Secure Peer Sampling Service: the Mosquito Attack*

Gian Paolo Jesi
Dept. of Computer Science
University of Bologna (Italy)
E-mail: jesi@cs.unibo.it

Alberto Montresor
Dip. di Ingegneria e Scienza dell'Informazione
University of Trento (Italy)
E-mail: montresor@disi.unitn.it

Abstract

Peer sampling – the capability of obtaining a random sample from a large population of peers – is a basic building block for the gossip paradigm. Current peer sampling services have been designed for "first-class" citizens of the Internet: peers that are able to accept incoming, sporadic connections. Unfortunately, the vast majority of peers do not belong to this class, as they are behind firewalls or have private addresses. Peer sampling is thus limited to peers that are freely accessible, while the others only play a limited role. The existence of two groups, with different roles and sizes, enables one group to exploit the other. This paper introduce one of such malicious scenarios, the mosquito attack*, a solution to which is proposed and evaluated.*

Keywords*: P2P, peer sampling, overlay, security, gossip*

1 Introduction

Gossip protocols have proven to be effective in dealing with the large scale and dynamism of modern distributed systems, such as P2P networks [3–5, 9]. A key requirement for gossip protocols is the ability to randomly select gossip partners from the overall system. The *peer sampling* service (PS) satisfies this requirement, by providing peers with continuously up-to-date *samples* selected uniformly at random from the global peer population [6].

An important issue of modern PS services is their potential exploitation by malicious peers. For example, an attacker can acquire a leading position into the overlay, by forcing the PS service to provide fake samples that include only malicious nodes; such attack can evolve in the complete defeat of the system, if malicious nodes simply disappear after having gained such leading position. The diffusion of bogus, ad-hoc messages is so fast that a small group of just 20 attackers can defeat a 10.000 nodes network [7, 8] in a few gossip exchanges.

The security problem can be solved through a prestige-based *secure peer sampling* (SPS) service [8, 10], in which a simple heuristic based on *social network analysis* (SNA) [2] can detect and react to the structural changes in the network in a timely manner. Nodes which have gained a central role in the network are identified and banned.

An important issue, which is neglected by both state-of-the-art PS and SPS services [6, 8] is that not all peers in the Internet are "first-class" citizens, due to constraints imposed by ISPs to home users. In fact, private addresses and firewalls often prevent peers from accepting incoming requests, making impossible for them to play the role of full peers. In this scenario, peers may be divided in two groups: those behind a firewall (FW) and those which are not ($\overline{\text{FW}}$). Peers in the $\overline{\text{FW}}$ group can run the PS service as full members: they obtain random samples of the $\overline{\text{FW}}$ set, and they can belong to random samples provided to other nodes. Peers in FW can only obtain random samples of $\overline{\text{FW}}$, without the possibility of being randomly selected. The reason is simple: since they cannot be contacted, there is no reason to spread around knowledge about them. The presence of two groups, with distinct roles, could give rise to a new breed of attacks. Since FW nodes do not appear in the samples, they have nothing to lose when they poison the network. For example, they could gossip the IDs of a subset of benign $\overline{\text{FW}}$ nodes, in order to discredit them.

The SPS service has been designed to address an infection coming from full members of the sampling service. Is the SPS service enough to prevent an infection coming from outside the $\overline{\text{FW}}$ group of peers? The aim of this paper is to answer this question and hence to verify the effectiveness of the SPS service in a new scenario, closer to real world conditions. We introduce a novel attack model, called the *mosquito attack*, which is a particular instance of the *hub attack* [8]. We improve the SPS algorithm to deal with this novel scenario, and we evaluate our solution by comparing it with the standard SPS algorithm.

The paper is organized as follows. Sec. 2 describes our

*This work is supported by the European Commission through the NAPA-WINE Project (Network-Aware P2P-TV Application over Wise Network – www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant No. 214412

scenario and the mosquito attack. In Sec. 3 we introduce the SPS algorithm and our modification. Experimental results are presented in Sec. 4. Finally, Sec. 5 and 6 survey related work and conclude the paper.

