Identifying Malicious Peers Before It's Too Late: A Decentralized Secure Peer Sampling Service

Gian Paolo Jesi, David Hales
Dept. of Computer Science,
University of Bologna (Italy)
E-mail: {jesi,hales}@cs.unibo.it

Maarten van Steen
Dept. of Computer Science,
Vrije Universiteit Amsterdam (The Netherlands)
E-email: steen@cs.vu.nl

Abstract

Many unstructured peer to peer (P2P) systems rely on a Peer Sampling Service (PSS) that returns randomly sampled nodes from the population comprising the system. PSS protocols are often implemented using "gossiping" approaches in which connected nodes exchange their links in a randomized way. However, such services can be defeated easily by malicious nodes executing "hub attacks" which distort the PSS such that all nodes in the network, ultimately, only gain access to malicious nodes. From this leading status - i.e. being a "hub" - the malicious nodes can affect the overlay in several ways, ranging from total network disruption to obtaining an application dependent advantage. We present a completely distributed defense against such attacks and give results from simulation experiments. The approach is generic as it is independent of the adopted PSS *implementation.*

Keywords*: P2P, overlay, security, gossiping.*

1 Introduction

P2P systems, without central servers, need to provide some method of initiating and maintaining connections between the nodes that comprise them such that all nodes form a single component. A partitioned network reduces the efficiency of the system, for many tasks, because nodes in different components cannot communicate. This can be a significant problem in unstructured systems which operate under highly dynamic environments with nodes constantly entering and leaving the system.

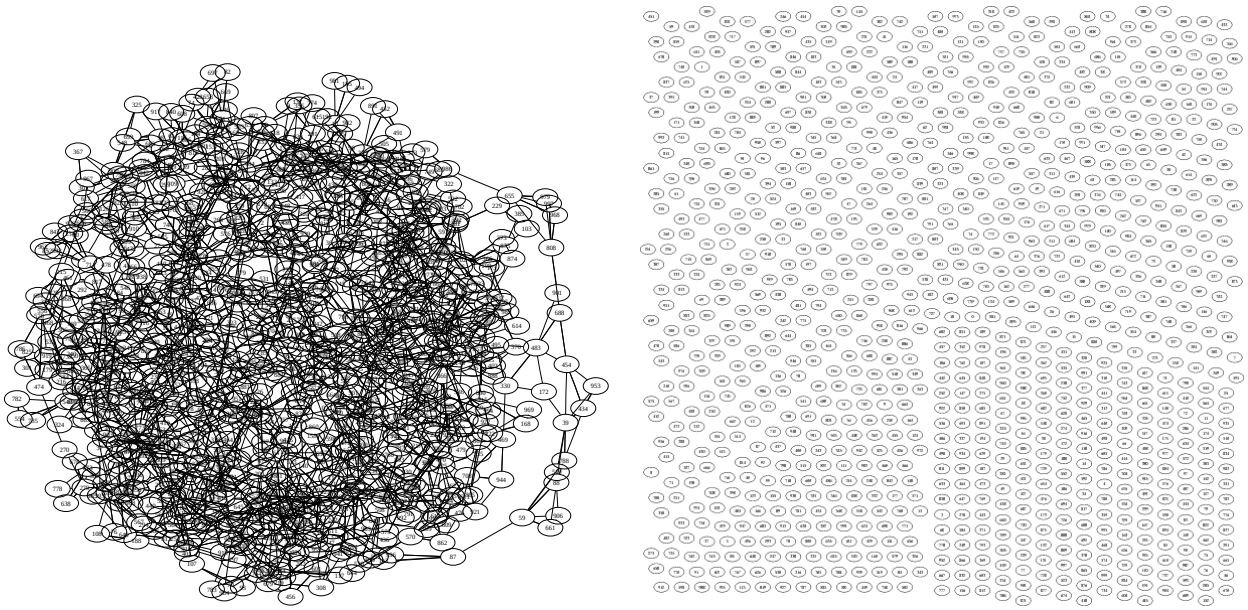
One general approach to this problem is to implement a protocol that maintains a connected overlay network between nodes which approximates a random topology. An overlay network topology consists of each node maintaining logical links to other nodes. A logical link consists of a node identifier that is sufficient to establish communication using some underlying network infrastructure (e.g. and IP address and port number over the Internet).

