, an interesting direct consequence of the averaging function is the ability of estimating the network size. This feature can be useful, for example, to fine-tune other protocols according to network size. This function can be achieved using a particular setup at the start of the protocol: essentially, we need the sum of all node values to be exactly 1. In particular, we can set just one node to the value 1, while all the others are set to 0. The average is $1/N$, thus N can be extracted directly.

However, it is not easy to set up values in this fashion in a real P2P distributed system. How to choose the node holding the value 1? Messages can be marked by a unique leader ID; only this leader node can set its value to 1. Nodes could also start multiple aggregation instances to increase the estimation accuracy and each instance must have its own leader node. Unfortunately, this technique, without a central service dedicated to the management of the leader nodes election, can be hardly adopted, as the agreement problem for the leader election is too hard, especially in a large-scale, dynamic environment.

4.1.1 Aggregation under hub attack

A comparison of the performance of the aggregation protocol under attack is summarised in Figure 4.1. We used the averaging version of the aggregation

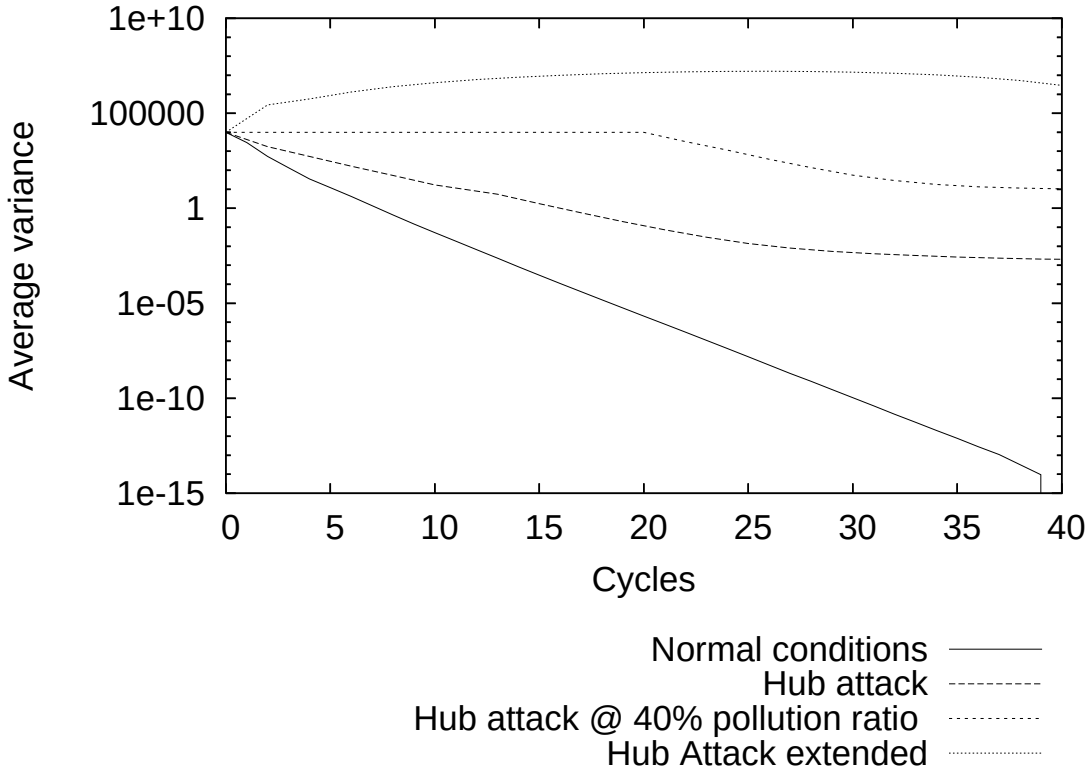


Figure 4.1: Impact of the hub attack on an aggregation protocol. Network size is 10,000 nodes. Distinct scenarios are compared.

protocol family described in the previous section. We measured the average variance of the aggregated values over time. The values to be averaged are assigned to each node by a peak distribution of parameter 10,000 in a bootstrap phase (e.g., before running the protocol). The network size is 10,000 nodes.The cache size c is 20.

Essentially, the goal of this aggregation experiment is to obtain a value of 1 (the average value considering the network size and the initialisation values) in each peer.

The lower line shows the normal behaviour of the aggregation protocol; 20 cycles are sufficient to obtain a negligible difference among the estimates of each peer. In less than 40 cycles instead, the estimates become all equal.

The other lines in the plot shows the behavior of aggregation when the underlying PSS is under a hub attack. The attack is started at the beginning of the simulation, if not specified otherwise.

The next upper line (from the bottom) shows the convergence when the hub attack is running. In all the attack scenarios of this section, 20 ($c = k$) malicious nodes are involved. The effect of the cache pollution is evident. In this particular case, the accuracy (e.g., the variance of the estimates) of the calculation is tolerable, but the protocol takes longer and it never reaches the exact value. In other words, the presence of a poisoned PSS is evident.

As aggregation is usually run at regular intervals (see Section 4.1) to produce a continuous monitoring process, the aggregation protocol is unlikely to start at the same time as the hub attack. In fact, the third line from the bottom shows a slightly different set-up: the aggregation starts when the hub attack has already polluted 40% of the PSS caches (on average). The effect of a biased topology is evident and the protocol accuracy is almost stopped in 15 cycles. As soon as the random graph has turned into a hub topology, the convergence makes no more progress and the variance of the estimates is still high.

Finally, the upper line shows a scenario in which the malicious nodes run also a simple, malicious version of the aggregation protocol; these nodes never average their value and always replay their same value (10,000 in the example). Essentially, this behavior is an adaptation of the hub attack general model to the aggregation specific case. The malicious behavior at the attackers is started at the beginning of the simulation. The convergence diverges in a first phase, then tends to converge (very) slowly towards 5,000. In fact, when the hub topology is emerged, every node is connected to a malicious peer and continuously gets 10000 and divides by 2.

The final example gives an idea of what can be done when the attackers acquire first a leader position (e.g., with the hub topology) and simultaneously inject malicious information in the higher level protocol or service. It is interesting to note that the malicious aggregation behavior would have very little effect with-

out the presence of the biased topology. The combination of the two approaches has a devastating impact. In this case, the hub attack opens the road to much more dangerous attacks and can be seen as a "Trojan horse".

4.2 QuickPeer protocol

QuickPeer [CJ05] is a latency-aware topology manager targeted to un-structured networks. Quick-Peer can effectively build and maintain large scale *latency-aware* overlay topologies. These topologies reflect the underlying IP-level network topology, and provide each peer with the knowledge of the closest (or furthest) neighbor available in the overlay network at a given time, according to network distance (latency).

There are several works that are similar or at least related with QuickPeer.

T-Man [JB05], for example, is a generic, gossip-based framework for managing and building large-scale overlay topologies that inspired our work on QuickPeer. However, in [JB05], T-Man performance is evaluated only on "geometric" topologies (e.g., torus, ring or binary tree). In contrast, we evaluate QuickPeer using more realistic topology models and in dynamic environments to illustrate QuickPeer's self-healing and adaptive behavior. Essentially, Quick-Peer can be considered as an instance protocol in the T-Man framework optimized for proximity topologies using virtual coordinates.

In [RHKS02], the authors propose a scheme to partition overlay nodes into "bins" according to network proximity information. This information is gathered from DNS and delay measures against a set of landmark nodes. Our approach, in contrast, does not need any infrastructure services and exploits a synthetic virtual coordinates system (VIVALDI [DCKM04]) to obtain distance measurements.

In [DGH+87], the authors propose an epidemic protocol, the *Localiser*, to optimize an unstructured overlay network built using SCAMP [GKM01]. Such protocol prove to be scalable and tolerates failures. However, no massive node join scenarios are evaluated.

Several architectures for global distance estimation services that exploit synthetic coordinates have been proposed recently. IDMaps [FJJ+99] and GNP [NZ02] rely on deployment of infrastructure nodes. In contrast, VIVALDI [DCKM04], PIC [CMAP04] and PCoord [LL04] provide latency estimates using distance measurements gathered only between end hosts in the overlay network. We opted for VIVALDI because of its fully distributed nature and simple implementation. However, QuickPeer is not tied to a specific coordinate system and can be used with any of the systems cited above.

In the followings, we introduce the problem of building the latency-aware overlay and the QuickPeer protocol. Section 4.2.2 defines the experimental setup and presents simulation results proving QuickPeer scalability and adaptiveness to dynamic environmental conditions. Finally, we briefly discuss the protocol features and in Section 4.2.4 we test the reaction of QuickPeer under the hub attack.

4.2.1 Latency-aware overlay topology management

System Model In our model, peers communicate via message exchanges, exploiting the connectivity provided by an underlying routed network (i.e., the Internet). Each peer knows a set of other peers (its *neighbors*) that define its local *cache*. As we consider networks of large size, partial membership information at each peer is required for scalability and manageability purposes.

Network distance between peers is modeled using the VIVALDI network coordinate system (see [DCKM04]). Each peer in the overlay has an associated *node identifier* (ID), e.g. a touple ⟨IP address, port, coordinate⟩. Due to this simple ID representation, we assume that peers run on distinct physical hosts (see Section 2.2).

We consider a dynamic overlay network, where peers may join or leave the network at any time (churning).

```
do forever
  wait(Δt)
  neighbor = SELECTPEER()
  SENDSTATE(neighbor)
  peer_cache = RECEIVESTATE()
  my_cache.UPDATE(peer_cache, trimPolicy)
```

(a) Active Thread

```
do forever
  peer_cache = RECEIVESTATE()
  SENDSTATE(peer_cache.sender)
  my_cache.UPDATE(peer_cache, trimPolicy)
```

(b) Passive Thread

Figure 4.2: QuickPeer protocol pseudo code. It fits perfectly in the standard gossip scheme, in fact it is almost identical (see Figure 2.1).

The VIVALDI protocol VIVALDI [DCKM04] is a decentralized, scalable, and efficient protocol developed at MIT. Using VIVALDI, nodes may obtain good coordinates with few RTT probes directed to a small subset of nodes. More importantly, VIVALDI can exploit normal traffic produced by applications using it, without requiring further communication.

The *estimate* of the latency distance between v_i and v_j is denoted $est(v_i, v_j)$. Being estimates, these values may differ from the actual latency. The pairwise error between the estimate and the actual latency can be computed as:

$$\frac{|\, lat(v_i, v_j) - est(v_i, v_j)|}{\min\{est(v_i, v_j), lat(v_i, v_j)\}}$$

where $lat(v, w)$ expresses the latency distance between a pair of nodes (v, w) and represents the average round-trip time (RTT) experienced by communications between them.

In our experiments, the number of dimensions of the virtual space is 5; measuring the error between all pairs of nodes, we found a median error of only 0.14, and a maximum error of 3.5.

The QuickPeer protocol QuickPeer (QP) is an epidemic protocol that, within few gossip exchanges among the participants, provides each peer with the *closest* (or *furthest*) peer identifiers (IDs) available in the overlay. The basic idea underlying the protocol, inspired by the work presented in [JB05], is as follows. Each peer maintains a fixed-size cache holding k IDs. The cache is sorted according to

the network distance estimates provided by VIVALDI coordinates. So, at any time, the first position in the cache holds the closest peer known so far.

At the beginning, QuickPeer caches needs to be initialized with a random sample of nodes taken from the whole overlay. For this purpose, QuickPeer relies on a PSS instance [JGKvS04]. We adopted the Newscast protocol [JKvS03] as a sampling service implementation. Starting from a first random snapshot, QP basically evolves the initial random overlay towards the desired latency-aware topology.

In order to evolve the topology, peers exchange caches in an epidemic fashion. Periodically, each peer actively selects a neighbor and starts a cache exchange process (see pseudo-code in Figure 4.2). Once the remote peer's cache has been received, it is merged with the local one. Note that this merge operation preserves the ordering of the local cache, i.e., newly received IDs are sorted according to the distance from the *local* peer coordinates.

After the caches have been merged, a trimming policy selects the k IDs (out of the possible 2k) that are kept in the local cache. Currently, QP supports two distinct trimming policies:

1. **ClosePolicy(k)**: selects the first k IDs in the cache (i.e., the closest neighbors seen so far);

2. **CloseFarPolicy(k)**: selects the first and the last k/2 IDs in the cache (i.e., the closest and furthest neighbors seen so far).

Merging and trimming operations described above are performed in the `UPDATE()` method shown in Figure 4.2.

In contrast to standard epidemic approaches [JKvS03, EGH+03], QuickPeer randomly picks the neighbor for an exchange only in the first half of the cache (i.e., among the first k/2 IDs). Experimental results [JB05] show that this strategy leads to faster convergence to a latency-aware optimal overlay. In addition, QuickPeer lets each node exchange at most once for each gossip round (actively

or passively). This ensures that, on the average, all peers exchange caches the same number of times during a QuickPeer session.

Failure detection QuickPeer detects failed nodes at picking time. If a neighbor selected for the cache exchange does not answer a probe message in a limited amount of time, it is considered failed and its ID is removed from the cache. In case of massive node failures, however, the second half of the cache will still be populated with references to failed peers, since the picking "cleans" only the first half of the cache. To overcome this limitation, QuickPeer periodically triggers a **cleanCache** procedure that probe peers that appear in the second half of the cache. The frequency at which this procedure is activated may be adaptively tuned to limit the network traffic generated by the probes.

4.2.2 Experimental evaluation

We validate QuickPeer effectiveness in building latency-aware overlay topologies using simulation. Experiments have been performed on Peersim, a Java-based cycle-driven simulator developed in the Bison project [bis]. We consider three different network sizes: 2^{12}, 2^{13} and 2^{14} nodes. Network topologies are generated with the Brite Internet topology generator [bri], using the Waxman algorithm on a flat router model. The output of this phase is a weighted graph, where weights represents latencies between routers in the generated topology. We then run all-pairs shortest paths on the generated graph to obtain a matrix of RTT distance between all pairs of routers in the network. Creating RTT matrix offline speeds up simulations and allows us to simulate larger networks. A VIVALDI simulation is then ran on this data to build the five-dimensional coordinates[1] used in QuickPeer experiments.

The QuickPeer cache size k is set to 40 in all the experiments discussed in this section. An instance of the Newscast protocol boostraps the QuickPeer caches at

[1] According to the version of the VIVALDI protocol we have adopted, five dimensions are sufficient to achieve a very good estimate of the RTT; see [DCKM04]

beginning of the simulations[2].

Our experiments focus on the evaluation of the following QuickPeer aspects:

1. protocol scalability and convergence rate: how well the protocol scales as the network size increase. We measure the convergence rate as the percentage of nodes which hold its closest node reference (ID) in cache.

2. robustness: how QuickPeer reacts to fluctuations in the peer population.

All the results presented here have been averaged over 10 simulation runs.

Static scenario We present experiments that evaluates QuickPeer scalability and convergence rate in static overlays (i.e., no nodes joining or leaving the networks). We presents results obtained using two distinct cache trimming policies: **closePolicy** and **closeFarPolicy**. These results show that QuickPeer scales well and is fast in constructing optimal, large-scale latency-aware topologies.

ClosePolicy Figure 4.3 presents the QuickPeer convergence performance using the **ClosePolicy** trimming policy with parameter $k = 40$. The first thing to note is that QuickPeer convergence rate does not depend on the network size. For all the three scenarios, more than 99.5% of the peers have their closest neighbor in cache at cycle 20. However, QuickPeer reaches 100% optimality only around cycle 60. In fact, as the protocol clusters close neighbors together, it becomes harder and harder for those peers that did not reach optimality to find their closest neighbor.

To improve the convergence speed in the final phase, we implement the following optimization. At each cache exchange, the randomized cache maintained by the underlying PSS is added to the merging process. This optimization yields 100% convergence at about cycle 30, as can be seen in figure 4.3. Note that this

[2]Newscast cache size is also set to 40

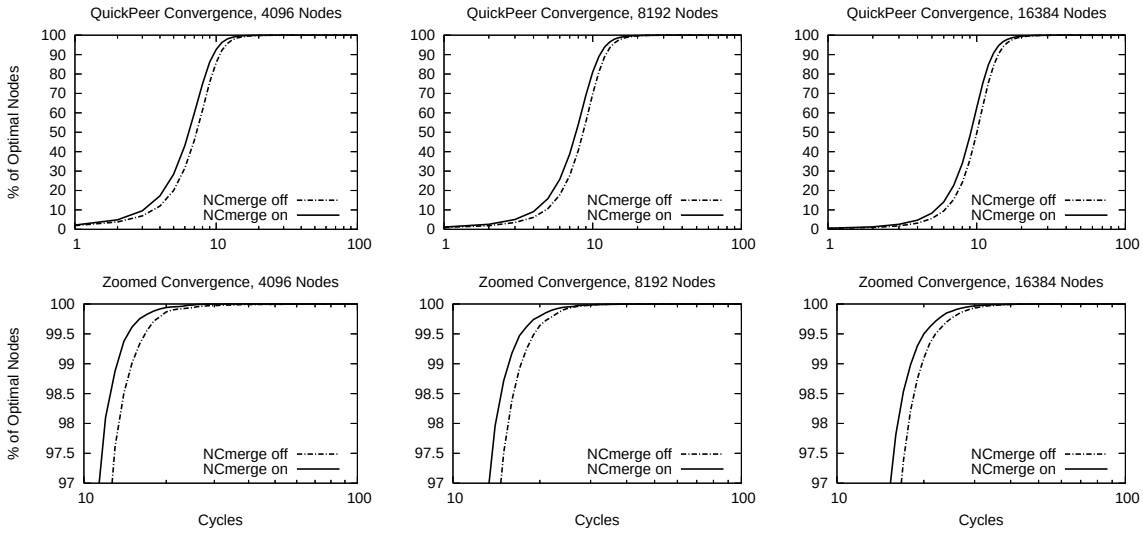


Figure 4.3: Convergence rate expressed in percentage of nodes for each network size. The second line of pictures show the final phase details.

feature comes at no added cost in terms of network usage since the merge process is local at each node. For these reasons, we have decided to keep this feature always on during all the other tests.

CloseFarPolicy Figure 4.4 shows the convergence performance obtained with the **CloseFar** trim policy. With this policy, QuickPeer provides each peer with the closest and the furthest neighbors present in the overlay. As can be seen in Figure 4.4, close and far convergence rate are pretty similar. However, in all our experiments, we experienced that QuickPeer locates more easily furthest nodes in the initial convergence phase.

This policy merges both the effects we would obtain by two distinct instances of the protocol having a Close and "Far" policy, without their overhead.

Dynamic scenarios We present experiments that evaluate QuickPeer scalability and convergence rate in dynamic overlays. We consider a massive node crash scenario in which half of the nodes are killed during QuickPeer convergence phase

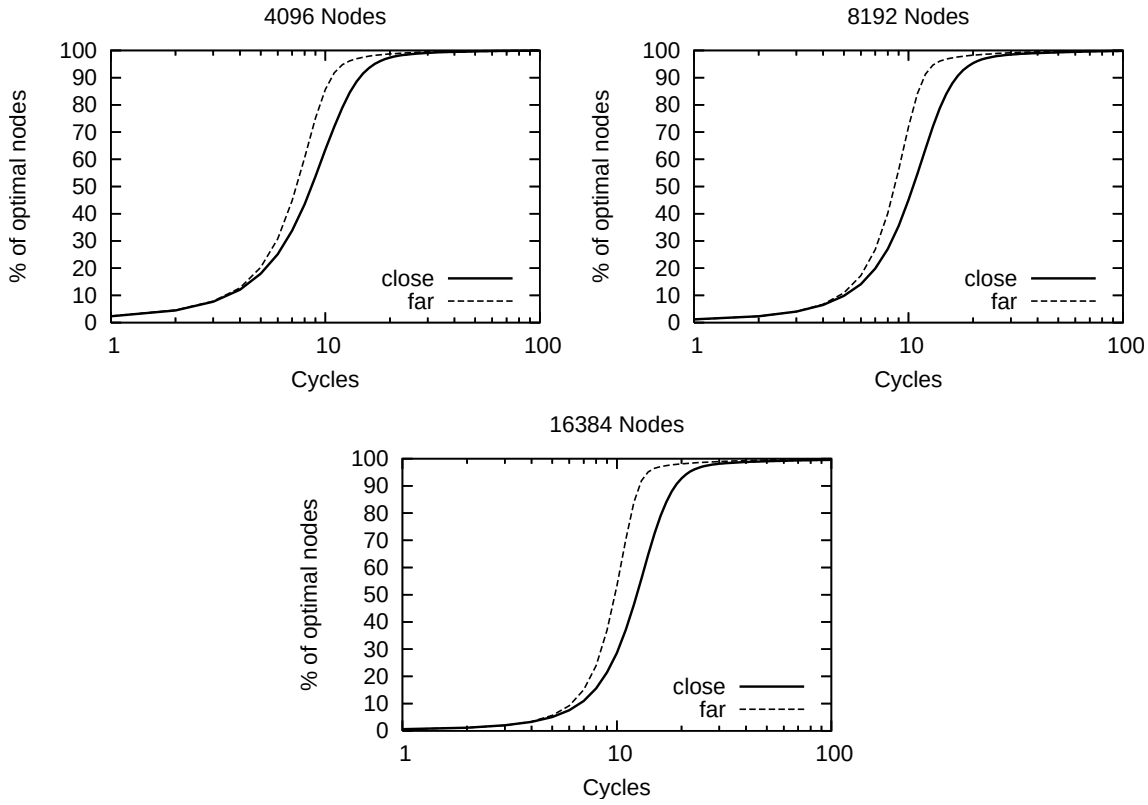


Figure 4.4: QuickPeer convergence performance for each network size. The *Close-Far* policy is used to trim the node caches. Two kinds of optimality are considered: close convergence (standard line) and far convergence (dotted line).

and show that the protocol handles the failures gracefully. In addition, we evaluate QuickPeer behavior in a scenario where a large number of nodes join the overlay during the convergence phase. Even in this case, QuickPeer adapts to the dynamic conditions of the environment.

Nodes crash To evaluate the protocol robustness in case of a massive node crash, we ran the following experiment. The experiment starts with a network of 2^{14} nodes. At cycle 5, right in the middle of the Quickpeer convergence process, 50% of the active nodes fail.