2 Background and Problem Definition

2.1 System Model

We consider a network consisting of a large collection of peers. The network is highly dynamic; new nodes may join at any time, and existing nodes may leave, either voluntarily or due to crashes. Byzantine failures, with nodes behaving arbitrarily, are excluded from the present discussion.

Peers are heterogeneous in their communication capabilities. Peers in the $\overline{\text{FW}}$ set can participate in gossip exchanges initiated by other peers, while FW peers cannot. In other words, communication from $p \in \overline{\text{FW}} \cup \text{FW}$ to $q \in \overline{\text{FW}}$ is always possible; communication from $p \in \overline{\text{FW}}$ to $q \in \text{FW}$ is possible only in response to a previous communication from q to p; communication between $p \in \text{FW}$ to $q \in \text{FW}$ is never possible. The $\overline{\text{FW}}$ class includes machines belonging to universities, public administrations or large companies which are characterized by public IP addresses and no firewall constraints. The FW class includes machines shielded by firewalls that prevent them from being contacted from outside. This makes impossible (or very challenging) for them to play the role of a fully enabled peer.

2.2 Peer Sampling

As the network size can grow to millions, no PS service can maintain a complete and up-to-date view of the entire system. Instead, current PS services store, at each peer, a *partial view* of the network, i.e. a short list of logical links to other peers (*neighbors*). To serve later as samples, a random selection of peers is stored in partial views. As peers voluntarily join and leave, or abruptly disappear due to crashes, the PS service updates the partial views, removing old members and spreading the news about new ones.

Peers and their logical links form a dynamic *overlay topology*. An important requirement is that such topology should remain connected in spite of failures (even catastrophic ones), otherwise separate system partitions would not be able to sample each other. If local views contain random neighbors, the resulting topology is a random graph, which has proven to be extremely robust and capable to maintain connectivity even after the crash of 70% of the peers [6].

Although the approach described here is generic enough to be applied to other protocols, the SPS service considered in this paper is based on NEWSCAST [6]. In NEWSCAST, each partial views contains c descriptors, i.e. pairs (*node address*, *timestamp*). Periodically, each peer p randomly selects a gossip partner q from its local sample, and sends its partial view to q, plus a fresh descriptor of itself. Peer q replies in the same way. After the gossip exchange, p stores in its partial view the c freshest descriptors out of the $2c+1$ available (c descriptors in its old partial view, c descriptors received from q, and the fresh descriptor of q). q behaves symmetrically. The continuous injection of new descriptors gradually removes old descriptors from the network, allowing the protocol to "repair" the overlay topology by forgetting information about crashed neighbors.

Nodes belonging to FW actively participate in the protocol by initiating gossip exchanges, but they do not inject their own identifier – since they cannot be contacted, there is no reason to do it.

2.3 Attack Model

The goal of the *hub attack* is: *to subvert the network in order to achieve a leading structural position*, i.e. becoming a hub [8]. This attack method consists of spreading fabricated partial views through normal gossip exchanges, affecting the logical links of peers and thus the corresponding overlay topology. Malicious peers collude by sending around partial views that only contain fellow malicious peers; in addition, the timestamps are manipulated to postpone the dropping of the corresponding descriptors. Eventually, all nodes will link to those malicious peers, creating a hub graph.

From a PS point of view, messages sent by malicious peers cannot be distinguished from non-malicious messages. Surprisingly, this weak integrity constraint can be found, for example, in real-world file-sharing application [11, 13], where peers do not verify the validity of item advertisement they receive.

If the population of the hub attack is not too large, the prestige-based SPS service introduced in [10] can prevent the hub attack. However, it has been designed to monitor the prestige or popularity in a scenario where every peer participating in the peer sampling service is addressable.