It is well known that a random network topology can maintain a fully connected network that is highly robust to benign node and link failure. Additionally, a random network offers short paths between any two nodes in the network which is valuable for many kinds of P2P tasks (e.g. broadcasting or routing messages between nodes).

A specific method for maintaining robust overlays with random-like topologies is through gossiping. Gossiping protocols rely on the randomized spreading of information between neighbors in a network. Typically, nodes maintain a set of neighbor links (a so-called cache or view) which indicates their currently neighbors. Periodically, each node selects some random neighbor from its view and communicates some information – i.e. gossips – which the receiving node may store and later forward to its own neighbors. The approach is loosely analogous to individuals in a social network gossiping between themselves or the spread of an epidemic in a population.

Gossip approaches are attractive because they spread information quickly and robustly over networks yet require only simple protocol implementations. A number of gossip-based protocols exist to maintain random overlay networks in unstructured P2P systems [5, 14]. Although implementations vary, the basic mechanism involves nodes gossiping their current neighbor links. In this way, using suitable update functions in the nodes, the cache (or view) can be kept up-to-date and maintain a random-like connected topology under conditions of high dynamism – where nodes constantly join and leave the network.

Ironically, however, the power of the gossip approach to spread information quickly over the entire network can become an achilles' heel if it is exploited by malicious nodes who wish to defeat the system by spreading false information to partition the network. Because information spreads so quickly in gossip networks the problem for non-malicious nodes is that *by the time they identify other nodes*



(a) A healthy random graph topology

(b) The graph after malicious attack

Figure 1. Overlay topology before and after a simple attack: the random graph depicted in (a) becomes fully disconnected in (b). The graph out-degree (constant) is set to 20, but only 3 links per node are displayed for clarity. Less than 20 gossiping cycles are required to disrupt the graph. Network size is 1000 nodes.

as malicious it is too late for them to take action since they are already disconnected from the network - i.e. their view is completely polluted by the malicious nodes.

We show that by maintaining multiple views over the network (multiple neighbor lists) nodes can identify malicious nodes before it is too late such that they can take appropriate action by placing them on an individually maintained black-list. The approach is adaptive, allowing formally blacklisted nodes to be white-listed if their behavior changes and vice versa. In addition no reputation information must be shared between nodes because blacklists and white-lists are only individually maintained. In other words, we present a fully decenralized Secure Peer Sampling Service (SPSS).

Previously, we proposed a centralized SPSS approach and obtained good results [8]. However such an approach requires one or more central trusted servers which present a single point of failure and/or add administrative overhead. We found that our distributed approach produces comparable results without producing such overhead.

The remainder of this paper is organised as follows: we describe a "hub attack" scenario in which several colluding malicious nodes are sufficient to completely partition a network using a gossip based protocol. We describe our gossip network model, followed by a brief description of the PSS implementation adopted. Then, we outline our previous centralised approach to tackle the problem and then the distributed protocol. We perform simulations that test the effectiveness of the distributed approach and compare it with the centralised approach. Finally, we briefly survey related work and conclude with a summary, open issues and possible future work.

2 Hub attack scenario

Here we describe an example of an attack scenario applicable to a gossiping topology maintenance approach. This attack is a form of the so-called "hub attack" (see [8]) and its effect is summarized in Figure 1. In a hub attack malicious nodes attempt to get other nodes to connect exclusively to them. If this is achieved, then the malicious nodes can exit the network leaving their former neighbors without any valid neighbors and hence partitioning them from the network. To illustrate the problem, consider a gossiping network in which each node maintains a list of c neighbors ($c = 20$, see Section 3.1), called its *cache* or *view* (e.g., it provides the relation "who knows whom"). When the network is performing correctly the peers are wired in a random graph topology with out-degree c and the graph is continuously rewired over time. The elements of these caches are continuously updated and exchanged between nodes during gossiping interactions.

At some point, k nodes in the system (e.g., $k = c$ in the example) start colluding and behaving maliciously exchanging forged caches; then, after a short amount of time spent in their malicious activities, they leave the network. The forged caches exchanged by the attacker nodes hold the identifiers of the other malicious nodes; this exact malicious content of the cache is always replayed at every gossip exchange.