In this catastrophic scenario, QuickPeer is still performing well, as depicted in Figure 4.5. Note that QuickPeer convergence is still increasing even one cycle

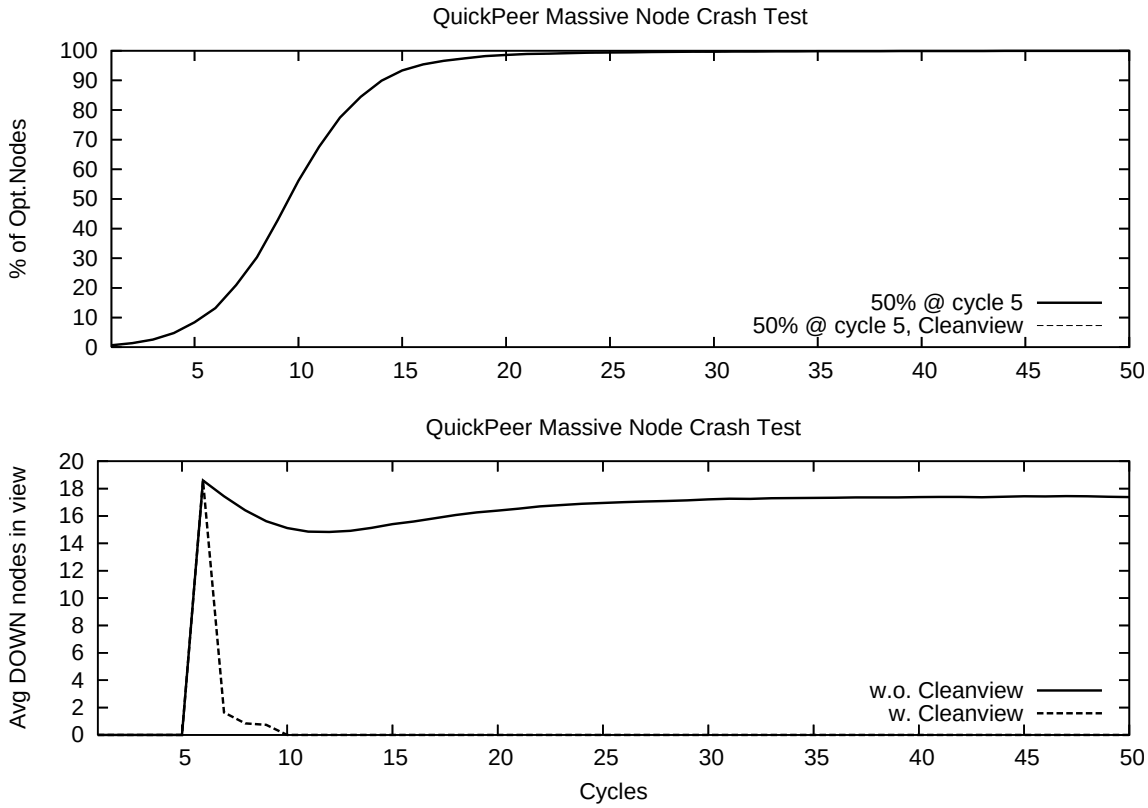


Figure 4.5: Massive crash scenario: 50% of nodes are randomly crashed (removed) at cycle 5. The two sub-figures depict respectively the convergence rate and average node cache pollution per node.

after the massive node crash and optimality is reached in about the 30 cycles. This behavior is expected, since now QuickPeer has an easier job to accomplish given the smaller size of the overlay.

After the node crash, each node holds in its cache, with high probability, references to failed nodes. In this experiment, the cleanCache procedure is triggered every 3 simulation cycles.

The bottom sub-figure in Figure 4.5 shows that, after the crash, the node caches are getting populated with failed IDs. Without the clean cache procedure, the number of failed IDs initially tend to decrease due to the cleaning process associated with the picking in the first half of the cache. However, after a few cycles,

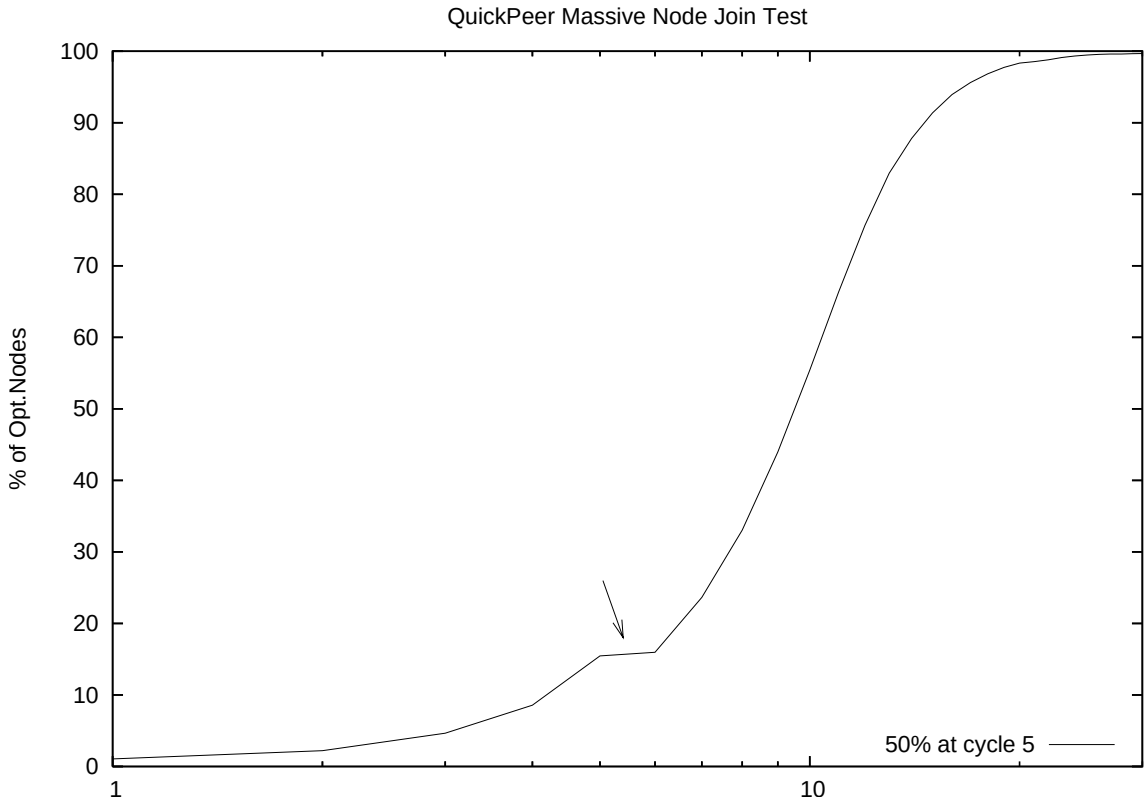


Figure 4.6: Convergence rate for the massive nodes join: starting from 2^{13} nodes network, 4096 new nodes are added at cycle 5. The arrow (between cycle 5-6) indicates a transient slow down in convergence rate due to the massive node join.

the average pollution stabilizes around 45% of the cache size (i.e., it fills nearly half of the cache). The **cleancache** procedure stops this pathological behavior at cycle 10.

Nodes join QuickPeer reaction to a massive node join scenario is depicted in Figure 4.6. The experiment starts with an overlay network of 8192 nodes. At cycle 5, 4096 new nodes join the overlay.

The convergence rate slows down at cycle 5, just after the massive join. After this step, the rate grows exponentially as in previous experiments until full

convergence is achieved at cycle 30.

4.2.3 QuickPeer discussion

The QuickPeer topology may be useful for several distributed applications, like distributed online gaming, context-aware P2P applications and QoS-aware publish/subscribe systems. The distinctive feature of QuickPeer is that it can manage large scale overlay topologies providing each host in the overlay with its closest or furthest neighbor, according to network distance (RTT), in few gossip rounds.

Experimental results proves Quickpeer scalability, robustness to failures and adaptiveness to scenarios in which large numbers of nodes join the overlay concurrently.

4.2.4 QuickPeer under hub attack

We have compared the performance of the QuickPeer (QP) topology manager protocol running over a corrupted PSS layer. We adopted a network of 8,192 nodes whose latency model has been generated by the Brite Internet topology generator according to the procedure described in Section 4.2.2.

Figure 4.7 shows the QuickPeer's performance (e.g., in terms of the percentage of the nodes successfully arranged in a latency-aware fashion) according to increasing levels of the PSS cache pollution.

With a pollution in the range $[0 : 80[$, QP is almost not affected. We can expect this result as QP starts from a copy of the current PSS cache and evolves by gossiping its own cache. If the starting cache is sufficiently random, then the protocol can wire its own latency-oriented topology.

In our example network, the QP latency-aware topology starts degrading from an 80% PSS pollution level, however the performance gap is negligible in this case. The performance drop is much more evident with the 85% and 90% levels of pollution in which the starting network is severely clustered and nodes

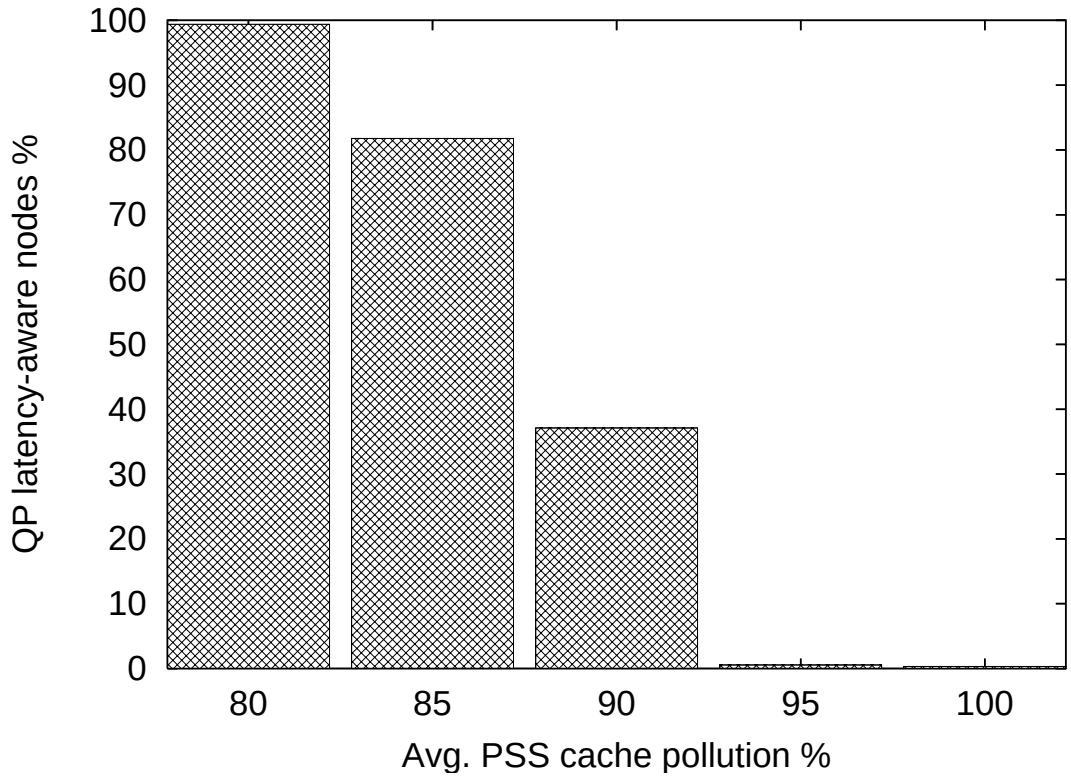


Figure 4.7: Impact of the hub attack over the QuickPeer protocol. Network size is 8,192 nodes.

are trapped in their local neighborhood. Higher levels of pollution lead to an obvious almost null result (e.g., no nodes hold the closest neighbor in cache).

The hub attack affects the QP protocol only when the PSS topology is close to the hub topology; however, this is still a serious threat. Essentially, it means that any QP instance started 25 or more cycles later than a hub attack (over the PSS used by the QP instance), will have troubles or will not be able to generate the desired topology at all.

In addition, the hub topology generated by the attack will prevent QP from perceiving new node arrivals in a dynamic environment, as all newcomers will join directly the malicious nodes.

The hub attack model is general enough to be applied directly to other topology manager algorithms such as QP. In other words, now we consider to apply the hub attack and its malicious behavior to the nodes playing the QP protocol instead of the PSS. In fact, both QP and the PSS are very close to the basic gossip scheme and just a small protocol-specific adaptation is required. The following example shows how flexible our attack model actually is.

While the ordinary attack for the PSS involves forging the timestamps of the IDs in the message (see Section), here we also forge the coordinates of the malicious nodes when they exchange the cache with a neighbor. The coordinate must be close as possible to the coordinate of the neighbor in order to be accepted by the neighbor in its next cache update as the closest node so far. Each attacker coordinate (with n dimensions) can be forged in the following manner:

$$\begin{aligned} xa_1 &= xn_1 \pm rnd(\epsilon) \\ xa_2 &= xn_2 \pm rnd(\epsilon) \\ &\vdots \quad \vdots \quad \vdots \\ xa_n &= xn_n \pm rnd(\epsilon) \end{aligned}$$

where xa_i and xn_i are respectively the i^{th} element of the attacker and neighbor coordinate; $rnd(\epsilon)$ is a function that returns a random number in the range $\in [0 : \epsilon]$. The value of ϵ can be selected according to the actual coordinate distribution of the network (latency-distribution).

This simple adaptation leads to the emergence of a hub topology instead of the QP latency-aware overlay. The same approach can be applied to the T-Man [JB05] protocol family.

The hub attack can be successfully applied to the QP protocol independently of the presence of a healthy or poisoned PSS. In fact, although the PSS is healthy and QP takes cache snapshots from it finding new potential neighbors, the neighbors coordinate will hardly be closer to the current node than the attacker's coordinate (if ϵ has been carefully selected). Thus, the attacker IDs will continue to persist in the node's cache.

4.3 SuperPeer protocol

Modern P2P networks present several unique aspects that distinguish them from traditional distributed systems. Networks comprising hundreds of thousand of peers are not uncommon. A consequence of such scale is extreme dynamism, with a continuous flow of nodes joining or leaving. Such characteristics present several challenges to the developer. Neither a central authority nor a fixed communication topology can be employed to control the various components. Instead, a dynamically changing overlay topology is maintained and control is completely decentralized. The topology is defined by ”cooperation” links among nodes, that are created and deleted based on the requirements of the particular application.

As we stated in Chapter 1, the choice of a particular topology is a crucial aspect of P2P design.

A distinct, but related problem regards roles that nodes may assume: original P2P systems were based on a complete “democracy” among nodes: “everyone is a peer”. But physical hosts running P2P software are usually very heterogeneous in terms of computing, storage and communication resources, ranging from high-end servers to low-end desktop machines.

The superpeer paradigm is an answer to both issues [Gnu, fas]. It is based on a two-level hierarchy: *superpeers* are nodes faster and/or more reliable than “normal” nodes and take on server-like responsibilities and provide services to a set of *clients*. For example, in the case of file sharing, a superpeer builds an index of the files shared by its clients and participates in the search protocol on their behalf. Superpeers allow decentralized networks to run more efficiently by exploiting heterogeneity and distributing load to machines that can handle the burden. On the other hand, this architecture does not inherit the flaws of the client-server model, as it allows multiple, separate points of failure, increasing the health of the P2P network.

The superpeer paradigm is not limited to file sharing: it can be seen as a

general approach for P2P networking. Yet, the structural details are strongly application-dependent, so we cannot identify a "standard" superpeer topology.

In this paper, we focus our investigation on a specific aspect of the problem: *proximity*. Our goal is to build a topology where clients and superpeers are related based on their distance (in terms of communication latency). The idea is to select superpeers among the most powerful nodes, and to associate them with clients whose round-trip time is bounded by a specified constant. This is a generic problem, whose solution can be beneficial to several P2P applications. Examples include online games such as Age of Empires [BT01], P2P telephony networks such as Skype [Sky] and streaming applications such as PeerCast [peea]. In all these cases, communication latency is one of the main concerns.

Our solution, called SG-2, is a self-organizing, decentralized protocol capable of building and maintaining superpeer-based, proximity-aware overlay topologies. SG-2 uses an epidemic protocol to spread messages to nearby nodes, and implements a task allocation protocol that mimics the behavior of social insects. These biology-inspired mechanisms are combined to promote the "best" nodes to the superpeer status, and to associate them to nearby clients.

To validate the results of our protocol, we considered a specific test case: *online games*. In these applications, a large number of players interact together (or against each other) in virtual worlds. Most online games follow a classic client-server model, but we believe that the superpeer paradigm could represent an interesting alternative. We envision a system where a small number of powerful nodes act as state servers when needed, with the remaining ones acting as clients. All nodes run the same code and can switch from the first role to the second when needed. Thus, superpeers dynamically change over time, depending on the environment conditions.

4.3.1 System Model

The system model for our superpeer scenario is quite similar to the one described in Section 4.2.1; however, the following extensions are required.

Nodes are heterogenous: they differ in their computational and storage capabilities, and also (and more importantly) with respect to the bandwidth of their network connection. To discriminate between nodes that may act as superpeers and nodes that must be relegated to the role of clients, each node v is associated with a *capacity* value $cap(v)$, that represents the number of clients that can be handled by v. To simplify our simulations, we assume that each node knows its capacity. In reality, this parameter is strongly dependent on the specific application, and can be easily computed on-the-fly through on-line measurements.

Besides capacity associated to each single node ("how many"), another parameter to be considered is the end-to-end latency between nodes ("how well"). In our model, each pair of nodes (v, w) is associated with a *latency distance* $lat(v, w)$, representing the average round-trip time (RTT) experienced by communications between them. The latency distance between a specific pair of nodes may be measured directly and precisely through ping messages, or approximately estimated through a *virtual coordinate service* [DCKM04]; given the dynamic nature of our system and the large number of nodes to be evaluated as potential neighbors, we will adopt the latter approach.

4.3.2 The Problem

Generally speaking, our goal is to create a topology where the most powerful nodes (in terms of capacity) are promoted to the role of superpeers, and the association clients/superpeers is such that each client obtains a configurable *quality of service* (in terms of latency distance) from its superpeer.

More formally, we define the problem of building a proximity-aware, superpeer-based topology as follows. At any given time, the problem input is given by the current set of nodes $\mathcal{V}$, and the functions $cap()$ and $lat()$ defined over it. Furthermore, a global parameter *tol* expresses the maximum latency distance that can be tolerated between clients and superpeers. The constraints describing our target topology are the following:

- each node is either a superpeer or a client;
- each client c is associated to exactly one superpeer s (we write $super(c) = s$);
- the number of clients associated to a superpeer s does not exceed $cap(s)$;
- given a superpeer s and one of its clients c, we require that $lat(s, c) \leq tol$.

To avoid ending up with a set of disconnected, star-shaped components rooted at each superpeer, we require that superpeers form another proximity-based overlay: two superpeers are connected if their latency distance is smaller than $tol + \delta$, where δ is another configuration parameter.

We aim at selecting as few superpeers as possible (otherwise, the problem could be trivially solved by each node acting as a superpeer, with no client/superpeer connections). This choice is motivated, once again, by the particular scenario we are considering: in online games, superpeers manage the distributed simulation state, so centralizing as many decisions as possible is important from the performance point of view. Note that given the dynamism of our environment, obtaining the minimum number of superpeers may be difficult, or even impossible. But even in a steady state, the resulting optimization problem is NP-complete.

4.3.3 The SG-2 Protocol

The architecture of SG-2 is shown in Figure 4.8; here, we briefly describe the rationale behind it, leaving implementation details to the following subsections.

Our solution to the problem described above is based on a fundamental observation: measuring precisely the RTT between all pairs of nodes (e.g., through pings) is extremely slow and costly, or even impossible due to topology dynamism. To circumvent this problem, and allow nodes to estimate their latency without direct communication, the concept of *virtual coordinate service* has been developed [DCKM04]. The aim of this service is to associate every node with a synthetic coordinate in a virtual, n-dimensional space. The Euclidean distance between the coordinates of two nodes can be used to predict, with good accuracy,

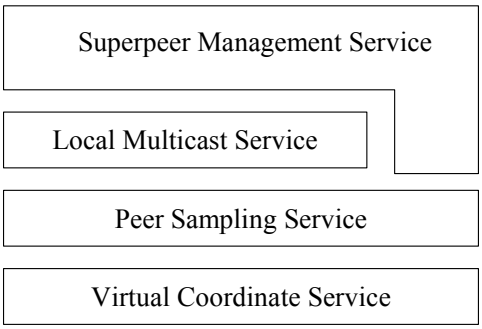


Figure 4.8: The set of services composing the SG-2 architecture.

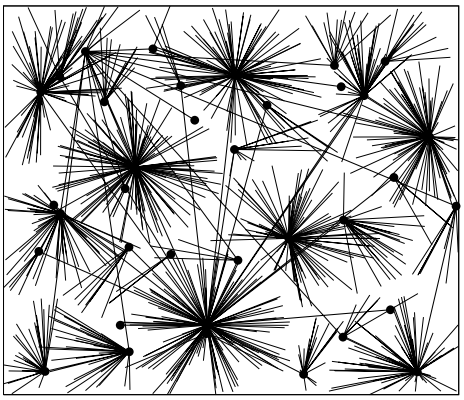


Figure 4.9: A superpeer topology in a bi-dimensional virtual space, where Euclidean distance corresponds to latency.

the RTT between them; in other words, it is sufficient for two nodes to learn about their coordinates to estimate their latency, without direct measurements.

Our problem may be redefined based on the concept of virtual coordinates. Nodes are represented by points in the virtual space; each of them is associated with an *influence zone*, described as a n-dimensional sphere of radius *tol* centered at the node. Our goal is to cover the virtual space with a small number of superpeers, in such a way that all nodes are either superpeers or are included in the influence zone of a superpeer. Figure 4.9 shows the topology resulting from the execution of SG-2 in a bi-dimensional virtual space.

Nodes communicate with each other using a *local broadcast service*, whose task is to efficiently disseminate messages to nodes included in the influence zone of the sender. This service is used by powerful nodes to advertise their availability to serve as superpeers, and by ordinary nodes to seek superpeers whose capacity has not been saturated yet.

The main component of SG-2 is the *superpeer management service*, which selects the superpeers and associates clients to them. The protocol is heavily inspired by the behavior of social insects [BDT99], such as ants or bees, that have developed

very sophisticated mechanisms for labor division. In summary, such mechanisms work as follows. In a totally decentralized fashion, specialized groups of individuals emerge, with each group aimed at performing some particular task. The task allocation process is dynamic and follows the community needs according to changes in the environment. The stimulus to perform some kind of task or to switch to another one can be given by many factors, but it is normally given by high concentrations of chemical signals, such as pheromones, that are released by other individuals and are spread in the environment. Each individual has its own response threshold to the stimulus and reacts accordingly.