The goal of the *mosquito attack* is: *to put discredit on a subset of nodes in order to disconnect or isolate them.* The idea is to consider scenarios for which the prestige-based SPS service has not been designed, i.e. the presence of two groups of peers with different capabilities. We assume that the $\overline{\text{FW}}$ group is entirely composed of well-behaving nodes, while the FW group is entirely made of colluding malicious peers. The size of FW is at least the same of the $\overline{\text{FW}}$ group. The prestige-based SPS service is helpless against peers in FW: they can initiate a large amount of gossip exchanges, avoiding double checks over their malicious messages. Thus, peers in FW look like a cloud of mosquitos tormenting the group of $\overline{\text{FW}}$ nodes with their bogus mes-

```
forever do
    done ← false
    R ← getSample(size)
    for each neighbor in R do
        sendstate(neighbor)
        state_q ← receivestate()
        if toss(1/size) ∧ not done do
            done ← true
            apply PSS update
        else
            getstatistics(state_q)
    recoverstate()
    wait(Δ_t)
```

(a) Active Thread

```
forever do
    state_q ← receivestate()
    sendstate(q)
    if not done do
        apply PSS update
    else
        getstatistics(state_q)
    recoverstate()
```

(b) Passive Thread

Figure 1. Prestige-based SPS algorithm.

sages.

The implementation of the mosquito attack is as follows. Malicious messages contain only a subset of the $\overline{\text{FW}}$ nodes. As in the hub attack, the descriptor timestamps are manipulated to ensure that they will be dropped as late as possible. The SPS service should react by suspecting and banning nodes in $\overline{\text{FW}}$, which are non-malicious.

Because we want to push the SPS service to the limits, we avoid cryptographic techniques and the presence of a central certification authority. Furthermore, to facilitate the task of malicious nodes, we assume that part of the $\overline{\text{FW}}$ set is composed of reliable, well-known machines, included in a list available to everybody. This static list could be normally used to bootstrap new peers – which try to perform gossip exchanges with peers in the list, until one of them replies with an up-to-date partial view. In the hand of a malicious peer, however, this list represents a security risk; it can be used to discredit peers included in them, by flooding the network with their IDs. The SPS should be considered as a *provocative* scenario, in which the SPS is forced to face the worst possible working condition. These extreme conditions may look like bad design choices. In fact, for example, it would be quite easy for a $\overline{\text{FW}}$ to detect whether a contacting peer is a FW node or not and a better – safer – design would suggest to ignore the neighbor list they provide; however, we agreed to not change the SPS basic behavior according to our aim.

3 Prestige-based SPS

The basic idea of the prestige-based SPS service is to: (a) play the actual PS implementation as usual and (b) monitor the overlay and react to structure changes when required. The key point is that the presence of the SPS service is transparent to the applications as the API of the PS service is not modified. As the infection of a poisoning attack can spread so fast, the main concern for a SPS service is to be able to build a suitable knowledge base to possibly recover its partial view in case of corruption.

In order to build the knowledge base, the SPS service makes a stochastic proportion of its gossip exchanges as "explorative", while the others are standard PS exchanges (i.e., they affect the partial view). The task of the exploration is twofold: on one hand it builds a particular sample of the current neighbor surroundings, while on the other hand it collects peers *ID*s that may become useful if and when the PS partial view becomes polluted by the spreading infection. As there is no way to detect if a neighbor has actually played the PSS or not, this behavior generates a *dilemma* that could tremendously limit the attacker's power.

The prestige-based SPS version adopts an inspiration from social network analysis (SNA) and in particular we consider the notion of prestige of peers in a directed network, where the peers that receive more positive choices are considered prestigious [2]. We adopt a (simple) technique to compute the structural prestige of a peer in terms of popularity (or in-degree). Intuitively, since the network should be random, detecting a peer showing a popularity value too distant from the average means that it could represent a network hub. Each peer builds its own knowledge about its surroundings and it does not share this information with its neighbors [13] to avoid further issues, such as the corruption of the exchanged knowledge.

Dilemma mechanism: Figure 1 shows the prestige-based SPS algorithm written in pseudo-code. Each peer can gossip with multiple neighbors in the same time unit (cycle); a subset of *size* random neighbors is selected from the partial view by the $getSample(size)$ function. Regardless of the number of neighbors selected by $getSample()$, the PS state update policy can be executed only once and with a probability of $1/size$.