Figure 1(a) shows the overlay in normal conditions; Figure 1(b) shows the same graph after the malicious nodes exit: within a very short time the original overlay is completely disrupted. The infection rate proceeds quickly, as can be expected from a gossiping protocol. In fact, every well-behaving node will accept the attacker's cache with high probability (see Section 5); this means that in just one exchange a node will have all its cache polluted by malicious identifiers if it is directly connected to a malicious node. In other words, all its neighbors are now malicious nodes and it can no longer *initiate* a gossip exchange with a non-malcious node.

Only a nondefeated, nonmalicious, node B can help a defeated one, say A, provided B has As identifier 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 identifiers of malicious nodes. As a consequence, both nodes will have 50% of their cache polluted; therefore, even contacting a nonmalicious node will generally also spread the pollution and increase the chance that a nondefeated node contacts a malicious one.

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 them. 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 identifiers. A very short time is required to have each node's cache completely polluted. At that point, the malicious nodes may decide to leave the network, leaving it in a completely disrupted state without any hope of recovery, as shown in Figure 1 (b).

This example refers to a small, 1000 nodes network, but as we have shown in previous work [8], no matter what the size of the network, a successful attack can be carried out swiftly. In this section we have introduced the general idea of a hub attack as applied to a gossip based topology manager service. In the next section we specify in a little more detail the generic gossip protocol approach and the specific protocol variant we used for the purposes of our simulations.

3 Gossip and attack model

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 an underlying 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. We do not address the presence of firewalls and NAT routing among peers.

Due to scalability constraints, a node knows about only a small subset of other participants. 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 relation "who knows whom" induced by the neighborhood set defines the overlay *topology*. The absence of items in the cache or the presence of incorrect or bogus IDs leads to an unrecoverable situation. In this case, a new initialization or *bootstrap* is required. In general, P2P applications provide a set of well-known, highly available nodes in order to provide a bootstrap facility and hence the initial neighborhood set.

In general, a *timestamp* is associated with each distinct node ID stored in the cache in order to eventually purge "old" ID references according to an aging policy.

The notion of time in our model is not strict because our gossip protocols need not be synchronized. 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.

In our attack model, we consider a practical scenario in which a small set of colluding malicious nodes or attackers play in the system. The size k of this set is equal to the cache size ($k = c$) in the worst case. The goal an attacker is to subvert the network in order to become a hub. The attack method involves the spreading of fabricated data through the messages gossiped among the participants which affects the logical links of the overlay. We suppose the attackers are "clever", in the sense that they operate in a beyond suspicious manner, in order to avoid being easily discovered.

A malicious node runs the actual PSS implementation as any other well-behaving node, but the content of its messages is fabricated with a specific (malicious) intent. At every gossip exchange it always replays a message containing the forged IDs of other malicious nodes in the system. This behavior is perfectly valid from a PSS point of view, as the only mandatory constraint is that cache entries are distinct. This intrinsic weak integrity constraint complies to reality. In real-world P2P file-sharing systems (see, e.g., [9,12,13]), when a peer receives an advertisement for an item, it does not check the items availability.

Other lower level details of the attack model have been

```
while(TRUE) do
 wait(Δt);
 neighbour = SELECTPEER();
 SENDSTATE(neighbour);
 n_state = RECEIVESTATE();
 my_state.UPDATE(n_state);
```

(a) Active Thread

```
while(TRUE) do
 n_state = RECEIVESTATE();
 SENDSTATE(n_state.sender);
 my_state.UPDATE(n_state);
```

(b) Passive Thread

Figure 2. The epidemic or gossip scheme.

extensively discussed in [8]. 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 it is completely partitioned (e.g., each peer has no more neighbors as depicted in Figure 1(b)).

In this work, we adopted a specific implementation of the PSS called NEWSCAST. However, our approach is generic and can be applied to any PSS implementation. Before introducing our SPSS approach in detail, we provide a brief background of NEWSCAST.