The superpeer protocol mimics this general picture. Un-associated nodes diffuse a "request for superpeers" signal through local broadcasts; the signal concentration in the network may stochastically trigger a switch to the superpeer role in some nodes according to their response threshold, which is proportional to their capacity. On the other hand, powerful nodes covering the same area of the virtual space compete with each other to gain new clients, by signaling their availability through local broadcasts. Clients associate themselves to the most powerful superpeers, and superpeers with an empty client set switch back to the client role. The combination of these two trends (the creation of new superpeers to satisfy client requests and the removal of unnecessary superpeers) finds its equilibrium in a topology that approximates out target topology.

The last component to be addressed is the *peer sampling service*. As we already know the features of this service from the previous chapters, we just remind that the task of this protocol-layer is to provide each node with a view containing a random sample of nodes [JGKvS04].

Virtual Coordinate Service

In SG-2, the virtual coordinate service is provided by VIVALDI [DCKM04], which has been already described in Section 4.2.1.

Local Broadcast Service

Unlike previous layers, based on existing protocols, the local broadcast service has adapted an existing protocol for the specific needs of SG-2 [EGH+03]. Each message m is associated with the sender identifier s_m and a radius parameter r_m. Message m is delivered to all those nodes that are within latency distance r_m from s_m, as estimated by VIVALDI. Hence, the name SPHERECAST.

The protocol may be described as follows. When a node either receives a message or wants to multicast a new one, it forwards it to its local *fan-out*. The fan-out of node v for message m is given by the subset of neighbors known to v that are potentially interested in the message, i.e. whose distance from s_m is not larger than r_m. SPHERECAST does not maintain its own topology; instead, it relies on the underlying overlay network provided by the peer sampling service.

When a message is originated locally, or it is received for the first time, it is forwarded immediately to all nodes in the fan-out. If a message has been already received, a node may stochastically decide to drop it (i.e., not forwarding it). This is a standard approach used to avoid flooding the network. A strict deterministic approach such as dropping any multiple copy would not work correctly due to the nature of the underlying overlay. The actual clustering coefficient of the underlying topology and the continuous rewiring process may stop the message spreading. The stochastic approach solves this issue in a straightforward manner.

The probability of dropping a message is given by the following formula: $p = 1 - e^{-s/\vartheta}$, where s is the number of times the node has seen this message and ϑ is a response threshold parameter. In this way, when a packet is received multiple times by a peer, it has less and less probability to be forwarded again. From an implementation point of view, digests of received messages are stored in a per-node table, together with the number of times that specific message has been received. This table is managed with a LRU policy, to avoid unbounded growth.

Superpeer Management Service

This layer is the core component of SG-2. Nodes participate in this protocol either as superpeers or as clients; a client c may be either associated to a superpeer ($super(c) = s$), or actively seeking a superpeer in its *tol* range ($super(c) = \perp$). At the beginning, all nodes start as clients; to converge to the target topology defined in Section 4.3.2, nodes may switch role at will, or change their client-superpeer relationship. The decision process is completely decentralized.

Each node v maintains the following local variables. *role* specifies the role currently adopted by v; $role = \text{SP}$ if v is a superpeer, $role = \text{CL}$ otherwise. *clv* and *spv* are two views, respectively containing the clients and the superpeers known to v. They are composed of node descriptors combining an identifier w and a logical time-stamp ts_w; the latter is used to purge obsolete identifiers, as in NEWSCAST. When v acts as a superpeer, *clv* is populated with the clients currently associated to v; it is empty otherwise. The size of *clv* is limited by $cap(v)$. *spv* contains descriptors for the superpeers that are in $tol + \delta$ range; its size is not explicitly limited, but rather is bounded by the limited number of superpeers that can be found within $tol + \delta$ distance. When v acts as a client, one of the descriptors in *spv* may be the associated superpeer of v.

Two distinct kinds of messages are broadcasted using SPHERECAST: CL-BCAST and SP-BCAST. The former are sent while in client state and are characterized by a radius parameter r_m equal to *tol*, i.e. the maximum tolerated latency. The latter are used in superpeer state and their radius parameter is equal to $tol + \delta$; superpeers need a wider radius to get a chance to contact other superpeers; furthermore, nodes with overlapping influence zones can exchange clients if they find a better client allocation that reduces their latency.

At each node, two threads are executed, one active and one passive. The execution of active threads may be subdivided in periodic *cycles*: in each cycle, superpeers emit a SP-BCAST signal which is broadcast in the surrounding area, to notify nodes about their presence and its residual capacity. Clients, on the other hand, periodically emit CL-BCAST messages if and only if they are not associated

to any superpeer. The shorter the cycle duration, the faster the system converge to the target topology; but clearly, the overhead grows proportionally. The passive threads react to incoming messages according to the message type and the current role. Four distinct cases are possible:

Superpeer v **gets** $\langle \text{SP-BCAST}, s, ts_s, cap(s) \rangle$: the pair (s, ts_s) is inserted in *spv*. If s was already present, its time-stamp is updated. After that, the capacity of the two supernodes is compared: if $cap(v) > cap(s)$, then a migration process is started. Clients associated with s that are inside the influence zone of v migrate to v, until the capacity is exhausted. Each affected client is notified about the new superpeer (v) by the current superpeer s. Node s, if left with no clients, switches back to the client role; it associates itself to v, if $est(v, s) \leq tol$ and v has still residual capacity; otherwise, it starts emitting CL-BCAST messages.

Superpeer v **gets** $\langle \text{CL-BCAST}, c, ts_c \rangle$: if $|clv(v)| < cap(v)$ (the capacity of v has not been exhausted), the client node is associated to v (unless, given the asynchrony of messages, it has been already associated with another superpeer).

Client v **gets** $\langle \text{SP-BCAST}, s, ts_s, cap(s) \rangle$: the pair (s, ts_s) is inserted in *spv*. If s was already present, its time-stamp is updated. If v is not client of any superpeer, it sends a request to s asking to be associated with it. The response may be negative, if s has exhausted its capacity in the period between the sending of the message and its receipt by v. On the other hand, if v is already client of another superpeer s' and $cap(s) > cap(s')$, then it tries to migrate to the more powerful superpeer. This strategy promotes the emergence of a small set of high-capacity superpeers.

Client c **gets** $\langle \text{CL-BCAST}, c, ts_c \rangle$: This kind of messages can trigger a role change from client to superpeer; it is the cornerstone of our approach. The willingness of becoming a superpeer is a function of a node threshold parameter and the signal

concentration perceived by a node in its influence area. The switching probability can be modeled by the following function:

$$P(role(v) = \text{CL} \rightarrow role(v) = \text{SP}) = \frac{s^2}{s^2 + \theta_v^2}$$

where s is the signal magnitude and θ_v is the response threshold of node v. This function is such that the probability of performing a switch is close to 1, if $s \gg \theta$, and it is close to 0 if $s \ll \theta$. If c_{max} is the maximum capacity, θ_v is initialized with a value which is $c_{max} - cap(v)$; in this way, nodes with higher capacity have a larger probability of becoming superpeers. The maximum capacity may be either known, or it can be easily computed by an *aggregation* protocol in a robust and decentralized fashion [JMB05].

After the initialization, in order to make the topology more stable and avoid fluctuations, the response threshold is modified in such a way that time reinforces the peer role: the more time spent as a client, the less probable it is to change role. Once again, the inspiration for this approach comes from biology: it has been observed, for example, that the time spent by an individual insect on a particular task produces important changes in some brain areas. Due to these changes, the probability of a task change (e.g., from foraging to nursing) is a decreasing function of the time spent on the current task [BDT99]. For this reason, θ_v is reinforced as follows:

$$\theta_v(t) = \theta_v(t-1) + (\alpha \cdot (t - t'_v))$$

Where t is the current cycle and t'_v is the last cycle in which v became a superpeer; α is a parameter to limit or increase the time influence. The peer normal responsiveness is re-initialized based on its local capacity if its superpeer crashes or if it becomes a superpeer node.

The reaction to CL-BCAST messages is the only mechanism to allow a client to become a superpeer. A superpeer can switch back to the client role only when other higher capacity superpeers have drained its client set. The θ adaptation process is only active when a node is in the client state.

4.3.4 Experimental results

We performed a large number of experiments based on simulation to validate the effectiveness of our approach. The goal of our experiments was twofold: first of all, we measured the speed of convergence in a stable overlay, in the absence of failures; second, we measured the robustness of our approach in a dynamic environment, where a fixed percentage of nodes are substituted with fresh ones periodically. Any node in the network can be affected by substitution, regardless of its role. Unlike the real world, where a superpeer is supposed to be more reliable, our choice is stricter and more "catastrophic". Finally, communication overhead has been measured. The experiments have been performed using Peersim [peeb].

In our experiments, network size is fixed at 1000 and 2000 nodes. Several kinds of networks have been considered, but here, due to space restrictions, the focus is on *gaming-oriented* scenario [ZS04, SGB+03]. Other scenarios present similar results. For each pair of nodes v, w, the latency distance $lat(v, w)$ among them has been generated using a normal distribution with average value $\mu = 250\,ms$ and variance $\sigma = 0.1$ [ZS04]. Then, we have run VIVALDI on this network, obtaining the corresponding function $est(v, w)$. In the corresponding virtual space, we have considered *tol* values of $200\,ms$, $250\,ms$ and $300\,ms$, which are typical of strategy and role-playing games. We have experimented with δ values of $200\,ms$, $300\,ms$ and $400\,ms$, corresponding to typical round-trip time that can be accepted for superpeer communication. The capacity function $cap()$, i.e. the maximum number of clients that can be served, is generated through an uniform distribution in the range $[1 : 500]$. The simulation is organized in synchronous *cycles*, during which each node has the possibility to initiate a gossip exchange; note, however, that in reality node do not need to be synchronized. All the results are averaged over 10 experiments.

Figure 4.10 illustrates the behavior of the protocol over time. All the figures in the left column are obtained in networks whose size is 1000 nodes, while the figures in the right column are relative to networks with size equal to 2000 nodes.

The content of each sub-figure is divided in two parts; in the main plot, the number of superpeer active at each cycle is shown; in the small frame inside the main plot, the percentage of clients that are already associated is shown. In these experiments, the network is static; no nodes are removed or added.

Figure 4.10(a) depicts a rather bad situation: in both network sizes, the convergence is extremely slow, and the number of nodes that are satisfied is low. This bad performance is motivated by the characteristics of the latency distributions [ZS04, SGB+03] and the tolerance value selected; most of the node pairs have a higher latency than 200 *ms*, and thus SG-2 cannot help much. Figure 4.10(b) shows a much better situation: a large percentage of clients (between 94% and 100% depending on size and parameter δ) have been associated after only few cycles (10-20). The number of superpeers is also very small, after an initial peak due to a large number of clients reacting to the signal. Almost every client can reach the required latency because 250 *ms* is the average pairwise latency in our game-like coordinates distribution. However, some nodes lies outside the 250 *ms* border and it is challenging for SG-2 to accommodate those nodes. The node density plays an important role for SG-2. In fact, the bigger network can be fully organized in a latency-aware fashion using the wider superpeer communication range ($\delta = 400\,ms$). Figure 4.10(c) shows the performance for $tol = 300\,ms$: a response time that is perfectly acceptable in a strategic/role playing game scenario. The latency-aware topology in the figure is very good with any δ value. We obtain 100% of in range clients with about 50 superpeers in the small network and about 63 in the bigger network, in less than 10 cycles.

Figure 4.11 is aimed at illustrating the robustness of our protocol. The size of the network is fixed at 1000 nodes. Its composition, however, is dynamic: at each cycle, 10% or 20% of the nodes crashes and are substituted with new ones. The figure shows that the number of superpeers oscillates over time, as expected, and that up to 80% and 70% of the clients are associated to superpeers. The nodes that are not associated are those that have been recently created and are trying to find a position in the topology.

Finally, we discuss message overhead; due to space limitations, we provide summary data instead of plots. We have measured the number of broadcast messages, including both CL-BCAST and SP-BCAST. Since the former type of message is broadcast only in case of lack of satisfaction, only a small number of them are generated: on average, less than 2 messages every thousand nodes. Superpeers, on the other hand, continuously send one message per cycle.

4.3.5 SG-2 discussion

The superpeer approach to organize a P2P overlay is a trade-off solution that merges the client-server model relative simplicity and the P2P autonomy and resilience to crashes. The need for a superpeer network is mainly motivated by the fact to overcome the heterogeneity of peers deployed on the Internet.

Yang and Garcia Molina [YGM03] proposed some design guidelines. A mechanism to split node clusters is proposed and evaluated analytically, but no experimental results are presented.

Superpeer solutions proved to be effective solutions in the real world: Kazaa / Fasttrack [fas] and Skype [Sky] are two outstanding examples. However, their actual protocols are not publicly available and they cannot be compared with any other solution or idea. At the time of writing, only a few works [LRW03, Sky] describe some low-level networking details.

The SG-2 protocol can be considered as a natural evolution of the SG-1[Mon04] protocol; the two solutions, however, cannot be directly compared from a performance point of view because of their different goals. SG-1 focuses on optimizing the available bandwidth in the system, while SG-2 introduces the notion of latency between peer pairs and poses a QoS limit on it. The definition of the target topology is straightforward in SG-1 (e.g., the minimum number of superpeers to accommodate all the peers according to the superpeer capacities), while it is a NP-problem in the SG-2 case. From the architectural point of view, they both rely on the existence of an underlying random overlay. The superpeer overlay is gen-

erated on top of it. The superpeer election process in SG-2 is strongly bio-inspired and much more randomized than approach used in SG-1.

In [SH06], the authors propose a socio-economic inspiration based on Shelling's model to create a variation of the super-peer topology. Such variation allows the ordinary peers to be connects with each other and to be connected to more than one super peer at the same time. This topology focuses on efficient search. As in our case, the superpeers are connected to each other to form a network of hubs and both solutions are suited for unstructured networks. However, they do not address the problem of the superpeer election.

The basic problem of finding the best peer, having the required characteristics, to accomplish some task (e.i., the superpeer task) is addressed in a more general form in [AKR+05]. The problem is referred as "optimal peer selection" in P2P downloading and streaming scenarios. The authors use an economics inspired method to solve the optimization problem; the developed methodologies are general and applicable to a variety of P2P resource economy problems. The proposed solution is analytically strong, but no experimental results are shown especially regarding a large and dynamic scenario as the one the authors are addressing.

Our implementation is based on VIVALDI (see section 4.3.3), but it is not tied to any particular virtual coordinate service. Other architectures can be adopted, such as IDMaps [FJJ+99] and GNP [NZ02] or PIC [CMAP04] and PCoord [LL04]. The first two rely on deployment of infrastructures nodes, while the other provide latency estimates gathered only between end-hosts, as VIVALDI does. We opted for VIVALDI because of its fully distributed nature and simple implementation.

In less strict latency context, the hop-count is usually preferred in contrast to the millisecond latency to provide distance estimation. Pastry [RD01, CDHR02], for example, uses a hop distance metric to optimize its response time.

Finally, SG-2 is a fully decentralized, self-organizing protocol for the construction of proximity-aware, superpeer-based overlay topologies. The protocol produces an overlay in which almost all nodes (99.5%) are in range with a *tol* la-

tency of 300 *ms*. The number of generated superpeers is small with respect to the network size (only 3-5%). The protocol shows also an acceptable robustness to churn. We believe that these results can be profitably adopted to implement several classes of applications, including strategy and role-playing games. Other classes of games, such as first-person shooter, are probably not suitable given their extremely strict latency requirements (inferior to 100 *ms*). These results are an improvement over existing decentralized games [BT01], that are based on strong replication [unr] or low-level facilities such as IP-multicast [GD98]).

We conclude noting that the results presented in this paper are only a first step toward the implementation of real superpeer applications; for example, in the case of P2P games, several other problems have to be solved, including security, state replication, state distribution, etc.

4.3.6 Superpeer topology under hub-attack

As we stated previously in Sections 4.1 and 4.2, the hub attack can affect not only the PSS itself, but also any higher level service relying on the PSS. This fact makes the attack a more serious threat for P2P systems.

We present an example showing the effect of the hub attack over a particular superpeer topology emerged by SG-2. The service [3] is highly dependent on the PSS and all its details are discussed in [JMB06].

From an algorithmic point of view, the key point is that the pheromone diffusion is made through a simple message-spreading protocol that uses the neighborhood managed by the PSS cache (see Section 4.3.3); a message has a maximum distance range to spread, therefore it is spread only to those neighbors that are at a *tol* distance range from the message source. Essentially, the SG-2 local broadcast service limits by itself the effective out degree of a node's PSS cache.

This fact implies that a full (100%) PSS cache pollution is not required to stop the spreading of a message, as a certain amount of the neighbors is excluded by

[3] The word "SG-2" can be used to refer to the SG-2 core protocol described in Section 4.3.3 or to the SG-2 service architecture depicted is Figure 4.8.

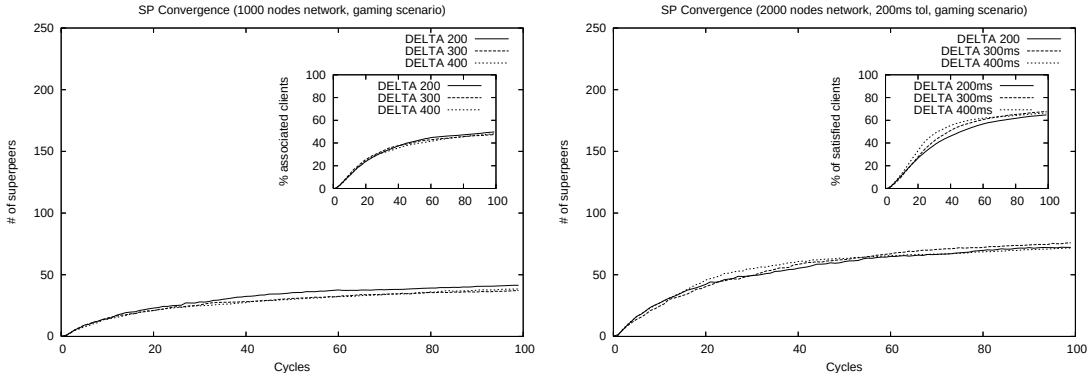
default due to the distance constraint. The broadcast service involuntary helps the attacker's job. Figure 4.12 shows two snapshots of the SP topology; the picture on the left shows the normal SG-2 behavior in standards conditions, while the picture on the right shows the effect of the hub attack over SG-2. Both snapshots have been taken when the topology is supposed to be completed (e.g., at cycle 30).

The network size is limited to 1,000 nodes and the distribution of the latencies follows a typical distributed game scenario (see Section 4.3.4 for details), $tol = 300\text{ms}$ allowed is 300ms. The big black dots depicts the emerged SP nodes; the thick lines show the SP connections, while the thin dotted lines show the connection relation between an ordinary nodes and its SP.

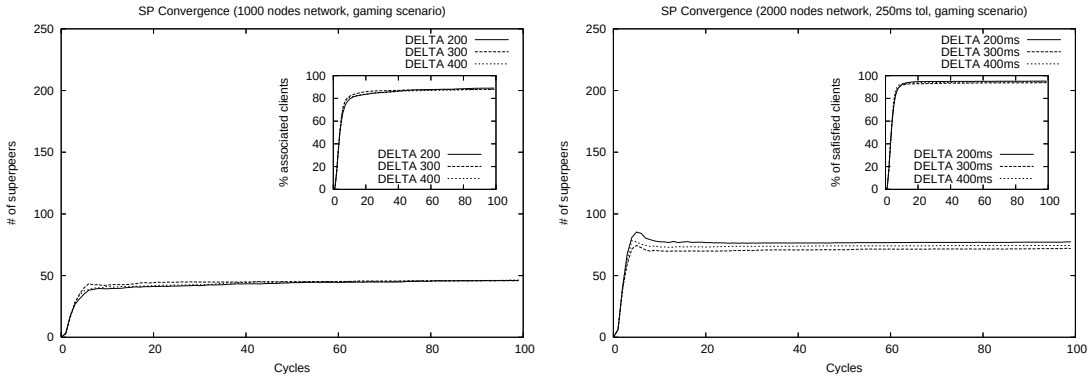
Figure 4.12(a) shows the normal SG-2 behavior in this context. The areas with a higher concentration of nodes are populated with more than one superpeer node; in this context, all ordinary peers are connected to an SP satisfying the desired QoS. Compared to the network size, the number of superpeer nodes is quite low (5%). Figure 4.12(b) depicts a dramatic situation. Without an effective pheromone communication, the system tends to be paralyzed. Each message takes a few cycles to cover the required area, but at the same time the attack pollutes the PSS caches. The result is that a message is either lost (e.g., finds no more neighbors to spread to) or it reaches an attacker node. This triggers the most capable attacker nodes to the SP status: the 14 superpeers available in Figure 4.12(b) are malicious nodes and they have acquired just a few clients in the early stages of the protocol. Many other nodes nodes cannot find any (good or malicious) SP because the SP messages can not be routed; in addition any capable ordinary node cannot switch to the SP status because it cannot perceive any message, as all traffic is sent to the attackers by default.

The effect of the hub attack may vary according to the actual SG-2 parameters; for example, with a different scenario, we had an enormous amount of SP (30% of the network population), but at the same time a low percentage of clients having the desired QoS (e.g., being connected to a *tol* range SP). In any case, the pres-

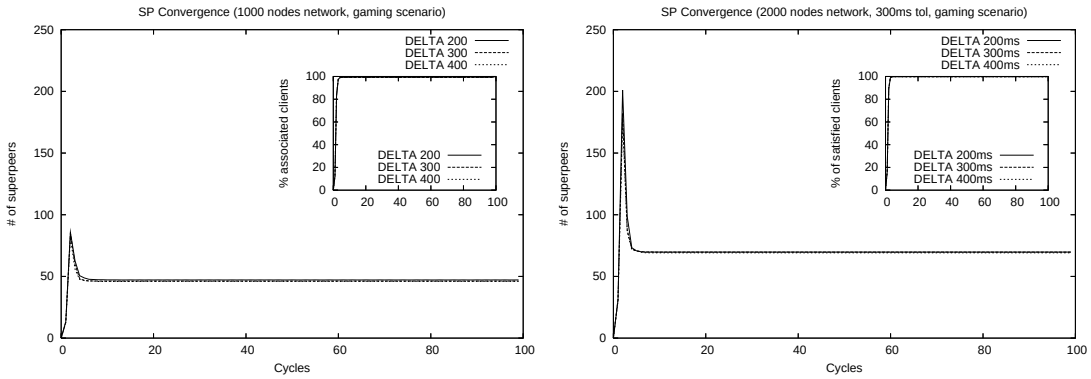
ence of the hub attack prevents the formation and the management of a healthy superpeer topology.