On the other hand, interactions with other neighbors are used to collect prestige related information about the peer neighborhood. This mechanism produces the dilemma in which a potentially malicious peer p never knows how another neighbor q is going to use the information provided by p. In fact, the more they pollute, serving malicious IDs during the exchanges, the larger is the probability of appearing as "suspect" in terms of popularity. This feature is designed to contrast the hub attack, but it is in turn exploited by the mosquito attack.

Prestige mechanism: The information collected during the gossip exchanges is used to build the knowledge base required to detect, with good accuracy, malicious peers and to eventually repair the partial view when it becomes polluted by the presence of malicious IDs. A table structure – the *prestige table* (PTABLE) – holds the following tuple: $\langle ID, \#hits, TTL \rangle$, where each detected peer *ID* is associated to a frequency value $\#hits$ and a time-to-live value (*TTL*)

expressing the time validity of the table entry.

Essentially, each peer explores its neighborhood and collects data about the frequency, expressed in *#hits*, with which the same peer *ID* has been reported by the received partial views. In just a single gossip exchange, each peer can collect a number of items equal to the size of the partial view (c items). Although we do not pose any size restriction to PTABLE, which can hence eventually grow up to the size of the network, this is very unlikely, as the entries are purged according to an aging policy, which decrements by 1 each *TTL* at each cycle. We found experimentally that with $TTL = 2$, the average size of the PTABLE grows until about 50 items. The *TTL* of an entry is incremented by 1 each time the same entry is detected. If the malicious peers tend to acquire a network-centric position, their prestige is likely to dominate over the other entries and its value will be far more than the average peers prestige. According to this simple idea, the PTABLE object maintains the average hits value: $\#hits_{avg}$ for the collected items; the value $\#hits_{avg} + \sigma$, where σ is the *#hits* standard deviation, is used as a threshold to distinguish between potential attackers (i.e., $\#hits_p \geq \#hits_{avg} + \sigma$ for a peer p) and well-behaving peers. σ is required to produce good suspicions and to limit the production of false positives and the PTABLE size. When an entry expires (i.e., $TTL = 0$), its *ID* is collected in a WHITELIST, as it can be (with high probability) considered a well-behaving peer. If the same ID is detected in a neighbor partial view in a subsequent gossip exchange, it is not removed from the WHITELIST until its *#hits* value has eventually reached the current PTABLE $\#hits_{avg} + \sigma$ value. These calculations and the management of the knowledge base are handled by the *getstatistics*() function.

The risk of having polluted partial views is always present and therefore a method to recover them is required. In Figure 1, the function *recoverstate*() satisfies this requirement. In the average case, it is sufficient to have a set of peer *ID*s in the WHITELIST that are present in the PTABLE, but have a lower *#hits* value than the $\#hits_{avg} + \sigma$ value, or that are not present at all in the PTABLE. When the knowledge base is empty instead, the peer cache could be completely polluted by an attacker. This would happen in the early stages of the protocol when a peer has just joined the overlay. The only chance to recover, from such scenario, is to be contacted by another well-behaving and non polluted peer. As the overlay is pseudo-random, on average, this can happen quite often.

False positives check: Unfortunately, the SPS service only relies on the mechanism of the *TTL* expiration to handle false positive suspicions. However, this mechanism could not be sufficient during a mosquito attack, when well-behaving peer *ID*s are heavily diffused by the attackers in order to put discredit on them. We introduce a new mechanism which helps to recover from wrong suspicions. Essentially, each well-behaving peer choose a potentially malicious peer from its PTABLE (i.e., choose from the set: $\{n|\#hits_n \geq \#hits_{avg} + \sigma\}$) and makes an explorative PS exchange with this potentially malicious peer. If the received message contains more than 25% of already known (potentially) malicious peers (which are listed in the PTABLE), the suspicion is considered correct and the corresponding *TTL* of the malicious peer is raised. Otherwise, the suspicion is considered incorrect and the peer is removed from the PTABLE. In addition, the peer is injected in the current partial view of the querying peer. This strategy allows the victims of false positive suspicions not to be relegated at the margins of the network and to maintain a strong connectivity in the (pseudo) random graph. This extension to the SPS protocol is embedded in function *getstatistics*() and comes at the cost of an extra PS gossip exchange. To limit the traffic, every peer is allowed to perform this check only once per cycle.