3.1 Newscast protocol

NEWSCAST is a gossip-based protocol heavily inspired by the prototypal gossip scheme depicted in Figure 2. It is a *topology manager* 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 for implementing several P2P protocols [1, 4, 6, 7].

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. The only requirement imposed to a cache is that no multiple copies of the same ID are allowed.

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 [5]) 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 due to the timestamp aging policy.

NEWSCAST is also cheap in terms of network communication (see [5]). Essentially, the number of exchanges per cycle can be modeled by the random variable $1+\phi$, where ϕ has a Poisson distribution with parameter 1. Thus, on average, we expect two exchanges per cycle. Practically, this involves the exchange of at most a few hundred bytes per cycle for each peer.

4 Centralized SPSS

In our previous work on preventing the hub attack [8], we designed the SPSS as an extension to the ordinary PSS. We summarize here the basic concepts of our previous work in order to introduce our new SPSS approach.

The system requires the presence of a central *Certification Authority* (CA). The CA is not involved in the protocol itself; it is just needed to acquire the credentials and to join the overlay. We cryptographically secure each ID structure using $[ID_A, ts_{creation}, ts_{expiration}, PK_A, \sigma]$, where ID_A is A's node identifier (see Section 3), the ts are timestamps, PK_A is A's public key and σ is the digital signature on the message.

The key idea relies in the introduction of a new primitive, `checkids()`, to verify whether the exchanged states (caches) are valid for further processing, such as merging with other caches. Essentially, the checking process has a stochastic nature and it is based on the fact that the *attackers always replay the same cache or at least the same subset of malicious nodes*. The probablity to raise a suspicion is proportional to the fraction of the common IDs found: (#common IDs) / c. When the check fails, a suspicion about the potential malicious neighbor is raised and delivered to a TRUSTED PROMPT node; it is similar to a proxy of the Certification Authority and provides peer credential management and access control.

The TRUSTED PROMPT collects the requests sent by any well-behaving node P and builds a table by logging (or updating) 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 of size c for the querying peer. To build the new cache, the TRUSTED PROMPT picks nodes randomly from the network. A node Q is selected for inclusion in a cache proportional to $1-\#(Q)/N$, where N is the (estimated) size of the network. A lower value in the frequency table corresponds to a higher chance to be present in the cache (and to be a nonmalicious node).

When considering the scalability of our approach, one may come to think that a single TRUSTED PROMPT is going to be a source of problems, but we have shown (see [8]) that, in order to improve the resilience of our distributed system, we can use multiple TRUSTED PROMPT nodes without introducing any modification to our basic algorithm. In addition,

we have shown that the message traffic generated by the queries in a large network can be sustained by just a single TRUSTED PROMPT. 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.

5 Decentralized SPSS

We believe that the main shortcoming of our centralized 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 organizations 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 setup, i.e., using a single TRUSTED PROMPT, we minimize 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 decentralized 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 decentralized solution, in which each node has the chance to detect the malicious nodes using its own resources.

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.

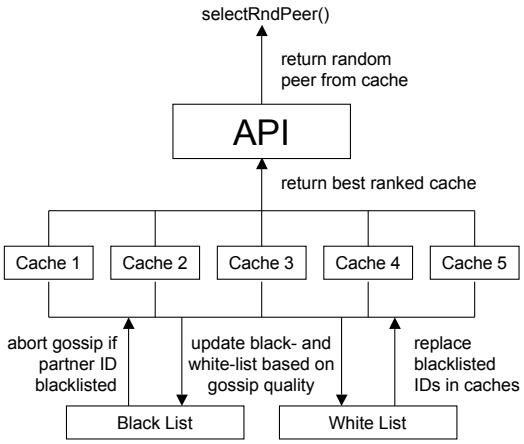


Figure 3. Schematic of the decentralized SPSS; it maintains multiple caches to support multiple random overlays. Black and white-lists screen incoming gossip requests and refresh malicious cache entries. The highest quality cache is mapped to the API to support standard peer sampling functions.

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 [12]). 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: $r = |\{cache_A \cap cache_B\}|$. 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 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 aging 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 3 shows a schematic of the main components maintained by the protocol within each node.

The algorithm: Our distributed 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 aging 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 rank value 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 aging 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., dubling 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.