(a) $tol = 200\,ms$



(b) $tol = 250\,ms$



(c) $tol = 300\,ms$

Figure 4.10: Convergence time. Three *tol* values are considered: $200\,ms$ (a), $250\,ms$ (b), $300\,ms$ (c). The main figures show the number of active superpeer at each cycle, while the small sub-figures show the number of clients that are in *tol* range. Three different δ values are shown in each figure.

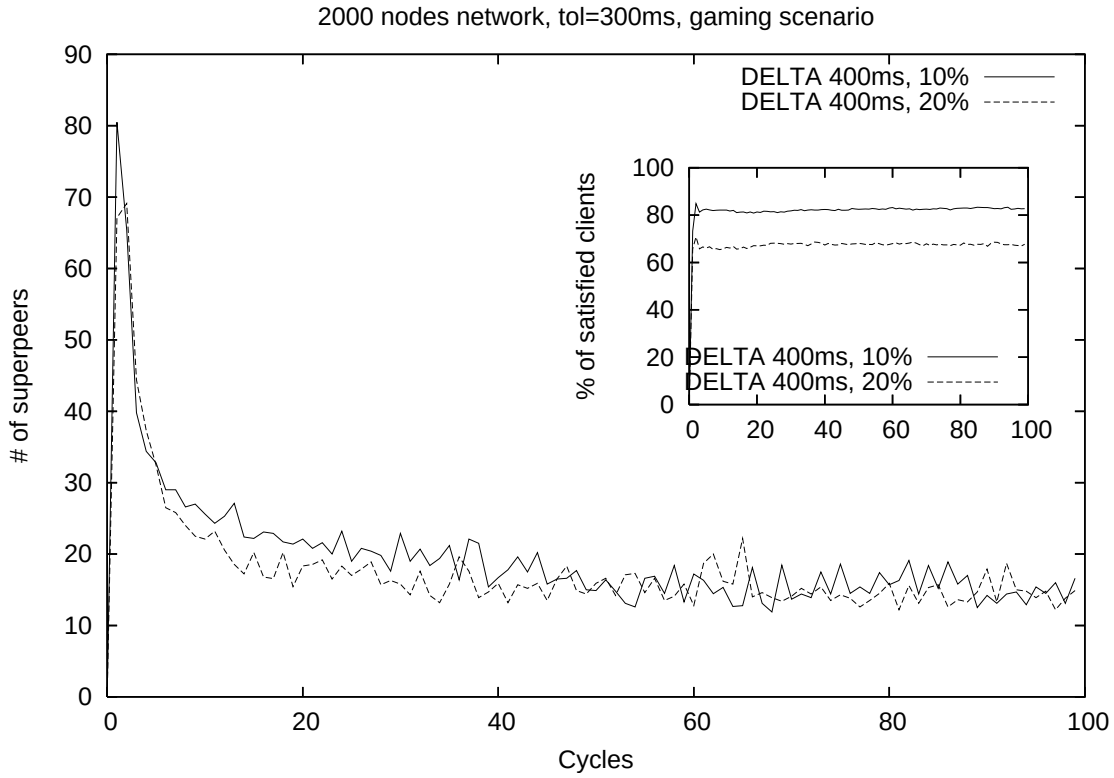
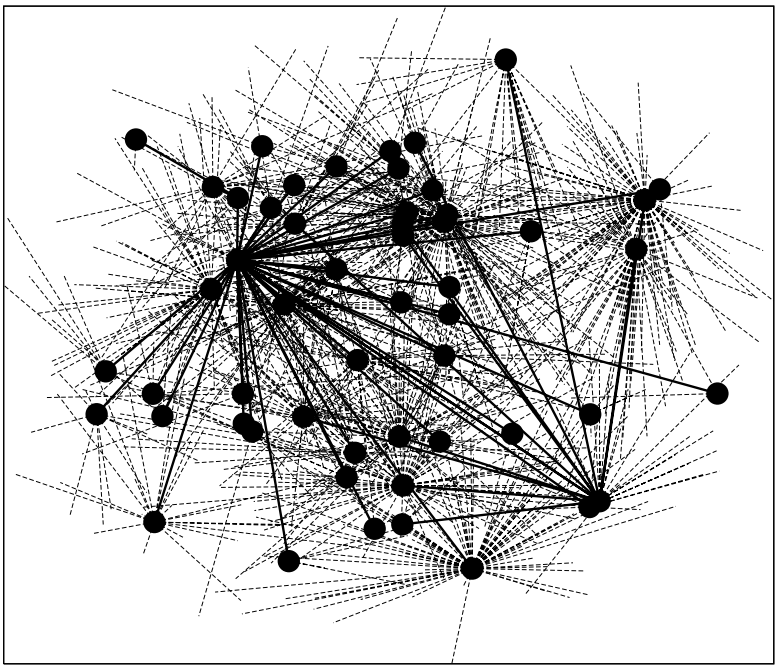
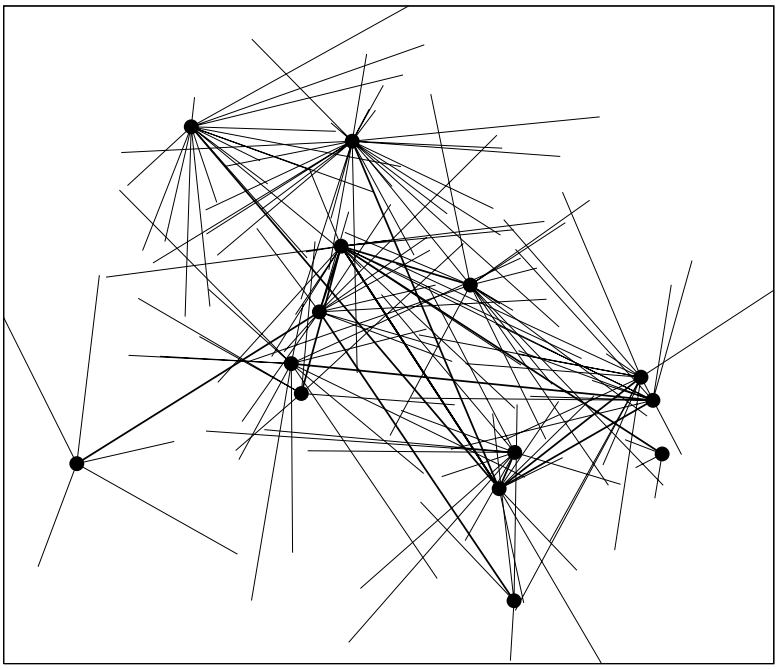


Figure 4.11: Experiments with churn. Network size is 1000; at each cycle, 10% or 20% of the nodes are substituted with new ones.



(a) Normal conditions SG-2 service



(b) SG-2 under hub-attack, 20 mal. nodes

Figure 4.12: A 1,000 nodes superpeer topology generated by the SG-2 service in normal conditions (a) and during a hub-attack (b); $tol = 300ms$, the PSS cache size c=20 (in (b) 20 attackers are involved). The big dots represent the SP nodes, the thick lines show the SP connections, while the thin lines show the connection

Chapter 5

Proposed approach: the Secure Peer Sampling Service

In this chapter we present our solution, the Secure Peer Sampling Service (SPSS), aimed to preserve the integrity of the PSS topology from a hub attack. Like the hub attack, our solution is independent of the actual PSS implementation adopted.

We present the SPSS as an incremental two-step solution: the first step involves the presence of one (or more) trusted node(s), while the second step focuses on a fully decentralised approach. For each solution step we present its own performance evaluation.

5.1 The problem

Generally speaking, our goal is to avoid the formation of hubs. The SPSS resulting topology must be as close as possible to the standard PSS topology (according to the actual implementation adopted). This goal must be achieved using only a node's local information [SP03], as sharing any extra information would represent another chance for the attackers to subvert the network (e.g., by diffusing fake suspicions).

As the only information available about the network is the local cache from a node's point of view, a good indication of the hub topology formation is the local

node *clustering coefficient* (CC).

However, the CC is computed as the proportion between the vertexes within the current node neighborhood (cache) divided by the number of links that could possibly exists between them. From the definition above, it is easy to argue that the node's local state represented by the cache is not sufficient to calculate the CC; in other words, the local cache and all the neighborhood caches are required. The "many-to-many" cache exchange would generate an excessive communication overhead and therefore we do not consider it as a viable approach.

As our goal is to limit the chance to generate hubs, we can let each node to evaluate if it is going towards a hub at each gossip exchange; the evaluation is based according to the local knowledge - i.e. the local cache - and to the neighbor state (cache) that each node receives at each gossip exchange. Notice that the essence of the PSS interaction scheme is not changed and the amount of information exchanged is the same.

Essentially, the idea is to let each node to *rate the quality* of the exchange in progress by comparing the caches of the involved parties. The *quality rate* is given by the number of items (IDs) lying in the intersection of the exchanged caches among node A and B: $r = |\{cache_A \cap cache_B\}|$.

This process of rating the quality of the exchange does not imply to accomplish the gossip round just because both parties have already exchanged their states, but instead it allows to accept or deny the exchange according to the perceived quality rate. The quality rate influences the *probability* to conclude the gossip exchange. Essentially, when two caches are similar (or identical) it is likely that the current neighbor is a malicious node and with high probability it should not be accepted.

Our aim is to verify the above property (i.e., the quality rate) in the least invasive manner. In other words, we aim to gently integrate this approach in the gossip scheme. We believe that designing our solution as a general scheme is a primary concern in order to protect any PSS implementation from the hub attack.

Due to this reason, we are interested in the design of a new simple primitive

do forever
wait(Δt)
neighbor = SELECTPEER()
SENDCACHE(neighbour)
$neigbor_{cache}$ = RECEIVECACHE()
checkIDs(myCache, $neighbor_{cache}$**)**
myCache.UPDATE($neighbor_{cache}$)

(a) Active Thread

do forever
n_state = RECEIVECACHE()
SENDCACHE(n_state.sender)
checkIDs(myCache, $neighbor_{cache}$**)**
myCache.UPDATE($neighbor_{cache}$)

(b) Passive Thread

Figure 5.1: The SPSS gossip scheme; essentially it is the PSS scheme extended by the checkIDs() primitive. The strict relation with the gossip scheme (see Figure 2.1) is evident; as the node's state in the PSS is the cache, the fundamental methods have a slightly different name, but they still hold the same semantic.

that incapsulates the details of the quality verification process. This primitive can be added in the standard gossip scheme depicted in Figure 2.1. We call this primitive function CHECKIDS(). Figure 5.1 shows how the CHECKIDS() function fits in the PSS scheme. The discussion of the actual action triggered by the function is an algorithmic detail and we are going to discuss it in the next sections.

5.2 SPSS requirements

The general system model described in Section 2.2 still holds; in addition, in our proposed SPSS solution, we require that in the *bootstrap* phase of the network, each node joining the overlay obtains a certificate for its public key from a central *Certification Authority* (CA). The CA is a centralised entity, but it is not involved in the protocol itself; it is just needed to join the overlay. After that, a joining node may obtain a starting neighbor list from a pre-assigned trusted node.

When a node leaves the network, voluntarily or due to a crash, its peer reference is quickly discarded by the underlying service, due to the properties of the PSS implementation. If a peer uses a special exit message to leave the network, then that peer must sign the message to prevent a DOS attack.

We cryptographically secure each ID structure using the following message format:

$$[\mathrm{ID}_A, \mathrm{ts}_{creation}, \mathrm{ts}_{expiration}, \mathrm{PK}_A, \sigma]$$

where ID_A is A's node identifier (see Section 2.2), the ts are timestamps, PK_A is A's public key and σ is the digital signature on the message. As noted earlier in Section 3.2, we assume the set of attackers to be relatively small and that they collude to forge each other's signed ID structures to create valid MN variant messages. For the FN variant, the attacker can forge signed ID structures for fake IDs on its own.

5.3 SPSS approach

Our approach to solve the hub attack is twofold: (1) we aim to detect malicious peers in the overlay with high accuracy and (2) we aim to reduce (nearly to zero) the effects of malicious peer actions on the topology structure. Our solution is designed to achieve these two goals with the least amount of effort.

The SPSS approach requires the presence of a trusted peer, the TRUSTED PROMPT node. This node has functionality similar to a proxy of the CA and provides peer credential management and access control. The important service provided for SPSS by the TRUSTED PROMPT node is credential revocation of peers that manifest malicious behavior.

To improve the resilience of our distributed system, we can also use multiple TRUSTED PROMPT nodes *without introducing any modification* to our basic algorithm. However, the trade-off between the robustness and the extra effort spent in deploying, configuring and maintaining many TRUSTED PROMPT nodes must be evaluated by the overlay designers. We stress that the TRUSTED PROMPT is not a potential bottleneck, as will become clear below.

The basic idea behind SPSS is the following: each peer executes the standard PSS algorithm, but, after each message exchange round, it performs an additional *checking step* on the received cache list. When the check fails, ordinary peers are *potentially* exchanging with a malicious peer; therefore, they contact the TRUSTED PROMPT reporting the ID-structure provided by the suspect peer. The TRUSTED PROMPT performs cryptographic checks to validate the raised suspicion and if correct then supplies the notifying peer with a new complete cache entry.

This checking step has two distinct issues to deal with (1) malicious node IDs and (2) fake IDs.

- **Malicious nodes check**: the peer compares its current cache C with the newly received one, C^* . This test has a stochastic nature and it is based on the fact that, in the MN variant, it is likely to receive from the malicious nodes a message holding a similar set (or subset) of malicious IDs. The probability to raise a suspicion is proportional to r/c, where r is the quality rate and c is the usual cache size.

  When the stochastic process raise a suspicion, then the peer terminates the exchange and contacts the TRUSTED PROMPT. This is a local check that does not require extra messages or communication.

- **Fake ID check**: the peer checks a certain percentage α (global parameter) of the IDs in the received cache by network communication (e.g., by sending ping messages). If a subset of fake IDs larger then α is found, the peer aborts the exchange.

In order to support its suspicion of peer P, the peer provides the TRUSTED PROMPT with both its own and the received caches as evidence. The TRUSTED PROMPT first verifies if P's ID signature is correct and that the ID matches the actual IP address it is transmitting from. If verification fails, then P is confirmed as a malicious or faulty peer and it is black-listed. Otherwise, it logs (or updates) an entry $\#(P)$ in a *frequency table* indicating how many times P has been reported as suspect.

Finally, the TRUSTED PROMPT builds a new cache for the querying peer. To build the new cache, the TRUSTED PROMPT picks nodes randomly from the network. A node Q is selected proportionally to $1 - \frac{\#(Q)}{\text{NetSize}}$ for inclusion in a cache. A lower value in the frequency table corresponds to a higher chance to be present in the cache (and to be a non-malicious node). The size of the network required by the formula can be computed in a distributed fashion using, for example, an aggregation protocol [JMB05]. The process continues until c suitable peers are found and then the cache is sent back to the peer. Clearly, the TRUSTED PROMPT is not fail proof as the attacker identification is based on reported suspicions, which cannot be 100% accurate. However, as shown in Section 5.4, the SPSS achieves very good results in preventing the hub attack.

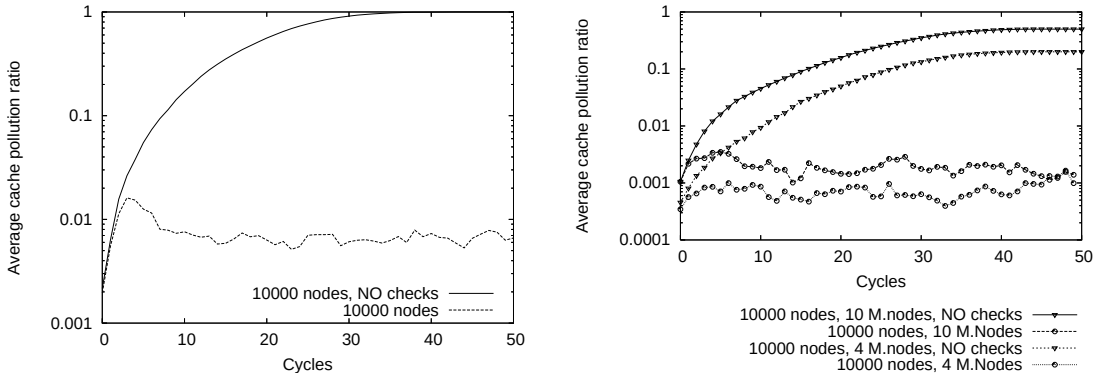
5.4 SPSS evaluation

To evaluate the SPSS, we adopted the same approach and the same set-up as discussed in Section 3.3; in addition, we set α=5%. This percentage corresponds to the amount of IDs actually checked at each cycle by each (non malicious) peer.

We restrict our discussion to the larger $10,000$ nodes overlay.

The SPSS performance is summarised in Figure 5.2. Figure 5.2(a) shows the average level of pollution lying in a peer's cache during a 20 malicious nodes (MN variant) attack. In this set-up, there is almost no difference between the distinct percentages of checking. This behavior could be explained due to the full ID pollution in this particular set-up: it is easy to detect a common subset of replayed IDs, regardless the actual amount of checks. The difference between the distinct checking efforts is still very low, this proves that our defence can prevent the attack even with a low checking effort.

In less than 10 cycles the pollution level stabilises in a very low oscillation range. The oscillations are due to the stochastic nature of the TRUSTED PROMPT that may inject malicious nodes when it provides the new cache to a querying peer. However, this has no long term negative consequences. The initial pollution



(a) MN attack, 20 m. nodes

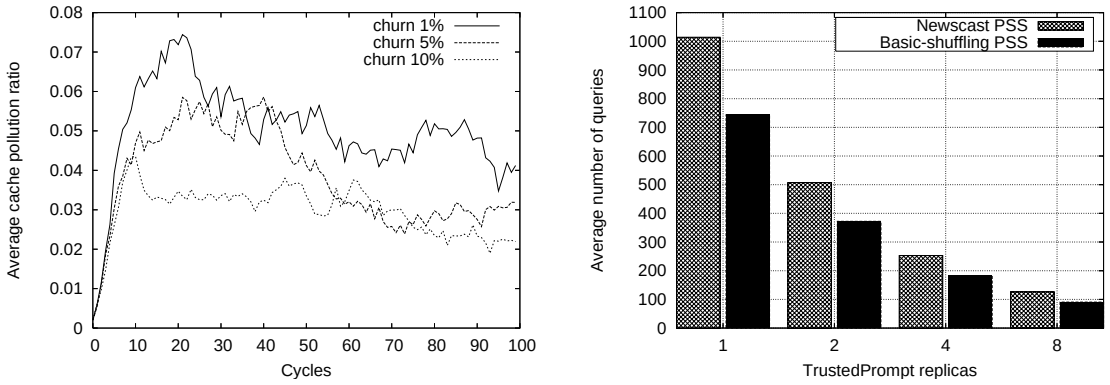
(b) FN attack, 10 and 4 m.nodes, α=5%

Figure 5.2: Comparison among the PSS and the SPSS pollution ratio under a Hub-Attack. The overlay size is $10,000$ nodes. Distinct checking setups are shown.

peak is absorbed as soon as the TRUSTED PROMPT has received sufficient feedback by the well behaving nodes.

Figure 5.2(b) shows how the SPSS deals with a FN variant attack. Two distinct set-ups are considered: using 10 and 4 malicious nodes, while the rest of the IDs in the cache are fake IDs. The check level adopted is α=5%, in order to detect fake IDs. The two distinct checks almost doubles the chance to immediately identify a malicious attack. Again, in less than 10 cycles, the pollution level drops to a near-zero level. With such a low amount of pollution in the cache, the SPSS preserves the original PSS topology properties and there is no danger of partitioning of the overlay due to the malicious peers leaving.

Due to the dynamic nature of any P2P system, we decided to consider churn and how the SPSS deals in this more realistic setting. Our churning results are shown in Figure 5.3. Periodically, a fraction of the peer population leaves and is substituted with new peers. The attacker nodes however, are not affected by the churning process and they are allowed to pollute for the whole duration of the experiment. We consider three churn set sizes: 1%, 5% and 10% of the network size. These three values are quite high churning rates (see [MCR03]), but they are



(a) Average cache pollution ratio (b) Query message traffic, 1% churn

Figure 5.3: Dynamic scenario results: distinct level of churning rate (1%,5% and 10% of the network population) are shown during a MN variant attack (20 m. nodes). (a) Depicts the average cache pollution, while (b) shows the difference among the average number of queries sustained by 1 or more (2, 4 and 8) TRUSTED PROMPT using two distinct PSS implementations.

fine in order to stress our solution. The attackers involved are 20, playing the MN variant of the hub attack.

Figure 5.3(a) shows the average cache pollution during the attack. With a churn rate of 1%, the cache pollution has a peak of 7.5% at about cycle 20 and then it tends to decrease. Strong oscillations are clearly visible in the picture due to the dynamism itself; however, the pollution level is always in a very low range.

It is interesting to note that for higher churn rates, the pollution level is even lower. This fact may seem counter intuitive, but the churning process triggers the FN defense policy and aborts any exchange with any neighbor reporting an unusual number of non existent node IDs in the cache. This side effect helps keeping the cache cleaner.

We tried to let the attackers leave the network as in Section 3.1, but we never had any partitioning report with such a low pollution levels.

Figure 5.3(b) shows the average number of queries per cycle. Multiple TRUSTED PROMPTS and two distinct PSS implementations (e.g., NEWSCAST versus basic-

shuffling, see Section 2.2.1) are considered. We restricted the TRUSTED PROMPT number to a maximum of 8 replicas to keep the management and configuration costs to a tolerable level.

The number of queries is about 10% of the network size in the NEWSCAST case, while the basic-shuffling has about a 30% advantage. In this test, the implementation of the PSS makes the difference. The worse results achieved by NEWSCAST are due to the higher CC that tends to produce a much higher number of false positive suspicions. The CC distribution is far from being uniform, thus many well-behaving nodes suspect non-malicious neighbors.

Clearly, the adoption of multiple TRUSTED PROMPTS allows a uniform distribution of the queries among the trusted nodes. This load balancing can be easily obtained by selecting randomly a TRUSTED PROMPT, as the current available TRUSTED PROMPTS are advertised by any of their reply messages.

When considering the scalability of our approach, one may come to think that the TRUSTED PROMPT is going to be a source of problems. However, we have found that on average, the TRUSTED PROMPT receives a manageable number of queries per cycle. Because of the periodicity independence of our approach, we conclude that having a single TRUSTED PROMPT is not a concern for a large network, as we can choose the cycle time scale according to the actual network requirements. Of course, the service should be highly available, which can be established through traditional (lazy) replication techniques.