4 Evaluation

We conducted an extensive set of experiments in a simulated environment to evaluate our SPS. We compare the prestige based SPS with the check for false positives respectively disabled and enabled in terms of average pollution level. Pollution is measured as the percentage of benign peers suspected to be malicious, that are present in each partial view.

In the evaluation, we want to measure: (a) how much time is required to achieve a stable (possibly low) pollution in the caches, (b) how the PS overlay becomes organized (e.g., are the false positive peers isolated?), (c) the performance when the attackers target a larger set of $\overline{\text{FW}}$ peers and finally (d) the performance in dynamic scenario.

The basic setup involves 1.000 peers in $\overline{\text{FW}}$ and 10.000 peers in FW, playing the mosquito attack. If not state otherwise, the $\overline{\text{FW}}$ peers targeted by the attackers is a set of $k = c = 20$, where c is the cache size of the underlying PS implementation adopted (NEWSCAST). Algorithms are identified as follows: PS is the standard peer sampling algorithm; SPS(sg) is the prestige-based algorithm described in [10], where s gossip exchange are performed at each cycle; SPS($sg + 1$) is the same prestige-based algorithm, where the additional check for false positive is performed.

Figure 2 shows the average pollution proportion in partial views. In the absence of defense, the peers becomes quickly polluted with (false positives) malicious peers. Essentially, malicious peers can easily reorganize the network, promoting the targeted $\overline{\text{FW}}$ peers as network hubs. This puts a tremendous stress on these peers that and it may lead to a crash with more probability. The SPS can prevent the diffusion of the targeted $\overline{\text{FW}}$ peers, as the average pollution level is extremely low. However, this means that the vast

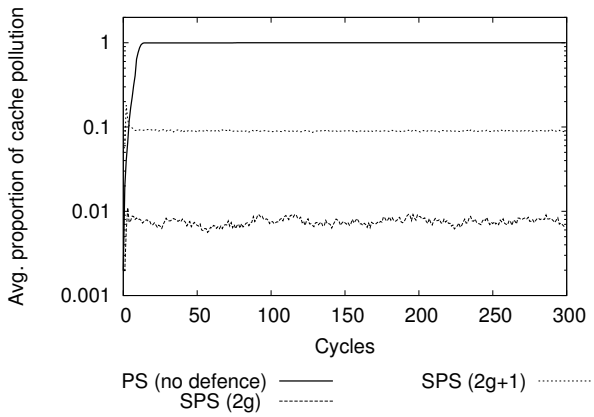

Figure 2. The average pollution level over time.

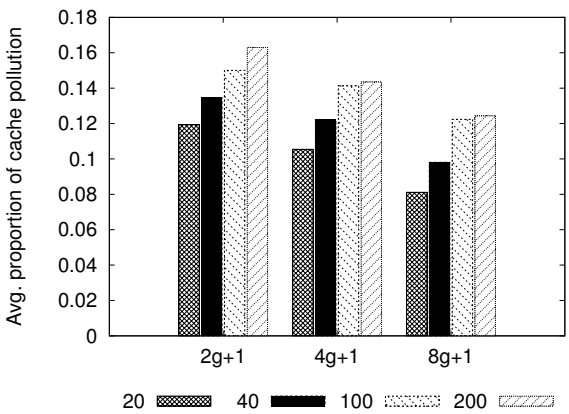

Figure 3. The average pollution level at the end of the simulation. Each bar represents a distinct size of the set of targeted peers. Each group of bars shows a distinct number of gossip exchanges.

majority of $\overline{\text{FW}}$ peers believe that the set of targeted peers is malicious, which is obviously not true. The false positives inducted by the mosquito attack, have an impact of the overlay structure. While we always found that the network remains connected (weakly) during our experiments, there is no guarantee that the pool of targeted peers can never disconnect (especially for larger networks). In addition, the $\overline{\text{FW}}$ peers tends to have no links to the targeted peers, while targeted peers maintain a few links to the rest of the network, but tends to be located in a clique.

