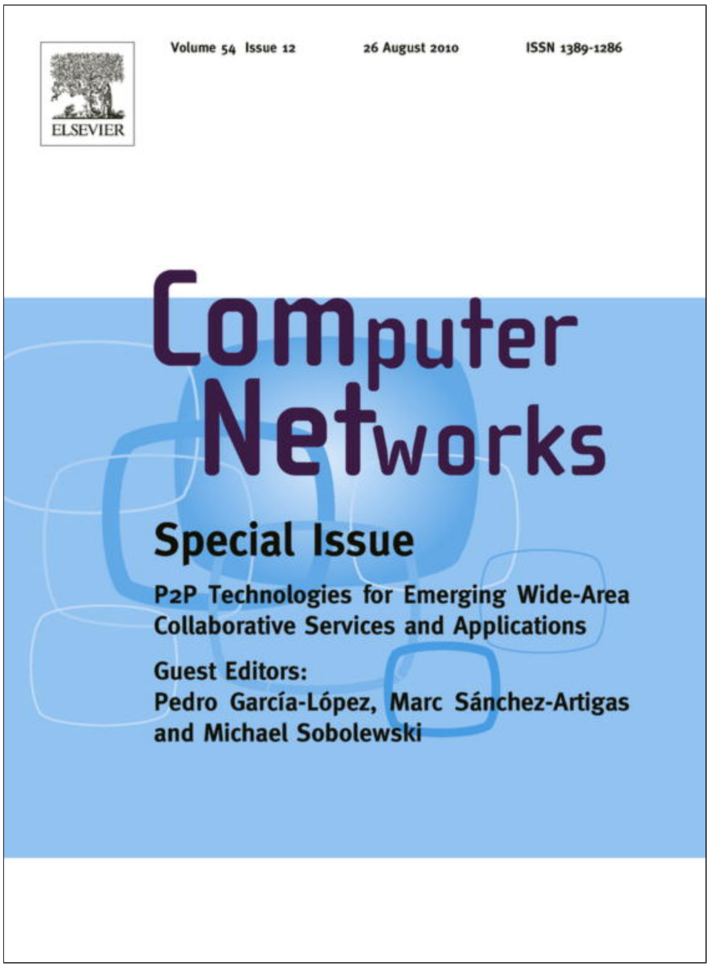


Volume 54 Issue 12 26 August 2010 ISSN 1389-1286

ELSEVIER

Computer Networks

Special Issue

P2P Technologies for Emerging Wide-Area Collaborative Services and Applications

Guest Editors:
Pedro García-López, Marc Sánchez-Artigas and Michael Sobolewski

Computer Networks 54 (2010) 2086–2098

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Secure peer sampling

Gian Paolo Jesi [a,*], Alberto Montresor [a], Maarten van Steen [b]

[a] Dip. di Ingegneria e Scienza dell'Informazione, University of Trento, Italy
[b] Dept. of Computer Science, VU University, Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Available online 9 June 2010

Keywords:
P2P
Peer sampling
Overlay
Security
Gossip

ABSTRACT

Gossiping has been identified as a useful building block for the development of large-scale, decentralized collaborative systems. With gossiping, individual nodes periodically interact with random partners, exchanging information about their local state; yet, they may globally provide several useful services, such as information diffusion, topology management, monitoring, load-balancing, etc. One fundamental building block for developing gossip protocols is *peer sampling*, which provides nodes with the ability to sample the entire population of nodes in order to randomly select a gossip partner. In existing implementations, however, one fundamental aspect is neglected: security. Byzantine nodes may subvert the peer sampling service and bias the random selection process, for example, by increasing the probability that a fellow malicious node is selected instead of a random one. The contribution of this paper is an extension to existing peer sampling protocols with a detection mechanism that identifies and blacklists nodes that are suspected of behaving maliciously. An extensive experimental evaluation shows that our extension is efficient in dealing with a large number of malicious nodes.