Finally, in Figure 5.4 we consider an extreme case in which a set of colluding attackers larger than the cache size ($c = 20$) is involved. This scenario is very unlikely because the attackers must be colluding by definition of the hub attack. Notice that we have excluded from our model the possibility of a Sybil [Dou02] attack and due to other simple ID scheme we have supposed that every PSS instance is running on a distinct host (see Section 2.2 and 3.2).

We considered 50, 100 and 200 malicious nodes corresponding respectively to 2.5, 5 and 10 times the actual cache size of the underlying PSS. With 50 attackers the SPSS still provides a good defence as the cache pollution level is about 15%, a

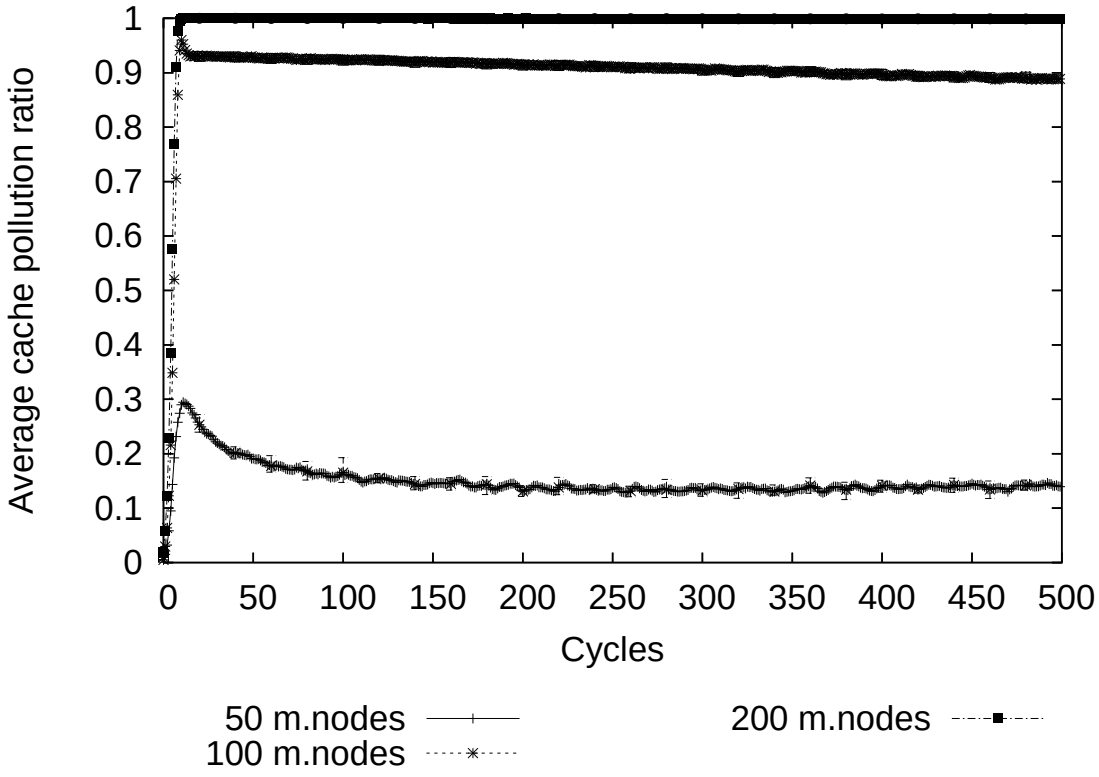


Figure 5.4: SPSS dealing with more than `cachesize` (c = 20) malicious nodes. The results regarding 50, 100 and 200 attackers are shown.

quite safe value and far from the risk of partitioning if the attackers leave.

Switching to 100 attackers, the situation changes dramatically. The pollution level is exponentially higher; the SPSS takes 500 rounds to achieve a pollution level of about 88%. Of course, the system is exposed to the risk of (massive) partitioning at any time. However, the SPSS curve is decreasing showing that the SPSS is reacting to the threat; although, the reaction is too slow to have any practical impact.

The upper line corresponds to the 200 attackers set. In this case, the SPSS is quickly defeated and seems incapable of any reaction.

In this unlikely scenario however, the SPSS shows to able to manage safely a set of colluding attackers larger than 2.5 times the size of the its current cache.

5.5 Decentralised SPSS

We believe that the main shortcoming of our centralised SPSS solution is that the deployment of trusted nodes over the Internet, in order to sustain our secure gossip system, is a viable approach only for organisations or companies with trusted administrative control.

In other words, the TRUSTED PROMPT approach requires some extra trusted infrastructure that complicates the system deployment and maintenance. Using the simplest set-up, i.e., using a single TRUSTED PROMPT, we minimise the deployment issue, but produce a single point of failure. If the single TRUSTED PROMPT is hacked or crashes then the whole network is vulnerable to attack.

A fully decentralised solution would be preferred as it would lower the burden to design and to deploy secure gossip systems and would not require trust external to the system (other than the CA which is external to the protocol), but is this possible to achieve? And what kind of trade-offs do we need to consider? Our aim is to refactor our previous approach in order to obtain a fully decentralised solution, in which each node has the chance to detect the malicious nodes using its own resources.

5.5.1 Multiple overlays

As we have seen the main obstacle to prevent and detect the hub attack is represented by its *high spreading speed*. Such a high speed leaves no time to the peers to make any successful guess about the identity of the attackers. This is why in our previous SPSS solution we rely on the TRUSTED PROMPT assistance.

The basic idea for the fully distributed SPSS is based on using multiple, concurrent instances of the PSS. Therefore, each node participates in multiple overlay graphs, and the neighborhood at every instance will be distinct with very high probability because the overlays have independently random-like topologies. Essentially, the multiple caches over the same node population, which every node

adopts, give each peer a snapshot of what is going on in distinct (random) neighborhood of the overlay. We call *extra caches* the set of caches belonging to each peer; every cache in the set is a random snapshot of a distinct PSS overlay

We assume the same attack model as before: a set of k colluding attackers, but running multiple PSS instances as well, will pollute all the available instances. This hypothesis makes our scenario more challenging.

Each node can monitor the pollution ratio by looking at its extra caches. Since the network population of all the PSS instances is the same, all the extra caches will become polluted by the same k malicious node IDs, if no checking action is performed. However, an attacker can pollute at most only a single node's cache at a time per overlay. In addition, due to the random nature of the available overlays, it is very unlikely that an attacker could defeat all caches of the same victim peer in a short time window. Essentially, the multiple caches are useful in order to perceive how malicious node are spreading the infection from distinct directions over distinct overlays. Due to the spreading infection, we expect that common node ID patterns will emerge in all (or in the majority) of the caches.

5.5.2 Quality rating

Each peer can build a set of statistics in order to guess or detect who are the malicious nodes from the emerging patterns. This knowledge base is stored as private, local black- and white-lists that it is never exchanged among neighbors (see [NCW05]). This obviates the second-order issue of malicious nodes spreading incorrect reputation information.

During a gossip exchange, both parties *rate the quality of the exchange*. The quality rate is given by the number of items lying in the intersection of the exchanged caches among node A and B: $r = |\{cache_A \cap cache_B\}|$, as described in Section 5.1. This quality rate influences the probability to conclude the gossip exchange with this current neighbor. Essentially, when two caches are similar (or identical) it is likely that the current neighbor is a malicious node and with high probability it should not be accepted. The probability to abort the exchange is proportional to

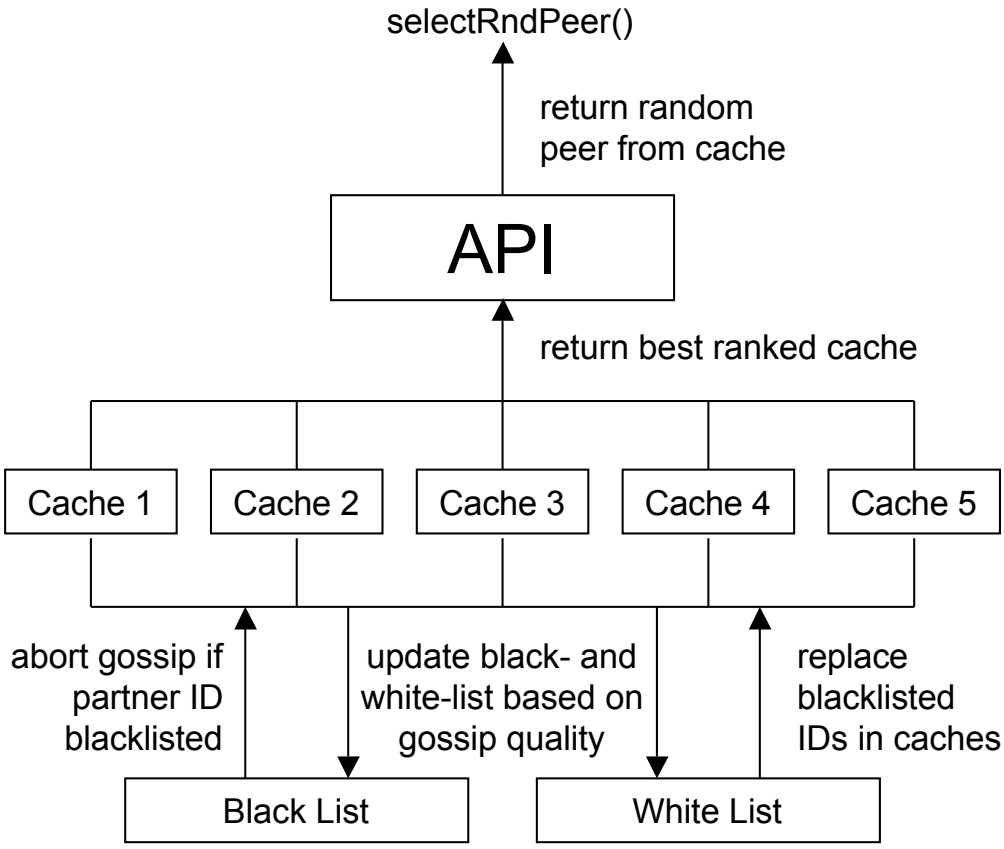


Figure 5.5: Schematic of the decentralised SPSS; it maintains multiple caches to support multiple random overlays. Black and white-lists screen incoming gossip requests and refresh malicious cache entries. The highest quality cache is mapped to the API to support standard peer sampling functions.

the fraction of the common IDs found among the two caches: r/c, where c is the usual cache size.

The rank results are collected in the node's *knowledge base*. The information collected in this structure is refreshed according to an ageing policy to avoid that any wrong guess would have unbounded consequences over time.

Any attempt to exchange with a neighbor (black-) listed as a high frequency and low quality rated node is declined. In addition, when a node suspects one of its caches is polluted, it tries to refresh the cache randomness by substituting the

currently blacklisted node IDs with high quality rated node IDs collected during the previous exchanges (if any).

During the protocols execution, one or more cache can be defeated by the attackers. However, this is not critical, as the cache will be restored as soon as the node has collected a suitable knowledge base. It is very unlikely that all node's caches become polluted in a short amount of time; in this unlucky condition and if the knowledge base is not ready or not correct, the only chance for a node is to be contacted by a well behaving node in order to partially restore at least one of its caches. This is the exact situation we have in the previous SPSS version. Figure 5.5 shows a schematic of the main components maintained by the protocol within each node. A 5 caches scenario is depicted.

5.5.3 The algorithm

Our decentralised approach is focused on the knowledge base each node has to build. Essentially, the knowledge base is represented by two list structures: BLACKLIST and WHITELIST; the former holds high frequency and low quality rated node IDs, while the latter holds high quality rated node IDs. We do not set any explicit size limit for these structures and, as a consequence, their size may grow to the actual network size. However, due to presence of an ageing policy, their actual size is much less than the theoretical maximum. The SPSS algorithm pseudo-code executed by a node A is the following:

1. Select a random neighbor $B \notin$ BLACKLIST, if any
2. Compute the quality rate r with B; proportionally to r/c decline and blacklist B, otherwise accept the gossip exchange, and:
   (a) whitelist B
   (b) perform the standard PSS exchange with B

These steps are performed in each cycle for every available cache. Two additional actions are performed concurrently by two threads at the end of each cycle.

The first action is to purge the BLACKLIST and WHITELIST according to an ageing policy; the second action instead, is to repopulate the caches suspected of being polluted (if any): each node ID in the cache listed in the BLACKLIST is substituted by a random node ID picked from the WHITELIST.

Another issue is to clarify how node IDs can be inserted and swapped from the BLACKLIST to the WHITELIST and vice-versa. When a node ID has to be inserted in the BLACKLIST for the first time, a standard TTL value (2 cycles) is bound to the stored ID; if the ID is already present instead, its TTL value is reinforced (i.e., doubling the current TTL value). This reinforcement process is needed in order to keep in the BLACKLIST the most frequent node IDs (with a poor rate).

About swapping the IDs among the two structures, suppose node B's ID is already in node A's WHITELIST, but now node A had to insert B's ID into its BLACKLIST. B's ID is removed from A's WHITELIST and it is inserted in the BLACKLIST. In other words, the BLACKLIST has *more authority* than the WHITELIST.

Likewise, if node A has to whitelist node B's ID, but it is already in A's BLACKLIST, the swap between the two list is not allowed until B's ID is purged from the BLACKLIST. This rule is designed to avoid that a node's PSS instance exchanging with a malicious node for the first time, would not overwrite a possible correct suspicion made by a more experienced instance.

5.5.4 Why it works

It is important to note that having multiple caches belonging to distinct PSS instances is very different from having a single PSS with a possibly huge cache. Multiple caches add extra randomness to the node's state and avoid to be defeated in just one exchange; in addition, in extreme conditions (i.e., when the set of attackers is larger than the cache size, see section 5.6.4) they still give the chance to identify the attackers.

The value added by multiple PSS overlays is that the infection proceeds from distinct multiple paths. These dynamics gives each peer more time to detect the most frequent node IDs that appear in their caches.

A higher-level protocol working on top of this fully distributed SPSS can see just a single cache, in order to maintain a seamlessly integration with the standard PSS API. A smart implementation of the fully distributed SPSS can dynamically export the *current best cache* according to concentration of suspected malicious nodes currently listed in the knowledge base (see Figure 5.5).

5.5.5 Evolutionary link

The multiple caching concept originates from previous socially inspired evolutionary models of "group selection" [HE05, HA06]. In these models anti-social behavior between nodes was avoided by allowing nodes to form and move between different clusters or groups in the population based on utility value comparisons. Essentially, nodes evaluated the quality of their neighbors by measuring the effectiveness of interaction with them over time – involving some application level task – and represented this as a utility value. By comparing utilities with other randomly selected nodes and copying the neighborhoods (caches) of those with higher utility, nodes could avoid interaction with anti-social free-riding nodes. In this approach nodes maintained a single overlay and made intra-overlay movements to find better (higher utility) neighborhoods.

For the distributed SPSS we implemented a similar scheme by allowing each node to store multiple caches and only selecting the best cache based on a measure of utility expressed as cache quality. From the point of view of what is passed to the API, nodes are constantly shifting between different views of the network since each cache represents a different set of neighbors. Furthermore, when the quality for a particular cache becomes low due to possible identification of malicious information, it is wiped and reinitialised from the white-list, hence low quality caches are dropped.

Hence in SPSS nodes do not move between distinct groups or clusters in a single overlay but maintain and effectively move between distinct overlays (inter-overlay movement) comprising the same population of nodes but in different topological configurations. Hence what is being selected here by each node is the

overlay which produces the best cache quality at each given point in time. Since all nodes actually stay in all overlays at all times (by maintaining a fixed number of multiple caches) this approach is less a form of evolution and more a form of redundancy with dynamic selection.

5.6 Fully decentralised SPSS evaluation

In order to evaluate our new approach, we investigate the following main issues: (a) how much time is required to achieve a tolerable[1] amount of pollution in the node's caches, (b) how many extra caches are required to prevent the attack, (c) how the performance scales according to the number of the extra caches adopted, (d) the performance of our approach in terms of communication cost; finally, we are also interested in (e) the performance when the hub attack is played by a number of malicious nodes larger than k or, in other words, when $k > c$.

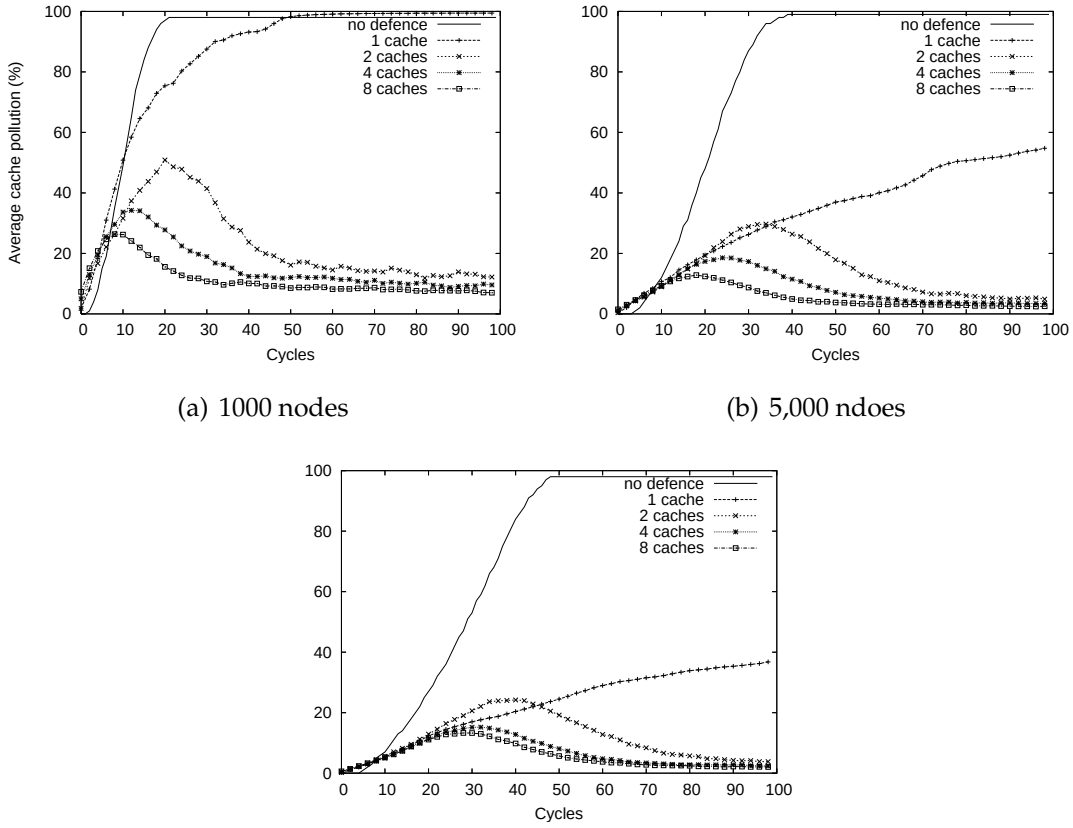
If not stated explicitly, in the following evaluation, we consider the usual scenario for a hub attack: when the number of malicious nodes k is equal to the (*single*) cache size ($k = c = 20$).

5.6.1 Static environment

Figure 5.6 shows the average pollution level in the node's caches for each considered network size (1,000, 5,000 and 10,000 nodes respectively). In this scenario, we consider a static network in which both malicious and well-behaving nodes are not subject to crashes; also network links are considered perfect and without message loss.

Each plot shows a SPSS set-up using a distinct number of concurrent caches per node; we have shown the results for 1, 2, 4 and 8 caches set-ups. As a reference, we also plotted what happens when no attack countermeasures are taken

[1] We consider the pollution in a tolerable range if the graph does not split into clusters when the malicious nodes leave.



(a) 1000 nodes

(b) 5,000 ndoes

(c) 10,000 nodes

Figure 5.6: Fully decentralised SPSS algorithm. The average pollution level in the caches is shown over time; multiple distinct caches per node are compared (e.g., 1, 2, 4 and 8 caches) for each network size (e.g., 1,000, 5000 and 10,000 nodes). 20 malicious nodes are involved in the attack.

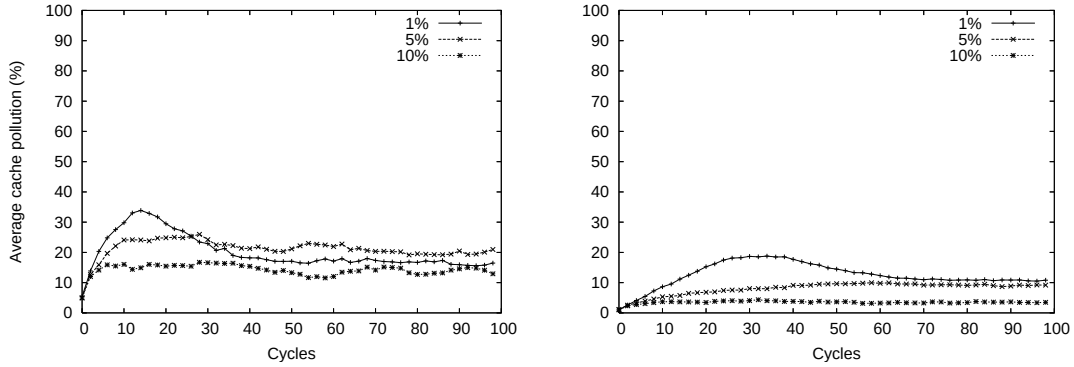
(see the solid topmost line in each chart). Of course, when nothing prevents the malicious node's activities, the cache pollution level quickly reaches 100%. When the distributed SPSS is run with just one cache, the pollution level monotonically increases; the smaller network becomes defeated in about 50 cycles because a degree of 20 is quite high compared to its size. However, this set-up cannot be considered a full countermeasure since we still use only a single cache. Essentially, the blacklist mechanism is not sufficient per se in order to recover the network.

By using two or more concurrent caches per node, the situation changes dramatically. Two caches are already sufficient to recover the network, regardless the network size. In all cases, the pollution level is never dangerous. Here, by dangerous, we refer to a level over which the network would suffer from partitioning if the malicious nodes leave the network; in general, this happens when the cache pollution is $\geq 75\%$ (see [JGGvS06]). By increasing the number of caches, we further lower the pollution, however, especially in the bigger network, the advantage in the adoption of 8 instead of 4 concurrent caches is almost negligible. In addition, a pollution level below 20% does not pose any threat of partitioning the network. For this reason, according to our experiments, we consider the 4 extra caches set-up a good trade-off between complexity and effectiveness.