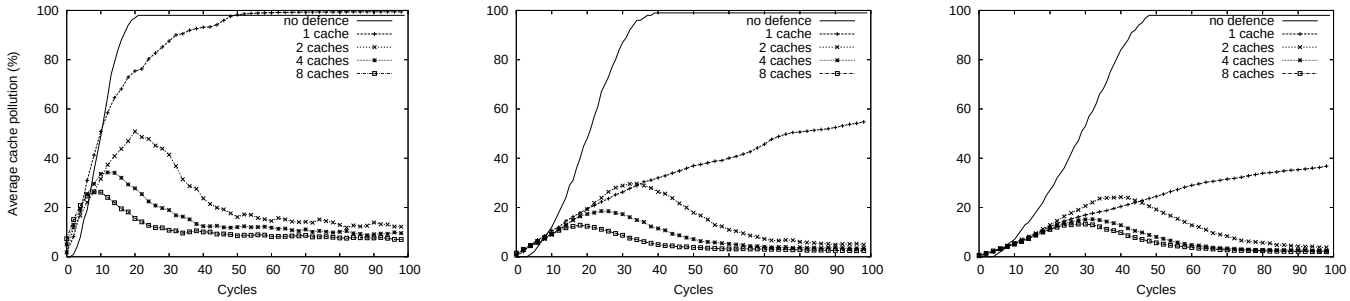
Likewise, if node A has to whitelist node B's ID, but it is already in A's BLACKLIST, the swap between the two list is not allowed until B's ID is purged from the BLACKLIST. This rule is designed to avoid that a node's PSS instance exchanging with a malicious node for the first time, would not overwrite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

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 6) they stil 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 3).

Evolutionary link: The multiple caching concept originates from previous socially inspired evolutionary models of "group selection" [2, 3]. 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.



(a) 1,000 nodes (b) 5,000 nodes (c) 10,000 nodes

Figure 4. Fully decentralized 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.

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 reinitialized 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.

6 Experimental results

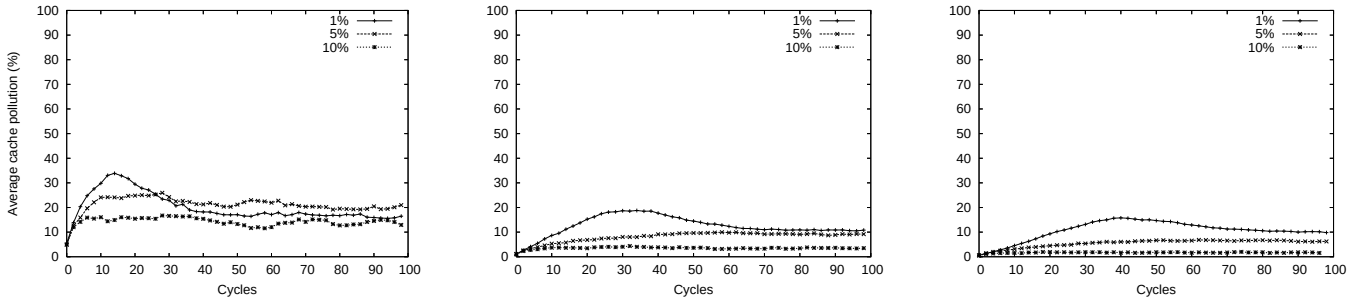
In order to evaluate our new approach, we investigate the following main issues: (a) how much time is required to achieve a tolerable[1] amount of pollution in the node's caches, (b) how many extra caches are required to prevent the attack, (c) how the performance scales according to the number of the 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 larger number of k malicious nodes ($k > c$);

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

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

Static environment: Figure 4 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 setup using a distinct number of concurrent caches per node; we have shown the results for 1, 2, 4 and 8 caches setups. As a reference, we also plotted what happens when no attack countermeasures are taken (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 setup 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 [8]). 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



(a) 1,000 nodes (b) 5,000 nodes (c) 10,000 nodes

Figure 5. Fully decentralized 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.

partitioning the network. For this reason, according to our experiments, we consider the 4 extra caches setup a good tradeoff between complexity and effectiveness.