1. Introduction

Would it make sense to implement large-scale collaborative applications like Facebook or Twitter in a completely decentralized way? From a social point of view, the answer is definitely affirmative: the privacy concerns raised by a company whose end-user agreement can be summarized as “all your data belongs to us” are astonishing. From a technological point of view, however, the answer is not so simple: the enormous scale (hundreds of millions of users) and dynamism of such systems pose enormous challenges to developers.

Gossip protocols have proven to be effective in dealing with these challenges, going beyond the basic dissemination services for which they have been originally designed [1] and implementing sophisticated services like topology management, aggregation and monitoring, load-balancing, semantic clustering, etc. [2–5].

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 nodes with continuously up-to-date *samples* selected uniformly at random from the global node population [6]. Informally, gossip-based PS services work as follows. Each node stores a collection of *node descriptors*, called the (*partial*) *view*. Execution is divided in periodic *cycles* during which each node p selects a node q from its view and initiates a push–pull communication exchange with it: p sends a subset of its own descriptors to q, plus a fresh descriptor of itself, and q replies in the same way. Node p updates its view based on the message received from q, and symmetrically q does the same. Old descriptors are progressively replaced by new ones; this mechanism keeps views continuously up-to-date with respect to node joins and leaves.

[*] Corresponding author. Address: Dip. di Ingegneria e Scienza dell'Informazione, University of Trento, via Sommarive 14, 37128 Trento, Italy.

E-mail addresses: jesi@disi.unitn.it (G.P. Jesi), montresor@disi.unitn.it (A. Montresor), steen@cs.vu.nl (M. van Steen).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.03.020

Several implementations exist [7,8], that can be distinguished based on how they select the exchange partner (e.g., completely at random or based on a timestamp); how many and which descriptors are exchanged (e.g., all of them or a small subset); how the update operation is performed (e.g., by discarding old descriptors or by swapping).

By interpreting descriptors as edges, the PS service can also be seen as a mechanism to maintain a topology among nodes. In all existing implementations, the resulting topologies present characteristics similar to those of random graphs: small diameter and extreme robustness against partitions. They are even self-repairing in the sense that old descriptors tend to be discarded and information about new nodes is naturally spread through gossip. These properties make them the right platform for gossip protocol development.