5.6.2 Dynamic environment (churn)

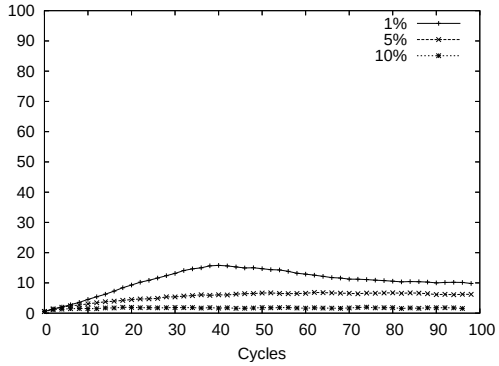
Figure 5.7 shows the performance of the SPSS under churn. We measured the average pollution level in the node's caches for each distinct network size (1,000, 5,000 and 10,000 nodes). We allowed three distinct churn set sizes: 1%, 5% and 10%, respectively; this amount of nodes leaves the network at every cycle and it is substituted by an equal number of new participants. The malicious nodes, however, stay in place and attempt to pollute caches for the whole duration of the experiment. Note that these values are actually quite high [MCR03], but will allow to demonstrate the feasibility of our solution. Each node has a 4 extra caches set-up.

It is surprising to see that the dynamism of the network helps the SPSS to keep the pollution level low. In fact, the level is lower than in the static scenario, for all the considered network sizes. In addition, a higher level of dynamism corresponds to a lower level of pollution. The reason lies in the fact that there is a higher proportion of fresh nodes injected in the system with a very low probability of having a malicious ID in cache; the well-behaving nodes that work in system for a longer time, will hardly diffuse the malicious IDs as they have already blacklisted them with high probability. Therefore, it becomes harder and



(a) 1000 nodes

(b) 5,000 ndoes



(c) 10,000 nodes

Figure 5.7: Fully decentralised SPSS under churn conditions. The average pollution level in the caches is shown over time according to three churn set sizes (1%, 5% and 10% of the network population) and for each network size (e.g., 1,000, 5000 and 10,000 nodes). 4 concurrent caches are adopted by each participant. 20 malicious nodes are involved in the attack.

harder for the malicious nodes to diffuse their bogus caches.

Essentially, on average no well-behaving node will play in the system enough time to detect successfully all malicious nodes, but this total knowledge is not required at all. The knowledge of who are the malicious nodes is distributed over the system as a whole; in other words, it is sufficient that every attacker is known by *some* healthy node.

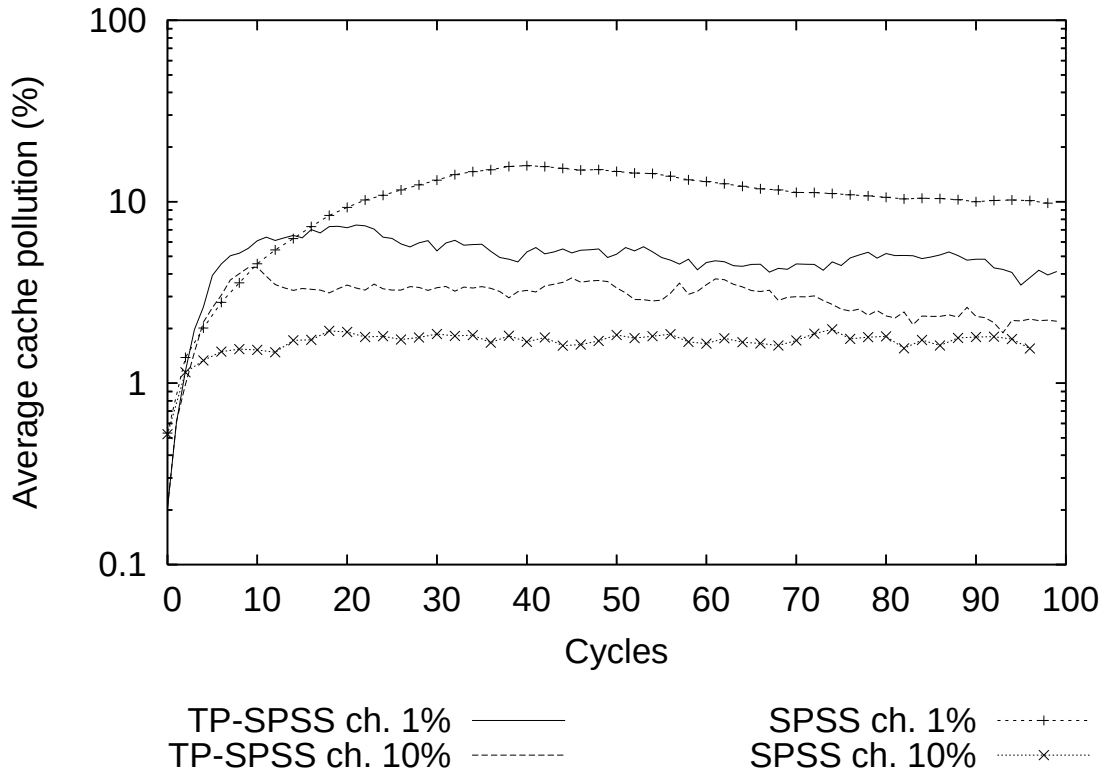


Figure 5.8: Comparison among our previous TRUSTED PROMPT based SPSS and the current decentralised one (4 extra caches). Two distinct churn scenarios are shown for each one. Network size is 10,000.

In Figure 5.8, we show a comparison between our previous TRUSTED PROMPT based SPSS and the new decentralised one in the dynamic environment. The set-up of the decentralised SPSS consist of 4 extra caches. We adopted two churn set sizes: 1% and 10% of the network population. The lines marked with the symbols $+$, $\times$ and $*$ depicts the decentralised SPSS, while the standard lines depict the TRUSTED PROMPT version. The cache pollution levels achieved are quite similar. The new version has a small disadvantage when the churn rate is low (e.g., 1%). However, in the worst case the pollution reaches a stable 10% and it is far from a critical range. In other words, we do not run the risk to have the network partitioned if the malicious nodes leave. In general, the decentralised SPSS achieves a

more stable pollution level than the centralised version.

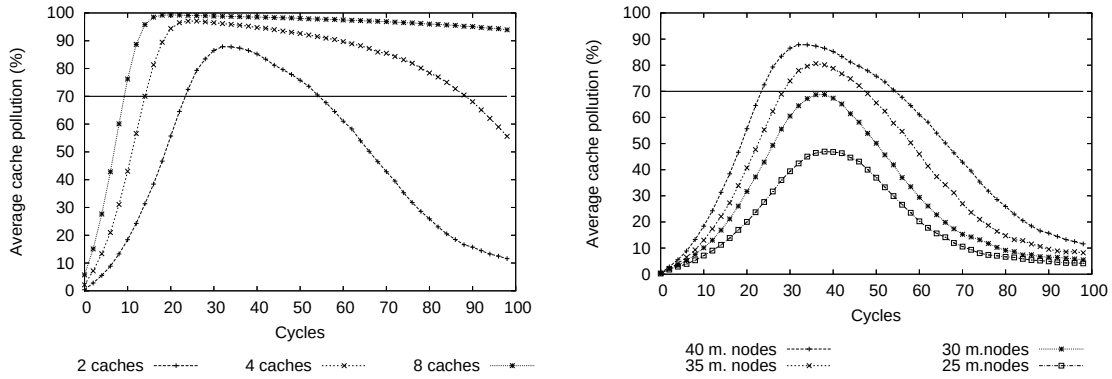
5.6.3 Message overhead

The main advantage of the decentralised version over the TRUSTED PROMPT based one, is the minimal message traffic cost. Essentially, the extra cost to sustain is due to the collection of extra PSS instances involved in the new approach. In fact, we avoid the traffic generated by the queries sent to the TRUSTED PROMPT (e.g., about 1,000 of queries per cycle in a 10,000 nodes network).

Using NEWSCAST as implementation, the cost is n times the cost of each PSS instance; as the average number of exchanges per node can be modelled by the random variable $1+\phi$ (see [JKvS03]), where ϕ has a Poisson distribution with parameter 1; the overall node cost per cycle is:

$$\left(\sum_{i=1}^{n} PSS_i\right) = \left(\sum_{i=1}^{n} 1 + \phi_i\right) = 2 \cdot n$$

5.6.4 Extreme conditions



(a) $k = c \cdot n$ m. nodes (b) 2 caches, $k > c$ m. nodes

Figure 5.9: Comparison of the graph topology properties in distinct scenarios. The clustering coefficient is shown in the left picture, while the avg. path length is shown in the right one. Network size is 10,000.

We are interested in verifying the tolerance limit of our approach in terms of number of colluding attackers and to make a comparison with the centralised approach. In the previous section we have seen that the decentralised SPSS can recover the overlay when $k = c$ malicious nodes and $n \geq 2$ caches are involved. This performance is given by the redundancy of the node's state. The experiments shown in Figure 5.9 depict the performance of the (decentralised) SPSS when $k > c$ malicious nodes are involved in a 10,000 nodes network.

Figure 5.9(a) shows what happens in the extreme case in which $k = c \cdot n$ attackers are injected in the network. Essentially, we consider to pollute all the extended state of the node. As before, we consider $c = 20$ the size of a single cache and n the number of caches adopted; we considered 2, 4 and 8 caches, corresponding to 40, 80 and 160 malicious nodes respectively. The effect of the presence of $c \cdot n$ attackers grows much faster than the benefit given by the multiple caches.

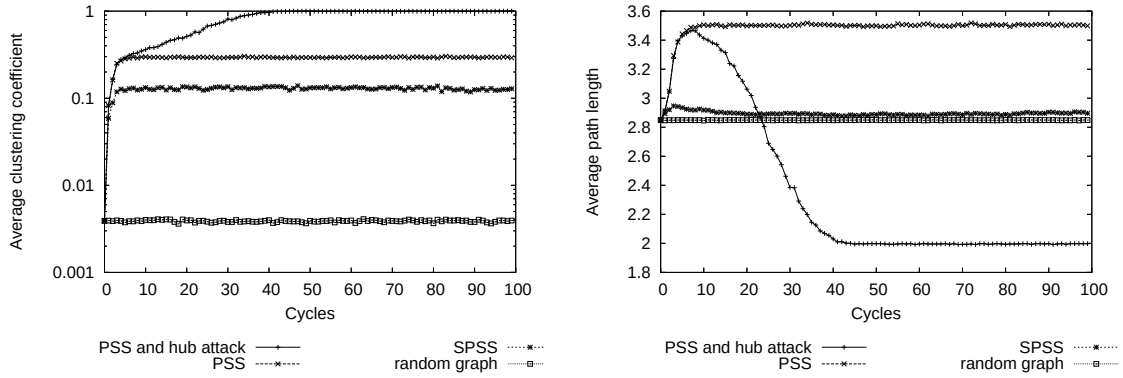
We may come to think that having $k = c \cdot n$ attackers is the same as the single cache case ($k = c \cdot 1$, see Figure 5.6), but, instead, the multiple cache presence allows the decentralised SPSS to slowly recover the network. However, the recovering process can be very slow and, more important, the pollution level grows over the dangerous level, depicted by the thick horizontal line, with any number of caches; if the attackers leave, the network will be severely partitioned. Basically, when the whole state can be polluted by a sufficiently large set of malicious nodes, the performance is bad (i.e., the network can be partitioned), but the decentralised SPSS is not paralysed as it is still capable of detecting the attackers.

In Figure 5.9(b), we check how many malicious nodes can be tolerated by a SPSS using 2 caches. We are interested to know the maximum number of attacker we can successfully tolerate, according to the actual redundancy (caches), without exceeding the threshold represented by the horizontal line. With this set-up, the system can tolerate $k = c \cdot 1.5 = 30$ malicious nodes. However, as can be argued by the situation depicted in Figure 5.9(a), an increment of the state redundancy (the number of the caches) has a less than linear increment in the number

of tolerable attackers. When considering $k = c \cdot n$ attackers, the centralised version is successful as the TRUSTED PROMPT handles the node states; therefore, the node states can always benefit from an extra state help.

5.7 PSS properties maintenance

The aim of this section is to show if and how the SPSS preserves the standard PSS topology properties. In other words, we are interested in checking if the topology generated and maintained by the SPSS can still be considered similar to a random graph from node's point of view.



(a) Clustering coefficient (b) Avg. path length

Figure 5.10: Comparison of the graph topology properties in distinct scenarios. The clustering coefficient is shown in the left picture, while the avg. path length is shown in the right one. Network size is 10,000.

Figure 5.10 compares the following scenarios: (a) the PSS during a hub attack, (b) the PSS in normal conditions, (c) the SPSS (during a hub attack) and (d) an ideal random graph generated by an oracle [2]. All the graph topologies involved

[2]The plot of the ideal random graph depicts oscillations while it should be perfectly flat, as it is produced by a static graph. The oscillations are produced by the measurement process; this process considers a random subset of the graph at every cycle to speed-up the computation, therefore producing a bit of variance in the values.

have a degree of 20 (e.g., the (S)PSS cache size is 20).

Figure 5.10 (a) shows the avg. clustering coefficient among the four scenarios. Scenario (a) quickly reaches a CC value close to 1 as a consequence of the hub topology formation, while in normal conditions (scenario (b)) the CC value is around 0.30 (e.g., the typical value of the NEWSCAST implementation). The value achieved by the (c) SPSS scenario instead is lower and closer to a real random graph set-up (d). The enhanced randomness is due to the TRUSTED PROMPT's provided caches.

Figure 5.10 (b) shows the avg. path length among the four scenarios. Scenario (a) initially depicts a behavior similar to the standard PSS until the attacker's action becomes dominant; then it drops to a value of 2 in 30-35 rounds. This value is expected as it is characteristic of the hub topology. Contrary to our selected PSS reference implementation, the SPSS achieves an avg. path length value very close to that of an ideal random graph (e.g., scenario (c) versus (d)).

These features show that the SPSS resulting topology can still be considered a random graph-like topology as it achieves better (lower) clustering characteristics than our reference PSS implementation (NEWSCAST).

5.8 SPSS discussion

In this chapter we have provided a general solution scheme suitable for any PSS implementation. The SPSS is based on a stochastic approach and achieves the goal of maintaining the underlying topology despite the presence of a set of malicious nodes playing the hub attack algorithm.

The SPSS scheme is designed to maintain the PSS properties when the number of (colluding) malicious nodes k is less or equal of the PSS cache size c, but it proves also to be effective when the set of attackers is 2,5 times larger than the cache size.

It is important to understand that the SPSS task is to maintain the underlying topology only in presence of attackers. In other words, it cannot prevent any

other malicious behavior running, for example, as a higher level protocol; this protocol may eventually rely on the PSS. However, the fact of having a "secured" PSS can slow down the malicious intent, but cannot prevent it for sure.

Chapter 6

Securing higher-level services

In this chapter, we focus on securing gossip services relying on the PSS facilities. Our securing effort targets a specific problem: the corruption of the gossiped information. Our aim is to limit the spreading of the corrupted information with the minimum possible effort.

We identify a class of generic gossip algorithms to which we apply a general, but effective technique, based on probabilistic checking, in order to limit the spread of corrupted information. The adopted technique has been already used with success in a different (wireless) context [GJGvS07].

6.1 The second problem introduction

From the conclusions of the previous chapter, we know that the SPSS preserves the random graph topology from the malicious action of a set of colluding attackers playing a generic attack model (e.g., the hub attack, 3). However, the SPSS can not prevent any other malicious action carried out by the attackers at an higher-level service, relying (or not) on the PSS facilities. For example, the attackers can *forge* or *corrupt* messages in various ways and, although without having the underlying topology control, can profit by their malicious actions in order to obtain a better *utility* or other kind of advantages. Of course, these actions cannot be prevented by the SPSS itself.

Essentially, we focus on services relying on the PSS for connectivity. The issue we want to address is to prevent the spreading of forged information in these services. A new gossip primitive to extend the standard gossip scheme is the solution scheme we would like to achieve. Extending the gossip scheme would make easier to adapt many standard gossip protocols in order to provide more secure services.

In order to define clearly our second problem, we have to deal with some issues. Basically, we must define exactly the following: (1) what kind of security or guarantee we would like to have, (2) in which exact context we would like to operate or, in other words, which kind of gossip service class can we target?

6.1.1 The scenario

We focus our attention on a specific class of gossip services. Our target class is made by generic information dissemination services that follow this simple schema: each node A and B exchange respectively a set of *items* s_A et s_B and both sets must be non-empty. We do not pose any restriction on the actual action performed to produce the new node state (see 2.1). The actual neighbor selection is performed in a random fashion. The neighbor is picked from an underlying topology that is provided by the presence of a PSS instance. Therefore, we assume each node also runs a PSS instance.

We consider an *item* as a generic representation for any protocol specific data unit (e.g., event, advertisement, description, sensor datum, etc.). As items are exchanged among peers, several copies of a single item may exist in the network. An item can be uniquely identified by the node ID of its source node and by a sequence number.

In this context, we consider that the attackers *forge* or *corrupt* the set of received items in their cache; the set held in cache is then forwarded to a neighbor. The reason for the malicious behavior may vary and is application dependent, but we consider that the malicious behavior provides some specific advantage over the other (non malicious) peers.

In gossip overlays, the fast diffusion of the information is a remarkable property, but it may become a side effect if malicious information is diffused instead. In fact, a few attackers may be able to subvert the application behavior by polluting the system with bogus information. Our goal is to limit the spread of the corrupted items by the malicious nodes on behalf of other nodes.

As can be argued, the actual item corruption process is highly application dependent.

In the absence of any mechanism to detect and remove corrupted items the system is likely to be flooded by the action of the attackers. In order to secure the system, the following issues must be addressed:

- **Access control**: the entry point to the system must be regulated. A *Certification Authority* (CA) in order to certify a public key for each peer can be a viable strategy. The CA intervention is required as a bootstrap step and has no other impacts.

- **Sender authentication**: as in large-scale gossip networks the messages are likely to "random-walk" across many other peers in order to reach a destination; therefore, the receiving node cannot make assumptions about the sender of the message or item. To uniquely identify the sender's item, we require each message to be signed by its original sender.

- **Message integrity**: as any message is likely to be forwarded among many peers, it is easy for an attacker to corrupt any item. However, if the previous requirement holds, any peer can check for the integrity of the received items by verifying the digital signature of the item's sender at any time.

The use of cryptography can trivially solve our aim to prevent any attempt of forging the items. In fact, by checking all the items at every gossip exchange would solve the problem. However, gossiping large amounts of items at a time and having multiple services relying on this technique could have a severe impact on the processing resources.

do forever
wait(Δt)
neighbor = SELECTPEER()
SENDCACHE(neighbour)
$neigbor_{cache}$ = RECEIVECACHE()
checkItems($neighbor_{cache}$**)**
myCache.UPDATE($neighbor_{cache}$)

(a) Active Thread

do forever
n_state = RECEIVECACHE()
SENDCACHE(n_state.sender)
checkItems($neighbor_{cache}$**)**
myCache.UPDATE($neighbor_{cache}$)

(b) Passive Thread

Figure 6.1: The gossip scheme extended by the anti-forge `checkItems()` primitive.

We have to stress that a typical PSS implementation seems to fit in the general class we have identified. Of course, we can ask: *why not to use the technique we are introducing to prevent the hub attack as an option to the* SPSS*?* The reason is due to one of the assumptions of the hub attack: the attackers are colluding as they need to share their secret keys to sign correctly the IDs in each message. For this reason, the technique we are introducing (see Section 6.2), based on cryptography, would never detect any malicious action during a hub-attack.

6.2 The anti-forge technique

The technique we are going to use, has been already successfully adopted in order to limit the spam of corrupted items in mesh of wireless routers [GJGvS07].

In our scenario we have a dynamic overlay topology (PSS) instead of a static set of routers. This fact makes the diffusion of corrupted items faster than in the wireless scenario.

Essentially, the anti forge technique aims to provide a new gossip primitive: `checkItems()`. The primitive is designed to run before the merge phase of the node's state as shown in Figure 6.1. Both the `checkItems()` and the SPSS `checkIDs()` (see Section 5.1) primitives follow the same design scheme, but their actual algorithm is completely different.

The key idea is that each peer performs a *probabilistic* integrity check over the received item set according to a system-wide P_{check} checking parameter. This features ensures the lightweight nature of this solution as it provides a very good ratio among effectiveness and resource consumption (see [GJGvS07] and Section 6.3).

The integrity check involves the verification of the selected item's digital signature. Only successfully verified items and not checked items are allowed to be merged in the new peer's state. Any invalid item, of course, is discarded. In addition, valid items are marked with a `checked` flag.

The actual algorithm run by the anti-forge primitive is the following.

1. $\forall\, item_i$ with prob. P_{check}, verify digital signature of $item_i$
2. if $item_i$ is valid, $item_i \leftarrow$ `checked`

6.2.1 Corruption attack model

The design of the anti-forge technique has severe implications for how an attacker can behave. Marking the items as `checked` can be considered as an *incentive to cooperate* among nodes.

First, all peers are supposed to verify a certain amount of items according to the P_{check} parameter; if an attacker refuses conform to this rule, it would raise suspicion as it is easy to calculate how many checked items a node should receive in a cache exchange (e.g., related to P_{check} and the cache size).

Second, the application of the `checked` flag is under the responsibility of the last forwarder. If a forwarder sends a checked item, he tells explicitly that he has previously checked this item instance and the item is valid. Therefore, if by lying we run the risk to raise suspicion if the receiving neighbor executes an integrity check on that item. In this case the attacker would be trivially discovered and banned from the system.