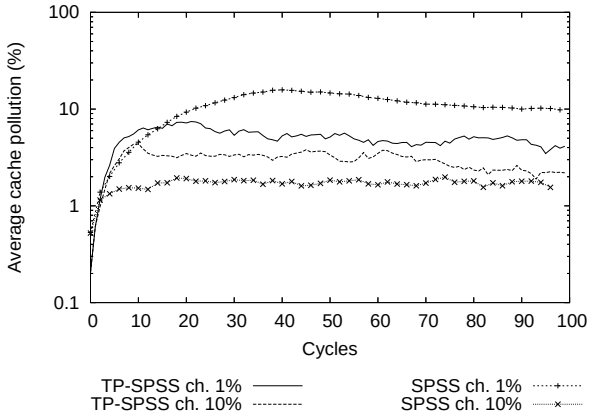


Figure 6. Comparison among our previous TRUSTED PROMPT based SPSS and the current decentralized one (4 extra caches). Two distinct churn scenarios are shown for each one. Network size is 10,000.

Churn: Figure 5 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 [10], but will allow to demonstrate the feasibility of our solution. Each node has a 4 extra caches setup.

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 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.

In Figure 6, we show a comparison between our previous centralized TRUSTED PROMPT based SPSS and the new decentralized one in the dynamic environment. The setup of the decentralized 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 decentralized 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 decentralized SPSS achieves a more stable pollution level than the centralized version.

Message overhead: The main advantage of the decentralized version over the TRUSTED PROMPT based one, is the minimal message traffic cost. 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 modeled by the random variable $1+\phi$ (see [5]), where ϕ has a Poisson distribution with parameter 1; the overall node cost per cycle is: $\sum_{i=1}^{n} PSS_i = \sum_{i=1}^{n} 1+\phi_i = 2 \cdot n$.

Extreme conditions: 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 7 depict the performance of the (decentralised) SPSS when $k > c$ malicious nodes are involved in a 10,000 nodes network.

Figure 7(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 4), 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 7(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 7(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.

7. Related work

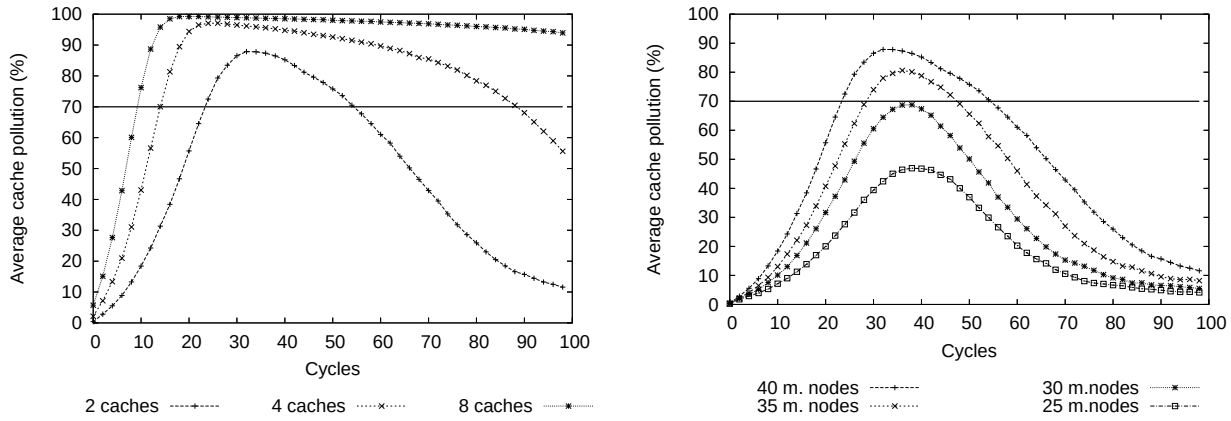
In our previous work [8], we surveyed the literature about attacks in overlay networks. Here, we just remind the attacks that are closely related with the hub attack.