Considering the huge amount of resources put into the field, the result of the mosquito attack facing the SPS is not so serious for the overlay in general. In fact, the overlay is still connected and shows acceptable values in terms of clustering coefficient ($\sim$0.22) and average path length ($\sim$2). However, the connectivity and the quality of the sampling of the service is for sure compromised for the targeted peers.

When the false positives check is enabled, the SPS shows a pollution level of $\sim$ 9-10%. This value is higher than before, but we have to remember that these measurements refers to the presence of false positives. In other words, now the SPS recognize false positives[1] and allows these $\overline{\text{FW}}$ peers to moderately increment their presence in the overlay structure. This mechanism makes the overlay structure almost uniform and the targeted peers are not relegated to the margins.

The average cost per cycle for each peer can be expressed in terms of total exchanges messages (using NEWSCAST implementation) as: $2 \cdot size + 1$.

In Figure 3 we show the pollution level at the end of the simulation. Each bar represents a distinct size of the set of targeted peers (i.e., 20, 40, 100 and 200 peers), while distinct number of gossips adopted by the SPS (i.e., 2g+1, 4g+1 and 8g+1) are shown on the x-axis.

The pollution or the presence of false positives in partial views is limited to a minimum and manageable level in every case -i.e., such levels do not pose any problem to the organization of the pseudo-random graph. As the number of gossips increase, their presence decreases almost linearly. Only when the set of target peers is 2% of the ($\overline{\text{FW}}$) overlay size, the proportion is sub-linear in the cases of 4g+1 and 8g+1 gossips. This happens because the target set size becomes too large compared to the partial views that are actually exchanged at every gossip ($k = c = 20$). The spreading of the target peer IDs becomes inefficient and becomes harder and harder to discredit such a large set.

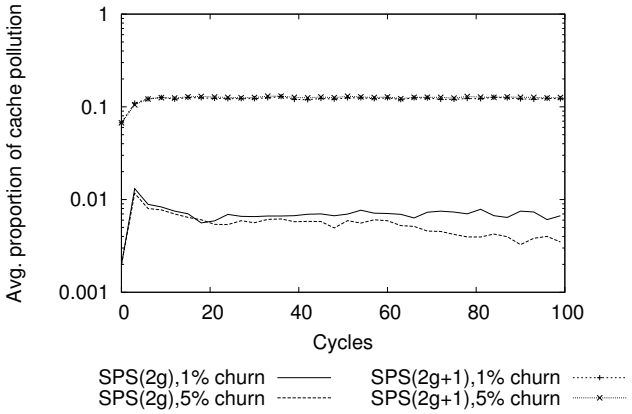

Figure 4. The average pollution level in a churn scenario over time. Churn rates are: 1% and 5%, 2 gossips and 2+1 are adopted.

[1] Because $\overline{\text{FW}}$ peers ranked as malicious do not deliberately foster the diffusion of their IDs, they can prove their non affiliation with the attackers.

Although we considered that $\overline{\text{FW}}$ peers are high-reliable hosts, we have also to consider a physiologic churn process caused, for example, by crashes, (voluntary) leave and joins. In Figure 4, the SPS respectively with (2 gossips+1) and without (2 gossips) the false positives check are compared. The standard SPS adopted involves 10000 peers, while the check enabled involves 1000 $\overline{\text{FW}}$ and 10000 FW peers. While the standard version tends to decrease (to almost zero) over time the proportion of the target peers in the partial views, the check enabled version allows a low, but constant presence of the target peers; this allows target peers not to be isolated. The difference between 1% or 5% or churning is basically negligible.