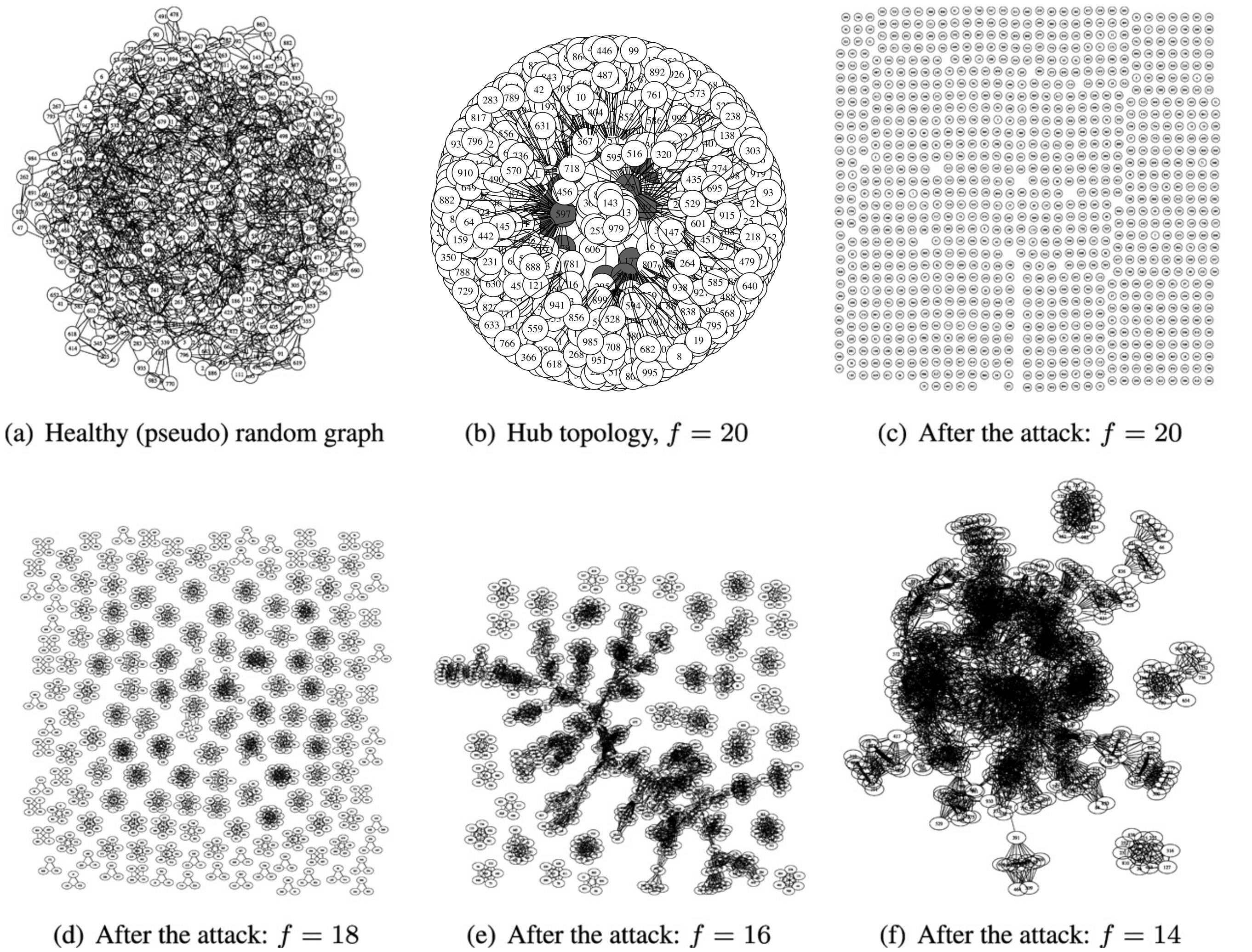
An important issue of modern PS services is their potential exploitation by *malicious* nodes (or *attackers* for short). The characteristics of the topology depend on the way descriptors are exchanged; if some of the nodes do not behave according to the protocol, the sample process can be biased toward a specific group of nodes instead of being random; the resulting topology can fail to show the desired properties.

The most important kind of attack that can be pursued against gossip-based PS services is the *hub attack*, where attackers attempt to gain a leading position in the topology (they attempt to become *hubs*), to later exploit their lead to cause havoc to the system, such as performing a DoS attack that leaves the topology in a disconnected state.

For a concrete example of such attack, look at Fig. 1. When the system runs correctly, a random topology is formed, as illustrated in Fig. 1a.

Now assume that a small number *f* of colluding attackers join the system. Note that *f* can be as small as the partial view size, which is around 20–30 for most PS services.

Instead of running the regular protocol, the attackers completely ignore the descriptors they receive and keep sending the descriptors of the malicious group members. The partial views in the entire system are progressively polluted by the attackers' descriptors, which keep being generated by malicious nodes and propagated by correct ones. As the percentage of malicious-node descriptors in a partial view grows, the probability of contacting malicious nodes grows proportionally, facilitating their job:



(a) Healthy (pseudo) random graph

(b) Hub topology, $f = 20$

(c) After the attack: $f = 20$

(d) After the attack: $f = 18$

(e) After the attack: $f = 16$

(f) After the attack: $f = 14$

Fig. 1. Overlay topology before (a), during (b) and after (c–f) the attack. The healthy random graph (a) is mutated in a hub-based overlay (b). The graphs (c), (d), (e) and (f) show what happens if the hubs leave the system (with *f* = {20, 18, 16, 14} colluding attackers). Only 3 links per node are displayed for clarity. Network size is 1000 nodes.

they can just "sit down" and wait for requests, which are always replied with malicious-node descriptors. Eventually, the situation looks like Fig. 1b: all the partial views of correct nodes point to malicious ones, which then have become hubs.