For these reasons, an attacker can corrupt all items with the exception of the items marked as `checked`. Malicious nodes are supposed to be careful and

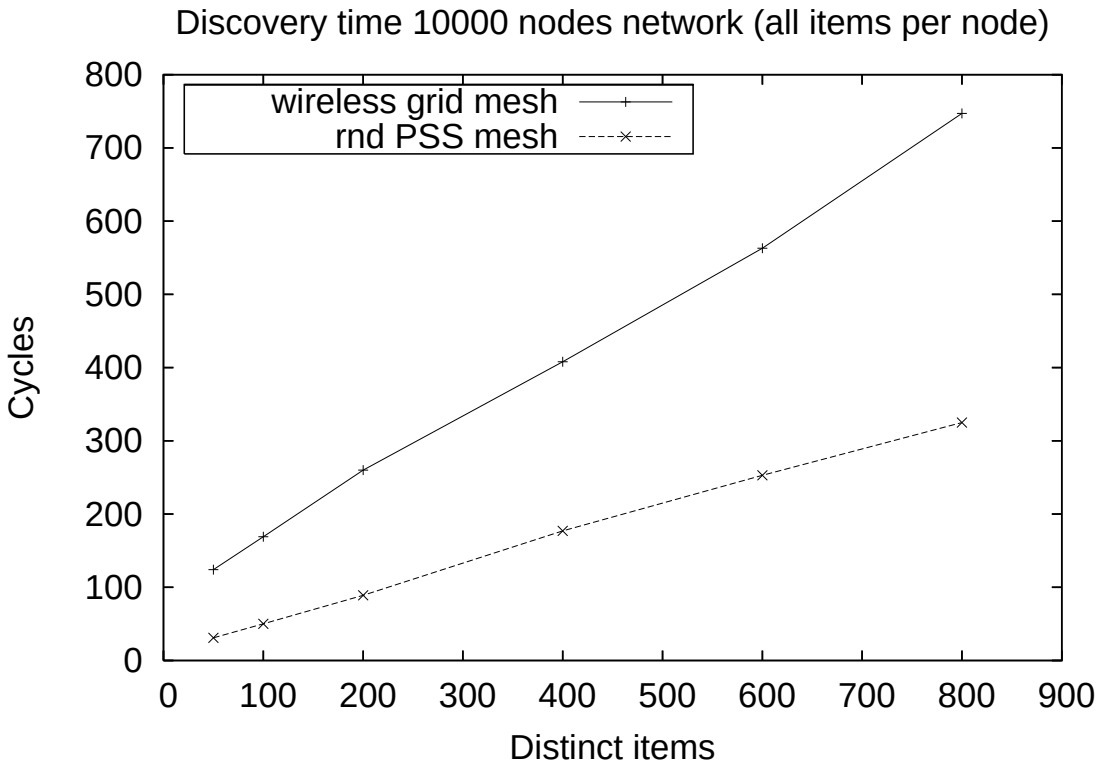


Figure 6.2: Items discovery speed comparison among the overlay and wireless scenario. The speed is expressed in terms of cycles required to discover the amount of distinct items on the x-axis by all the nodes. The network size is 10,000 nodes.

do not run the risk to be trivially discovered, thus they are forced to perform integrity checks with P_{check} probability and to only corrupt items that are not marked as `checked`.

Of course, the attacker's power is much more limited.

6.3 Anti-forge technique evaluation

We tested the effectiveness of the anti-forge technique in our particular overlay environment. Our goal is to check the effectiveness of the probabilistic check in

limiting the spread of the corrupted items in our overlay setup.

We perform our experiments over three distinct network sizes: 1000, 2500 and 10000 nodes. The underlying topology is managed by a PSS instance, therefore it looks like a random graph from each peer's point of view. Different concentrations of malicious node are considered: 1, 2, 5, 10, 20 and 30%; here, the malicious nodes act on the dissemination service only.

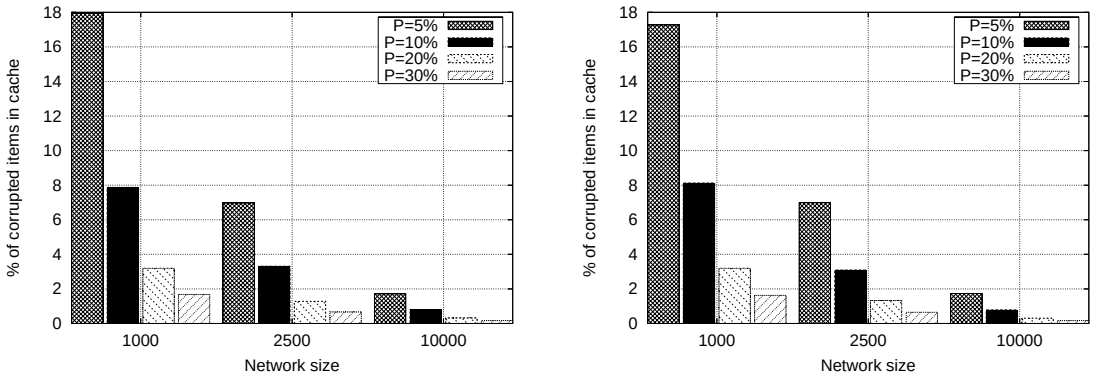
The actual gossip algorithm used as a dissemination service to diffuse the items is basic-shuffling (see Section 2.2.2) tuned by a special setup; essentially, the differences are in the following: (a) generic items are exchanged instead of node IDs, (b) the set size of the exchanged items is equal to the item's cache size (the cache's whole content is exchanged) and (c) the random peer selection is performed by a PSS instance running on each node as well as the dissemination service.

The cache size used by the item diffusion mechanism (basic-shuffling) is 50 or 100, while the PSS cache size c is set to 20. In the following, when we refer to the generic term "cache", we refer to the diffusion mechanism cache.

Figure 6.2 shows a brief summary of the difference in speed among our scenario and the wireless scenario described in [GJGvS07], in which wireless routers are arranged in a grid-like fashion (4 neighbors set-up). The speed is expressed in terms of time (cycles) required to discover all the distinct items by all the nodes in the system. The x-axis reports the actual number of distinct items injected in the system. In addition, the dissemination service uses an item cache of size 50. In this experiment there are no attackers and hence no corrupted items; the experiment is just to show the speed dominance of our set-up. The use of the basic-shuffling as a diffusion algorithm is explained in [DvS06].

The wireless set-up is much slower than the PSS overlay; this fact is expected, as the PSS provides a dynamic communication layer that spreads the information faster. Not only does dynamism play an important role, but also the higher node degree of the PSS topology is a great boost for performance.

Figure 6.3 shows the effect of the malicious nodes when they start corrupting



(a) 1% malicious nodes

(b) 5% malicious nodes

Figure 6.3: Average cache pollution (percentage of corrupted items in the items cache) according to network size (1,000, 2,500 and 10,000 nodes) and distinct values of P_{check} (%5, %10, %20, %30).

the entries. The attackers corrupt all the items they are exchanging with a neighbor, except for the items marked as `checked`. The figures are generated after 500 cycles. Each figure shows the average percentage of corrupted items in each node's cache according to the network size and the checking probability value (P_{check} = {5,10,20,30%}). In Figure 6.3(a) the size of the set of the attackers is 1% of the network size, while in Figure 6.3(b) it is 5%. The levels of corrupted items in the cache are very similar with both concentration of attackers (1 and 5%). Using the same proportion of attackers and the same checking probability P_{check}, the smaller networks tends to be more polluted. This fact can be explained by the degree of the underlying PSS topology; a degree of 20 in fact, is quite large for a 1,000 nodes network and the corrupted items can be spread faster than in the other network sizes and a higher P_{check} value is needed to limit the pollution. In the bigger network instead, a 5% P_{check} value is fine to limit the corrupted items to a negligible level (e.g., an average of 2% of the node's item cache size).

The time required for the system to reach a stable pollution state in the item cache is about 50 cycles in the worst case, corresponding to the smallest P_{check} value as shown in Figure 6.4. The upper line in the figure shows how the number

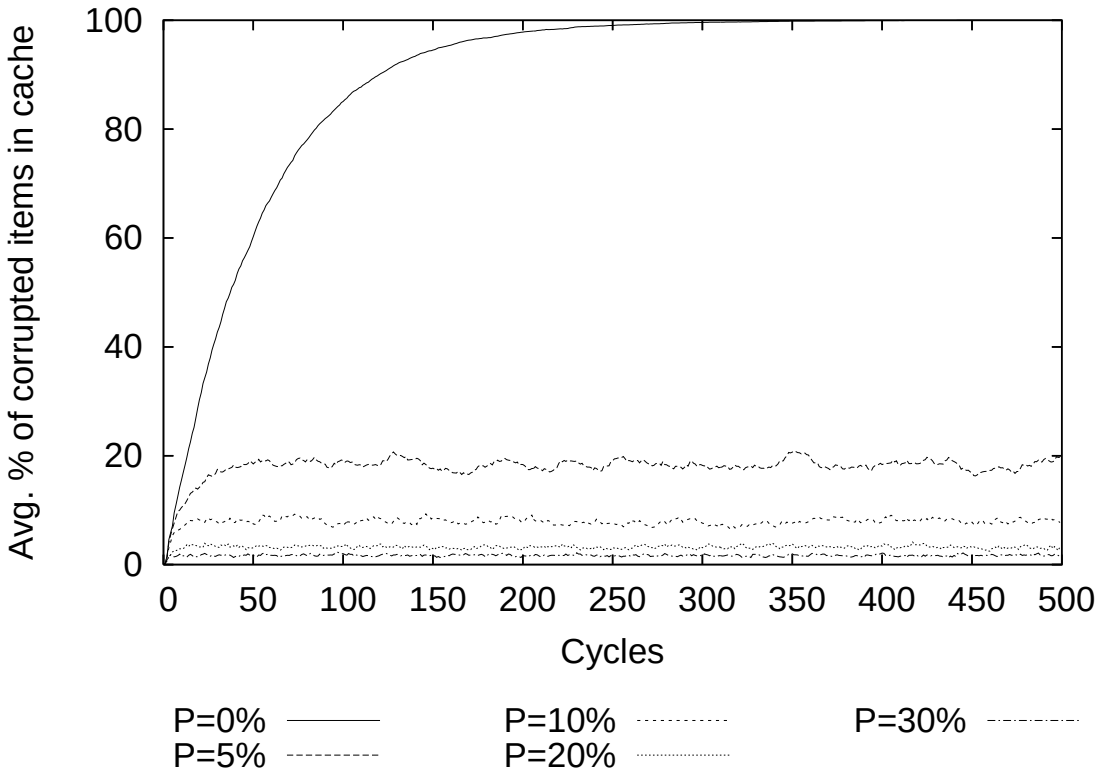


Figure 6.4: Time required to corrupt the node's cache. The upper line shows what happens without any checking attempt, while the other lines show a distinct checking probability (e.g., P_{check} = 5, 10, 20 and 30%). The network size is 1,000 nodes.

of corrupted items in the cache increases until all node's cache entries are corrupted. Figure 6.4 depicts the situation regarding a 1,000 nodes network, but for the other network sizes, the time-scale values are similar.

The main difference between our scenario and the wireless one described in [GJGvS07], is how the corrupted items are distributed in the network. In our set-up, the corrupted items tends to be uniformly distributed among the node's caches, while in the wireless one the items have a short range of influence (measured in hops). This is one of the reasons why the corrupted items spread for a longer distance. We measure this distance in hops, each time a corrupted in-

stance is exchanged, its hop counter is incremented. This process stops when the corrupted item is discovered and discarded by the check mechanism.

Figure 6.5 shows our hops measurements in the bigger (10,000 nodes) network. The picture on the left (Figure 6.5(a)) shows the hop distance travelled over time according to distinct P_{check} values, while the picture on the right (Figure 6.5(b)) shows the hop distance travelled with distinct P_{check} values, according to different malicious nodes concentrations (1 and 5%).

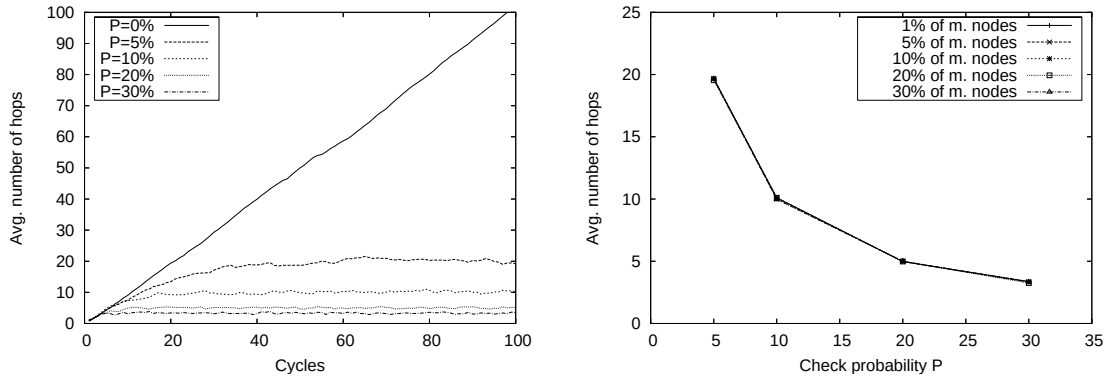
In Figure 6.5(a), the average hop distance travelled reaches a stable state in a few cycles (e.g., 20). As the dissemination is faster and almost uniform among all the peers in the network, the corrupted items travel more distance than in the wireless context (see [GJGvS07]). Of course, without any check a corrupted item will travel forever among peers as shown by the topmost line. The distance travelled grows almost linearly.

Figure 6.5(b) depicts two distinct properties: (1) the average number of hops (distance) traveled by corrupted items is inverse proportional to the actual P_{check} probability value; (2) the distance results are independent from the number of malicious nodes (1, 5, 10, 20 and 40% of the network size) in the system, as all lines in the plot are almost identical.

6.4 A case study: evaluating the SPSS and the anti-forge technique together

After having analysed the anti-forge technique in our overlay set-up, securing a generic information dissemination service, we allow the presence of a malicious PSS layer, in which malicious nodes play the hub attack.

Essentially, we are interested to see if and how the hub attack can increase the malicious item diffusion rate spread by the dissemination service. In addition, we show how the presence of both techniques can limit to a negligible level the effect of the malicious nodes over both services.



(a) Distance over time

(b) Distance over P_{check} values

Figure 6.5: Hops distance travelled by corrupted items in a 10,000 nodes network. The picture on the left shows the distance travelled over time according to distinct P_{check} values; the 5% of malicious nodes have joined the network. The figure on the right instead, shows the distance travelled with distinct P_{check} values, according to different malicious node concentrations (1, 5, 10, 20, 40%); the plots *overlap* showing that the distance travelled is independent from the number of malicious nodes.

6.4.1 Scenario

We evaluate our techniques in the following scenario.

All nodes in the network (10,000) run both the (S)PSS and the item dissemination service. We allow the presence of a set of malicious colluding attackers. These malicious nodes run both the corresponding malicious versions of the services. The fact of colluding does not add any strength to our specific item corruption mechanism, but it is a fundamental requirement for the hub attack (see Section 3.2). The attacker set size k is very small: only 20 malicious nodes are present in the system ($k = c = 20$). The reason why we believe a small set of attackers is a realistic scenario, has been previously stated in Section 3.2.

Figure 6.6 shows the proliferation of forged items in the diffusion service while a hub attack is mutating the PSS underlying topology into the hub topology. Distinct P_{check} values (e.g., 0, 1, 5, 10%) are compared. The tremendous impact

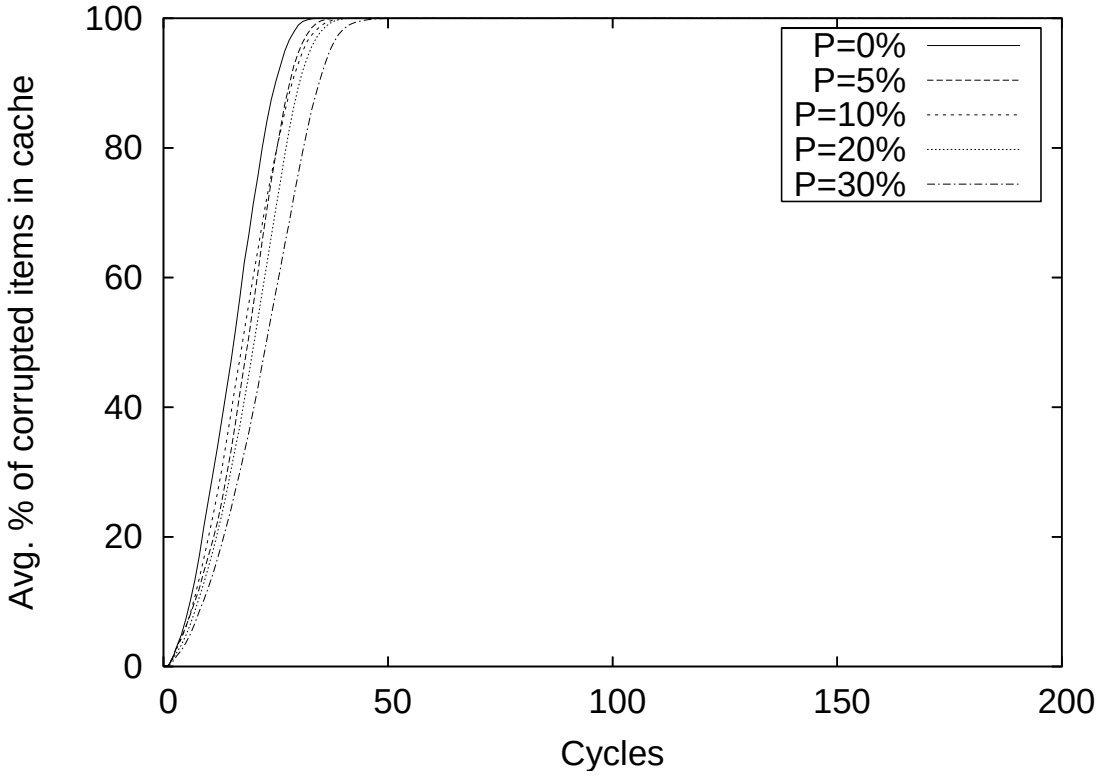


Figure 6.6: Average of corrupted items in node's caches in a 10,000 nodes network. Each node is running a PSS instance affected by a hub attack and an item diffusion service in which malicious nodes corrupt the items they forward. 20 m. nodes run the corresponding malicious version of both services. Distinct P_{check} values are compared.

of the hub topology on the forged items diffusion is shown by all the plots in the figure; less than 50 cycles are sufficient to fill the entire network with forged items, regardless of whatever effort spent in the checking process. 50 cycles is the amount of time required on average to build the hub topology.

Increasing the probability value has almost no effect, as just a few cycles of delay are introduced before the total corruption of the items in the system. We limit the maximum value of P_{check} to 30% as our aim is to keep the computational burden low.

The most surprising aspect is that this terrible situation can be introduced by just 20 attackers (e.g., 0.2% of the network size); as soon as the hub topology is completed, the power of malicious nodes increases exponentially. Therefore, ensuring the underlying topology health is a primary concern in this scenario.

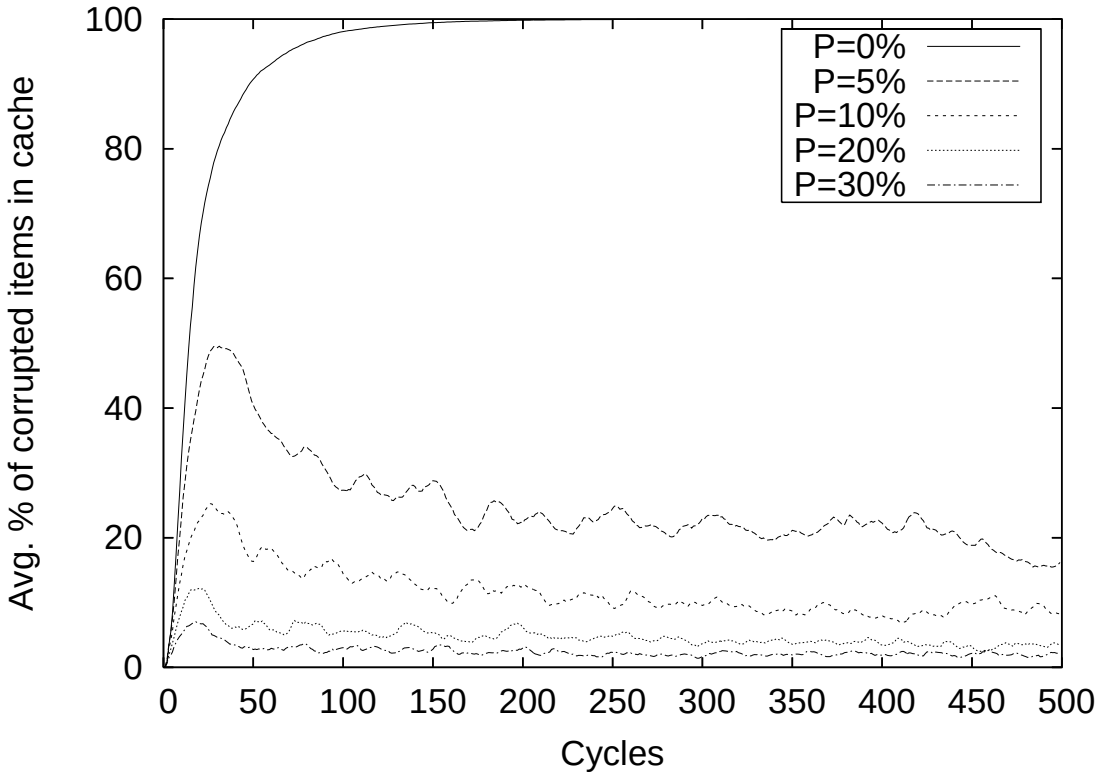


Figure 6.7: Average of corrupted items in node caches in a 10,000 node network. Each node is running a SPSS instance providing a defense for the hub attack and an item diffusion service in which malicious nodes corrupt the items they forward. 20 malicious nodes run the corresponding malicious version of both services. Distinct P_{check} values are compared.

In Figure 6.7, the PSS is replaced by an SPSS run by every node. In this setup, when the diffusion layer performs no checks, the diffusion rate of the forged items still pollute the entire network, but it proceeds much slower and in is similar to the results presented in Figure 6.4. Switching on the checking process leads

to a stable state of the node's cache pollution which is proportional to the actual P_{check} value adopted. Also the time required to achieve the stable state is proportional to P_{check} value; in the worst case (e.g., $P_{check} = 5$), 150 cycles are required.

It is easy to understand why the hub attack can speed up the diffusion of forged items. The hub topology forces each well-behaving node to exchange its cache with a malicious node at least once at every round. This process will flood any ordinary node with forged items and the probabilistic check is not enough to purge the information, unless using very high values of the P_{check} probability that would have a computational impact close to a full checking strategy. Essentially, the forged item's diffusion during a hub attack can be avoided at the expense of a higher CPU consumption in order to check all the item's digital signatures. However, this is exactly what we want to avoid, as we we are interested in a lightweight approach.

The hypothesis of using a full item check strategy when the hub topology is complete can be useful in this particular scenario only. For example, as the attackers are few in number and they are colluding, they can change the item's corruption strategy and can produce forged items on behalf of the other malicious nodes. Each forged item will be regularly signed with the corresponding malicious node key leading to no suspicions by other nodes.

The key idea to protect the diffusion service is to first ensure the regular structure of the underlying overlay. In fact, the chance to travel along random paths is a crucial aspect to ensure the effectiveness of the probabilistic item's check approach. The SPSS is an ideal candidate to provide (a) the standard PSS features (see Section 5.7) when a set of attackers strike a hub attack and (b) a secure substrate on which other services can trust for their own activities.

Chapter 7

Related work

Some real-world applications have developed techniques to tolerate malicious node behavior. In this context, the attackers exhibit selfish behavior and are usually referred as *free-riders*. Essentially, they tend to use the other peer resources without returning any favour.