The *index poisoning* attack [9] focuses on lowering the quality of the indexes that map hash keys to current file locations in file-sharing applications. A poisoned index, for example, may contain hash keys that refer to nonexisting or nonaccessible files. works because many P2P systems do not check the integrity of their indexes. Index poisoning can be applied to structured as well as unstructured overlays. However, in contrast to a hub attack, index poisoning affects only the overall QOS, but otherwise allows applications to continue functioning correctly. A typical countermeasure is to authenticate or rate the sources that insert keys. In [11] the authors combine the previous index poisoning attack with poisoning routing tables in DHT file-sharing systems. This combination leads to an effective denial-of-service (DOS) attack. In this case, a selected victim host is referenced by many other (poisoned) overlay participants, effectively signficantly increasing the probability that a message will be routed to the victim. As in our attack, poisoning routing tables disrupts the topology. The main difference is that in the hub attack, a malicious node will first need to obtain a leader position before the attack can be carried out successfully.

8 Conclusion

We have presented a decentralised secure peer sampling service (SPSS) which maintains the underlying random overlay in the presence of malicious nodes playing the hub attack. We compared our results with those obtained from a previously developed centralized method and obtained similar results.

On one hand, the decentralised approach gives up the presence of the TRUSTED PROMPT, while on the other hand allows each node to store a redundant state (multiple caches). This achieves our goal to build a fully decentralised and lightweight solution in terms of communication cost. avoid the extra infrastructure requirement and the costs associated with flux of queries delivered to the TRUSTED PROMPT. We have seen that by allowing nodes to maintain a number of independent overlays they can halt the spread of malicious information by identifying the malicious source nodes before it is too late to take action. We found that this resulted in a situation in which no healthy



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

Figure 7. 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.

node knows about *every* malicious node but every malicious node is known by *some* healthy nodes. This is sufficient to stop the attack from being successful. This insight may have implications for counteracting other kinds of infection spreading over networks and this is possible future work. The centralised SPSS achieves better results - i.e., is successful - when the entire node state is polluted ($k = c \cdot n$ attackers, with $n = 1$ in the centralised case). However, the TRUSTED PROMPT manages the node states and we can say it is part of the node state in some sense. Essentially, until the TRUSTED PROMPT is up and running the node's state it is not really completely polluted. In the decentralised approach instead, if the full state - i.e., all the caches - becomes polluted ($k = c \cdot n$ attackers, with $n > 1$ caches) the network is exposed to partitioning if the malicious nodes leave the network; nevertheless, the system does not becomes paralysed, but it can still slowly recover. We have shown an example in which we identify the maximum number of malicious nodes that can be tolerated for a particular n cache set-up.

In essence, these two approaches are two distinct philosophies that privileges distinct aspects. We do not claim that our SPSS is secure against other attack scenarios such as "Sybil attack" variants or more complex collusive attacks. Rather we propose a relatively generic and lightweight solution for plugging the obvious vulnerability of gossip based protocols to simple hub attacks. SPSS therefore has applications for augmenting several proposed protocols that currently rely on unsecured gossip based PSS such as [1, 2, 4, 6, 7].

References

[1] A. Ceccanti and G. P. Jesi. Building latency-aware overlay topologies with QuickPeer. In *ICAS/ICNS*, Oct. 2005.

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

[3] D. Hales and B. 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.

[4] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. In *ESOA*, July 2005.

[5] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Technical Report IR-CS-006, Vrije Universiteit, Amsterdam, The Netherlands, Nov 2003.

[6] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(1):219–252, 2005.

[7] G. Jesi, A. Montresor, and O. Babaoglu. Proximity-aware superpeer overlay topologies. In *SelfMan*, June 2006.

[8] G. P. Jesi, D. Gavidia, C. Gamage, and M. van Steen. A secure peer sampling service. UBLCS 2006-17, University of Bologna, Dept. of Computer Science, May 2006.

[9] J. Liang, N. Naoumov, and K. Ross. The index poisoning attack in p2p file sharing systems. In *INFOCOM*, Barcelona, Spain, Apr. 2006. IEEE.

[10] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *IPTPS*, Feb. 2003.

[11] N. Naoumov and K. Ross. Exploiting p2p systems for ddos attacks. In *InfoScale*, New York, NY, 2006. ACM Press.

[12] S. J. Nielson, S. Crosby, and D. S. Wallach. A taxonomy of rational attacks. In *IPTPS*, 2005.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, Nov. 2001.

[14] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2), 2005.