5 Related work

Poisoning attacks are closely related to both the hub and the mosquito attacks. In particular, the index poisoning attack [11] 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 non-existing or inaccessible files. It 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. In [12], the authors combine the previous index poisoning attack with poisoning routing tables in DHT file-sharing systems. This combination leads to an effective DoS attack. In this case, a selected victim host is referenced by many other (poisoned) overlay participants, effectively significantly increasing the probability that a message will be routed to the victim.

In [1], the authors presents a sampling membership algorithm where each local view converges to a uniform sample and can resists to the failure of a linear portion of the nodes. This approach defines and uses its own sampling algorithm, while we focus on securing an already existent service.

In [15], the authors introduces a fully decentralized approach for securing synthetic coordinate systems. They adopt a sort of social-like, vote-based approach in which each coordinate tuple must be checked by a (small) set of other nodes. For each node producing a coordinate tuple, the set of nodes that have to check and eventually approve that tuple is given by a hash function based on each node's unique identifier. The system is very resilient to attacks targeting instabilities and inaccuracies to the underlying coordinate system. However, this approach requires the presence of a DHT facility that adds complexity and may become an extra source of issues (e.g., DHT attacks).

Social network principles (e.g., reciprocity and structural holes) are also adopted in JetStream [14] to optimize and build robust gossip systems. The basic idea is to make a predictable neighbor selection when gossiping to avoid unpredictable, excessive message overhead. In addition, the traditional scalability and reliability of gossip are maintained.

6 Conclusions

This paper presents a particular instance of the hub attack, called mosquito attack. We evaluated the behavior of an existing SPS service under such scenario, discovering its inadequacy, and developed a modified version of it capable to tolerate such attack. The proposed variant can successfully detect which peers are targeted by malicious peers, while maintaining a reasonable random structure of the overlay. This result is obtained through simple SNA methods to the topology built by the PS service.

References

[1] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer. Brahms: Byzantine resilient random membership sampling. In *ACM PODC*, 2008.

[2] W. de Nooy, A. Mrvar, and V. Batagelj. *Exploratory Social Network Analysis with Pajek*. Cambridge University Press, Cambridge, 2005.

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

[4] M. Jelasity, A. Montresor, and O. Babaoglu. The Bootstrapping Service. In *ICDCS*, Lisboa, Portugal, 2006. IEEE.

[5] M. Jelasity, A. Montresor, and O. Babaoglu. T-Man: Gossip-based fast overlay topology construction. 2009. To appear.

[6] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based Peer Sampling. *ACM Trans. Comput. Syst.*, 25(3):8, 2007.

[7] 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, 2006.

[8] G. P. Jesi, D. Hales, and M. van Steen. Identifying Malicious Peers Before it's Too Late: A Decentralized Secure Peer Sampling Service. In *SASO*, Boston, MA, 2007.

[9] G. P. Jesi, A. Montresor, and O. Babaoglu. Proximity-aware Superpeer Overlay Topologies. In *SelfMan*, volume 3996 of *LNCS*, Dublin, Ireland, 2006. Springer.

[10] G. P. Jesi, S. K. Nair, M. van Steen, and E. Mollona. Prestige-based Peer Sampling Service: Interdisciplinary Approach to Secure Gossip. In *SAC*, Honolulu, US, Mar 2009.

[11] J. Liang, N. Naoumov, and K. Ross. The Index Poisoning Attack in P2P File Sharing Systems. In *INFOCOM*, Barcelona, Spain, 2006.

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

[13] S. J. Nielson, S. Crosby, and D. S. Wallach. A taxonomy of rational attacks. In *IPTPS*, volume 3640 of *LNCS*, pages 36–46. Springer, 2005.

[14] J. A. Patel, I. Gupta, and N. Contractor. JetStream: Achieving Predictable Gossip Dissemination by Leveraging Social Network Principles. In *NCA*, Cambridge, MA, 2006.

[15] M. Sherr, B. T. Loo, and M. Blaze. Veracity: A fully decentralized service for securing network coordinate systems. In *IPTPS*, 2008.