BitTorrent [bit] and Scrivener [NNS+05] adopt a reputation scheme which promotes the peers who have reciprocated in the past. The difference between the two systems is that, in the former, the scheme is purely local, while the latter uses a distributed reputation scheme. A reputation scheme is a critical component and it has been show to be possibly subject to subversion, especially in its distributed flavour. A recent work [HP05], suggests that the success of BitTorrent file sharing and its relative low amount of cheaters is mainly due to user's sociological aspects rather than to technological features. In fact, its reputation algorithm seems easy to subvert. Stronger attempts to build a trusted distributed reputation scheme have also been made (e.g., EigenTrust [KSGM03]).

Splitstream [CDK+03] is a tree-based multicast protocol. It achieves load balancing by dividing the content in stripes and by using a distinct tree per stripe. To circumvent free riders, Splitstream periodically rebuilds trees and nodes keep a local reputation list regarding potentially malicious nodes.

BAR Gossip [LCW+06] is an interesting alternative gossip scheme. Using BAR Gossip, the authors propose a multimedia streaming application that guaran-

tees predictable throughput, low latency and deals with **B**yzantine, **A**ltruistic and **R**ational peers. BAR Gossip relies on two main primitives: (i) verifiable pseudo-random neighbor selection and (ii) fair enough exchange. The first feature ensures that a gossip partner can verify that its selection is really (pseudo) random, while the second feature promotes co-operation among selfish nodes.

In our context, we cannot let neighbors verify the random selection as each node's neighborhood is changing every round and the neighborhood size is limited. BAR gossip achieves this feature sacrificing dynamic membership: each participant must register first at the broadcaster node before the streaming starts. After the multimedia event is started, no nodes can join or leave. Essentially this means that the BAR topology is a clique in which "everyone knows everyone".

Again, as each node's neighborhood is changing every round, it is very hard, if not impossible, to build a reliable reputation scheme in our environment. In fact, by the time reputation have settled, attackers would already have subverted the network.

7.1 Attacks

In the following we present the most interesting attack approaches along with their proposed solution (if any). Much of the emphasis is focused on *structured* overlays, but very little attention has been paid to unstructured networks that are probably even more sensitive [DKK+05].

7.1.1 Sybil attack

One of the first P2P-oriented attacks is the "Sybil" attack [Dou02]. In the Sybil attack, any malicious peer can adopt many distinct identities and can therefore control a substantial fraction of the system. The redundancy often adopted by P2P systems to mitigate the presence of malicious peers does not help as it requires the ability to distinguish whether two identities are actually different or

not. The author shows that without a central authority, this malicious behavior is always possible. The Sybil attack is still a cornerstone attack model in overlays, as there is no way to solve it in a distributed manner suitable for large-scale overlays; in fact, a few extreme and unrealistic assumptions are required.

An obvious consequence of this attack model is the requirement to adopt a Central Authority (CA) in order to provide a *minimal* level of trust to the system. This level is minimal and not sufficient, as malicious peers can easily obtain distinct "legal" identities from the authority. Many solutions have been proposed to solve or mitigate this issue. They are based on the idea of considering identities as "precious" resources and to force peers (or users) to pay (see [CDG+02]) or to limit the number of identities issued over time. However, these mechanisms tend to be very complicated and they have a non-negligible risk to be exploited as well. What is worst, is that these approaches tend to discourage well-behaving peers (or their users) from joining these overlays.

Two recent works [Bor06, YKGF06] however, seem promising; the first one is based on computational puzzles; the second one, is based on the "social network" among user identities. This novel scheme is based on the fact that an edge between two nodes reflects a human-established trust relationship between the users themselves. Malicious nodes instead, can hardly establish trust relationships.

7.1.2 Eclipse attack

The *Eclipse* attack is focused on DHT overlays and proposed in [SCRD04]. It is a generalisation of the Sybil attack. An attacker can use first a Sybil attack to trigger an Eclipse attack by generating fake distinct peer identities to populate the neighborhood of well-behaving nodes. Correct nodes are thus "eclipsed" from the overlay. Any solution for the Sybil attack may be useless in this case, as the Eclipse attack works at the overlay management layer. Essentially, it modifies the topology by polluting the correct nodes neighborhood links; we adopted a

similar approach in our attack model (see Section 3.2), but we have a different underlying system model.

The authors also present a defence scheme based on degree auditing and they claim their solution is general enough for unstructured overlays, but their evaluation is focused only on structured overlays. However, in [NCW05] the authors discourage using auditing because it is hardly manageable in a distributed fashion and because the attackers can elude the sanctions generated by the auditing process (e.g., using the *Sybil* [Dou02] attack). The approach we will propose (see Chapter 5) to resist to our attack model instead, it is not based on distributed auditing mechanisms.

A distributed auditing process is in general weak because it is based on *artificial incentives* (versus *genuine incentives*), a particular kind of incentives in which a node has only to *appear* to co-operate. However, the design of genuine incentives it is not always a viable solution and the algorithm designers are forced to adopt the weaker kind of incentive. These categories are just an example of *Distributed Algorithmic Mechanism Design* (DAMD) (see [SP03, NCW05]).

7.1.3 Poisoning attacks

The *index poisoning* attack [LNR06] focuses on lowering the information quality of the indexes responsible to map hash keys to the current file locations. The poisoned indexes, for example, may bind the hash keys to random hosts addresses, but also other strategies can be adopted. This simple approach works because these P2P systems do not check the file advertisement locations. The approach is suitable for both DHT (e.g., Overnet) and non DHT overlays (e.g., FastTrack). The index poisoning affects the provided QoS by poisoning the application information (e.g., the indexes) rather than damaging the topology structure. Essentially, the system is polluted by bogus high-rated entries, but the system structure (topology) is not affected. The author's countermeasure based on rating the sources works because it is assumed that the routing layer is still working fine.

It is interesting to note that just one malicious node may poison a sufficient number of high-rated entries to cause trouble. For example, this kind of attack is used in real file-sharing systems by companies that are worried about the violation of their intellectual properties; downloading a famous artist mp3 or mpeg file and listening or watching to an anti-piracy spot is not uncommon.

In [NR06] the authors combine the previous index poisoning attack with the *routing table poisoning* in DHT file-sharing systems. This combination leads to an effective DoS attack. The latter attack focuses on making the victim host an overlay neighbor of many of the overlay participants. When a poisoned peer forwards a message, it may select the victim host as the next neighbor. Due to the millions of active peers in many P2P system, a significant fraction of peers ("zombies") tend to flood the victim host with messages. The victim may not be part of the P2P system. As in our attack, the routing poisoning attack disrupts the topology infrastructure, but, while in the latter the attackers are injected in ad-hoc overlay positions [1] by the hacker, in our scenario the attackers have to climb up to the leader position. The authors discuss some viable countermeasures based on checking the source by contacting it, but we have seen in our case this approach is not sufficient at all since the topology mutation is too fast.

7.1.4 Other attacks

In [BKS04], the authors propose a methodology to eliminate the vulnerabilities of gossip-based multicasts to DoS attacks. The solution is based on low-level socket techniques; it focuses on limiting and eliminating bogus message fragments, but the system relies on a fixed neighborhood overlay. Essentially, it protects the information quality, but not the overlay infrastructure that allows the application to work. In [WLC03] instead, the impact of overlay topology is analysed in order to tolerate DoS attacks by hiding an application's location. Different topologies have distinct levels of effectiveness in a location hiding purpose. This paper en-

[1] Corresponding to the most preferred keys.

forces our claim about the relevance of preserving the actual overlay topology.

In [JMB03] the authors propose a "frequency" attack in unstructured gossiping networks in which attackers communicate more frequently than others and diffuse illegal information very fast. Their defense is based on having an offline *Certification Authority* (CA) and using an auditing scheme based on message history.

In [CDG+02], Castro et al. studied attacks aimed at preventing fair message routing in DHT overlay (Pastry [RD01]). As we also did with our SPSS, they identified their secure routing as a key building block that can be combined with higher level services.

Chapter 8

Concluding remarks and future directions

Secure gossiping techniques are becoming increasingly necessary in P2P applications. The research presented in this thesis addresses the problem of developing gossip-based solutions for large-scale distributed systems, in the presence of malicious nodes. This problem is difficult as it needs to be carried out in a collaborative fashion.

In these kind of systems, in which no strict control on users and software versions can be ensured, the presence of malicious nodes should be considered the norm rather than the exception. Therefore, the design of P2P services and protocols must deal with this new concept.

The attack model is based on the fact that each participant holds a small list of references of other nodes, called cache. This cache structure is regularly updated among nodes through message exchange (gossiping). In this manner, each node is offered a fresh list of nodes participating in the network. In the attack model, a small group of colluding attackers forge special messages such that entities always refer to a malicious node. The effect of this malicious behaviour is that the attacker's group isolates well-behaving nodes from the rest of the network. In the second model, a smaller set of colluding nodes

In particular, we focused on securing a specific and well known gossip service, the Peer Sampling Service (PSS) from our generic attack model. We have shown the severe consequences our simple attack model can lead to.

In addition, the consequences of the attack are not limited to the PSS, but we applied our work to other cases such as aggregation, (latency-aware) topology management and optimal superpeer selection. These higher-level (gossip) services, relying on the PSS, suffer tremendously from the attacker's presence.

Our proposed solution, the Secure Peer Sampling Service (SPSS), is aimed to be "elegant" in the sense of integrating nicely in the standard gossip scheme. It is also robust and requires minimal effort to be carried out by well-behaving nodes. Essentially, a trusted entity (TRUSTED PROMPT) is used to identify malicious nodes using statistical analysis on reports sent by well-behaving nodes. The trusted entity can also restore the state of attacked nodes.

A fully decentralised evolution - i.e., without the trusted entity - of this solution is also presented. By using multiple random overlays and extra data structures, each node is able to build its own suspicion statistics in order to detect the malicious nodes with high probability.

It has been shown that our solution, the SPSS, is: (a) successful in reducing the malicious node's effect to a negligible level, (b) is abstracted by a basic primitive function that fits in the standard gossip scheme and, (c), it can be fully decentralised if required. In addition, the attack model we adopted can be considered a threat not only for the PSS itself, but also for other services relying on it.

Finally, we have presented a second generic technique aimed to prevent the diffusion of forged information in gossip diffusion services. We provided an example scenario in which both the SPSS and this diffusion prevention technique are combined to protect the system. This technique has been already applied in an ad-hoc (wireless) context [GJGvS07], while in this work, it has been applied to our overlay network scenario.

8.1 Future directions

The SPSS technique in particular is based essentially on a heuristic that has been validated through simulation; however, future work is needed to achieve a better

and more formal understanding of its working.

Future work may also involve the adoption of a real world implementation prototype to verify the results achieved by simulation. A suitable environment for the deployment can be the PlanetLab [NPB03] testbed.

References

[AKR+05] Micah Adler, Rakesh Kumar, Keith W. Ross, Dan Rubenstein, Torsten Suel, and David. D. Yao. Optimal peer selection for p2p downloading and streaming. In *Proc. of IEEE Infocom*, Miami, FL, March 2005.

[BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[bis] The bison project. http://www.cs.unibo.it/bison/.

[bit] Bittorrent. http://bitconjurer.org/BitTorrent/protocol.html.

[BKS04] Gal Badishi, Idit Keidar, and Amir Sasson. Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast. TR CCIT 477, Department of Electrical Engineering, Technion, March 2004.

[Bor06] Nikita Borisov. Computational puzzles as sybil defenses. In *Proc. 6th IEEE International Conference on Peer-to-Peer Computing (P2P '06)*, pages 171–176, Washington, DC, USA, 2006. IEEE Computer Society.

[bri] Brite internet topology generator. www.cs.bu.edu/brite/.

[BS06] Salman A. Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *Proc. of INFOCOM'06*, Barcelona, Spain, April 2006.

[BT01] Paul Bettner and Mark Terrano. 1500 archers on a 28.8: Network programming in age of empires. In *Proc. 14th Game Developer Conference (GDC)*, mar 2001.

[CDG+02] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36:299–314, 2002.

[CDHR02] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Antony Rowstron. Exploiting network proximity in distributed hash tables. In Ozalp Babaoglu, Ken Birman, and Keith Marzullo, editors, *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, pages 52–55, June 2002.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, October 2003.

[CDKR02] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002.

[CJ05] Andrea Ceccanti and Gian Paolo Jesi. Building latency-aware overlay topologies with QuickPeer. In *Proc. of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS)*, pages 24–29, October 2005.

[CMAP04] M. Costa, M.Castro, A.Rowstron, and P.Key. Pic: Practical internet coordinates for distance estimation. In *Proc. of ICDCS'04*, 2004.

[CRSZ01] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proc. ACM SIGCOMM*, pages 55–68, San Diego, CA, USA, August 2001.

[CRW01] A. Carzaniga, D. S. Rosemblum, and A. L. Wolf. Design and evaluation of a Wide-Area event notification service. *ACM Trans. on Computer Systems*, 19(3):332–383, August 2001.

[DBK+01] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. In *Proc. of 8th Workshop on Hot Topics in Operating Systems (HOTOSVII)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society.

[DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *Proceedings of the ACM SIGCOMM '04 Conference*, Portland, Oregon, August 2004.

[DGH+87] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proc. of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC87)*, pages 1–12, 1987.

[DKK+05] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. In *Proc of ACM Sigmetrics Conference*, 2005.

[Dou02] John R. Douceur. The sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *IPTPS*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 2002.

[DvS06] Spyros Voulgaris Daniela Gavidia and Maarten van Steen. A gossip-based distributed news service for wireless mesh networks. In *Proc. 3rd IEEE Conference on Wireless On demand Network Systems and Services (WONS)*. IEEE Computer Society, January 2006.

[EGH+03] P.T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and L. Massoulié. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[EGKM04a] Patrick Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié. From epidemics to distributed computing. *IEEE Computer*, 37(5):60–67, May 2004.

[EGKM04b] Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, May 2004.

[fas] Fasttrack Home Page. http://www.fasttrack.nu.

[FJJ+99] P. Francis, S. Jamin, C. Jin, Y. Jin, V. Paxson, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: a global internet host distance estimation service. In *Proc. IEEE Infocom '99*, 1999.

[fre] The freepastry/simpastry simulators. http://research.microsoft.com/ antr/Pastry/.

[GD98] Laurent Gautier and Christophe Diot. Design and evaluation of mimaze, a multi-player game on the internet. In *Proc. International Conference on Multimedia Computing and Systems*, pages 233–236, June 1998.

[GJGvS07] Daniela Gavidia, Gian Paolo Jesi, Chandana Gamage, and Maarten van Steen. Canning spam in gossip wireless networks. In *Proc. 4th IEEE Conference on Wireless On demand Network Systems and Services (WONS)*, Obergurgl, Austria, January 2007.

[GKM01] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In Jon Crowcroft and Marcus Hofmann, editors, *Proceedings of the Third International COST264 Workshop (NGC 2001)*, number LNCS 2233 in Lecture Notes in Computer Science, pages 44–55, London, UK, November 2001. Springer-Verlag.

[Gnu] Gnutella web site. http://gnutella.wego.com.

[HA06] David Hales and Stefano Arteconi. Slacer: A self-organizing protocol for coordination in p2p networks. *IEEE Intelligent Systems*, 21(2):29–35, March / April 2006.

[HE05] David Hales and Bruce Edmonds. Applying a socially-inspired technique (tags) to improve cooperation in p2p networks. *IEEE Transactions in Systems, Man and Cybernetics - Part A: Systems and Humans*, 35(3):385–395, 2005.

[HP05] David Hales and Simon Patarin. Feature: Computational sociology for systems "in the wild": The case of bittorrent. In *IEEE Distributed Systems Online*, volume 6, 2005.

[JAB01] M. Jovanovic, F. Annexstein, and K. Berman. Scalability issues in large peer-to-peer networks - a case study of gnutella. Technical report, University of Cincinnati, Department of Computer Science, 2001.

[JB05] Márk Jelasity and Ozalp Babaoglu. T-Man: Gossip-based overlay topology management. In *Proceedings of Engineering Self-Organising Applications (ESOA'05)*, July 2005.

[JGGvS06] Gian Paolo Jesi, Daniela Gavidia, Chandana Gamage, and Maarten van Steen. A secure peer sampling service. UBLCS 2006-17, University of Bologna, Dept. of Computer Science, May 2006.

[JGJ+00] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. 4th Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, Oct 2000.

[JGKvS04] Mark Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Proc. of the 5th International Middleware Conference*, Toronto, Canada, October 2004.

[JKvS03] Márk Jelasity, Wojtek Kowalczyk, and Maarten van Steen. Newscast computing. Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, November 2003.

[JM04] Márk Jelasity and Alberto Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 102–109, Tokyo, Japan, March 2004. IEEE Computer Society.

[JMB03] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Towards secure epidemics: Detection and removal of malicious peers in epidemic-style protocols. Technical Report UBLCS-2003-14, University of Bologna, Department of Computer Science, Bologna, Italy, November 2003. presented at FuDiCo II: S.O.S, Bertinoro, Italy, June, 2004.

[JMB04] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonelli, editors, *Engineering Self-Organising*

Systems: Nature-Inspired Approaches to Software Engineering, number 2977 in Lecture Notes in Artificial Intelligence, pages 265–282. Springer-Verlag, April 2004.

[JMB05] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(1):219–252, 2005.

[JMB06] Gian Paolo Jesi, Alberto Montresor, and Ozalp Babaoglu. Proximity-aware superpeer overlay topologies. In Jean-Philippe Martin-Flatin Alexander Keller, editor, *Self-Managed Networks, Systems and Services*, number 3996 in LNCS, pages 43–57, Dublin, Ireland, June 2006. Springer. Best Paper Award.

[kaz] Kazaa Home Page. http://www.kazaa.com.

[KMG03] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3), March 2003.

[KRAV03] Dejan Kostic, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of the 19th ACM symposium on Operating systems principles*, pages 282–297. ACM Press, 2003.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proc. 12th International Conference on World Wide Web (WWW '03)*, pages 640–651, New York, NY, USA, 2003. ACM Press.

[LCW+06] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proceedidings of the 2006 USENIX Operating Systems Design and Implementation (OSDI)*, November 2006.

[LL04] Li-wei Lehman and Steven Lerman. Pcoord: Network position estimation using peer-to-peer measurements. In *Proc. 3rd International Conference on Network Computing and Applications (NCA '04)*, volume 00, pages 15–24, Washington, DC, USA, 2004. IEEE Computer Society.

[LNR06] Jian Liang, Naoum Naoumov, and Keith Ross. The index poisoning attack in p2p file sharing systems. In *Proc. of INFOCOM*, Barcelona, Spain, April 2006. IEEE.

[LRW03] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the kazaa network. In *Proc. 3rd IEEE Workshop on Internet Applications (WIAPP '03)*, page 112, Washington, DC, USA, 2003. IEEE Computer Society.

[MCR03] Ratul Mahajan, Miguel Castro, and Antony I. T. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In M. Frans Kaashoek and Ion Stoica, editors, *Proc. of IPTPS*, volume 2735 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 2003.

[MJB04] Alberto Montresor, Márk Jelasity, and Ozalp Babaoglu. Robust aggregation protocols for large-scale overlay networks. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, pages 19–28, Florence, Italy, June 2004. IEEE Computer Society.

[Mon04] Alberto Montresor. A robust protocol for building superpeer overlay topologies. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, pages 202–209, Zurich, Switzerland, August 2004. IEEE Computer Society.

[NCW05] S. Nielson, S. Crosby, and D. Wallach. A taxonomy of rational attacks. In *Proc. of IPTPS*, Ithaca, NY, february 2005.

[NNS+05] Animesh Nandi, Tsuen-Wan "Johnny" Ngan, Atul Singh, Peter Druschel, and Dan S. Wallach. Scrivener: Providing incentives in cooperative content distribution systems. In *Proceedings of the ACM/IFIP/USENIX 6th International Middleware Conference (Middleware 2005)*, Grenoble, France, November 2005.

[NPB03] Akihiro Nakao, Larry Peterson, and Andy Bavier. A Routing Underlay for Overlay Networks. Technical Report PDN–03–012, PlanetLab Consortium, April 2003.

[NR06] Naoum Naoumov and Keith Ross. Exploiting p2p systems for ddos attacks. In *InfoScale 2006: Proceedings of the 1st international conference on Scalable information systems*, page 47, New York, NY, USA, 2006. ACM Press.

[NZ02] T. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proc. 21th IEEE Infocom*. IEEE, June 2002.

[PB02] Peter R. Pietzuch and Jean Bacon. Hermes: A distributed event-based middleware architecture. In *Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCSW '02)*, pages 611–618, Washington, DC, USA, 2002. IEEE Computer Society.

[peea] Peercast P2P Radio. `http://www.peercast.org`.

[peeb] Peersim Peer-to-Peer Simulator. `http://peersim.sf.net`.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM'01*, pages 161–172, 2001.

[RHKS02] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. 21th IEEE Infocom*. IEEE Computer Society, June 2002.

[SCRD04] Atul Singh, Miguel Castro, Antony Rowstron, and Peter Druschel. Defending against eclipse attacks on overlay networks. In *Proc. of the 11th ACM SIGOPS European Workshop*, Leuven, Belgium, September 2004.

[SGB+03] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in Warcraft 3. In *Proc. of the 2nd Workshop on Network and System Support for Games*, pages 3–14, New York, NY, USA, 2003. ACM Press.

[SH06] Atul Singh and Mads Haahr. Creating an adaptive network of hubs using schelling's model. *Commun. ACM*, 49(3):69–73, 2006.

[Sky] Skype home page. http://www.skype.com/.

[SP03] Jeffrey Shneidman and David C. Parkes. Rationality and self-interest in peer to peer networks. In *2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[The] Freenet. http://freenet.sourceforge.net.

[unr] Unreal networking protocol notes by tim sweeney. http://unreal.epicgames.com/Network.htm.

[VGvS05] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2), 2005.

[WLC03] J. Wang, L. Lu, and A. Chien. Tolerating denial-of-service attacks using overlay networks – impact of overlay network topology. In *Proc. ACM Workshop on Survivable and Self-Regenerative Systems*, Oct 2003.

[YGM03] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proc. 19th IEEE International Conference on Data Engineering (ICDE'03)*, volume 00, page 49, 2003.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 267–278, New York, NY, USA, 2006. ACM Press.

[ZS04] Serafeim Zanikolas and Rizos Sakellariou. Towards a monitoring framework for worldwide grid information services. In Marco Danelutto, Marco Vanneschi, and Domenico Laforenza, editors, *Proc. of the 10th International Euro-Par Conference*, volume 3149 of *Lecture Notes in Computer Science*, pages 417–422. Spinger, 2004.