Once the hub topology is completed, a massive DoS attack can be struck by simply leaving the overlay. The result is shown in Fig. 1c: all correct nodes are completely disconnected with no hope for recovery.

Due to drawing limitations, Fig. 1 has been generated in a small system (1000 nodes). Clearly the same issue applies to larger ones. The number of attackers, $f = 20$, has been chosen equal to the partial view size. However, even a smaller number of malicious nodes is sufficient to cause havoc to the system: the second line of Fig. 1 shows the final state when 18, 16, 14 attackers are involved, with several partitions showing around.

Unfortunately, there is no way to detect if a particular message comes from an attacker, as the message format is very simple: just a list of descriptors. So, the only way to identify and ban "bad guys" is to perform a structural analysis of the network, and react opportunely when a node gains a unmotivated prominent role.

This work extends existing PS protocols in this sense. We have designed and experimentally tested a prestige-based *secure peer sampling service* (SPS) based on heuristics inspired by social network analysis (SNA) techniques [9] used to measure the structural prestige of nodes in a network. We show how techniques from SNA can be adopted in gossip-based distributed systems in order to solve the identified security issues. This approach is in line with works that exploit social networks towards building robust systems [10].

The rest of the paper is organized as follows. Section 2 describes our scenario, while the attack model is explained in 3. In Section 4 we introduce the SPS algorithm. Experimental results are presented in Section 5. Finally, Sections 6 and 7 survey related work and conclude the paper.

2. Background

2.1. System model

We consider a network consisting of a large collection of *nodes* that communicate through message exchanges. Each node is uniquely identified by an identifier *id*, required to communicate with the node. We consider the simplest form of *id*: the pair (*ip address*, *port*). The network is highly dynamic; new nodes may join at any time, and existing nodes may voluntarily leave. Nodes may be *correct*, in which case they obey the given protocol but may crash unexpectedly, or *maliciously*, if they behave arbitrarily because of a bug, misconfiguration, or ill-will. Malicious nodes, sometimes called *attackers* in this paper, may know each other and collude together.

Communication may incur unpredictable delays and is subject to failures. Single messages may be lost, links between pairs of nodes may break; but we assume that the integrity of messages is not at risk, and thanks to cryptographic techniques (see Section 2.3), malicious nodes cannot impersonate correct ones.

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 node, a *partial view* of the network, i.e., a short list of logical links to other nodes. As nodes 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.

Nodes and their logical links form a dynamic *overlay topology*. An important requirement is that such a topology should remain connected in spite of failures (even catastrophic ones), otherwise separate overlay partitions would not be able to sample each other. If local views contain random nodes, the resulting topology is a random graph, which has proven to be extremely robust and capable of maintaining connectivity even after the crash of 70% of the nodes [6]. This is the reason why the peer sampling focuses only on random overlays; therefore, we do not consider other kinds of networks such as DHT or super-peer overlays.

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 view contains c descriptors, i.e., pairs (id, ts), where id is the node identifier and ts is the timestamp. Periodically, each node p randomly selects a gossip partner q from its local sample, and sends its partial view to q, plus a fresh descriptor of itself. Node 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. This mechanism is depicted in Fig. 2.

The continuous injection of new descriptors gradually removes old descriptors from the network, allowing the protocol to "repair" the overlay topology by forgetting crashed nodes. No explicit "leave" message is required; it is sufficient to stop executing the protocol, thus suspending the refreshment of its own descriptor.

2.3. Cryptography

We assume that each node has exactly one private/public key pair bound to its permanent identifier *id*, assigned by a certification authority. This has two purposes: to limit the number of distinct identities that can be assumed by a node, thus avoiding the possibility of a Sybil attack [11], and to allow nodes to sign their messages and their descriptors, to avoid the possibility of impersonating other nodes and modifying descriptors that are in transit. For this reason, we extend the descriptor of a node p with its public key and its signed hash; we denote it as: $\langle id, ts, PK \rangle_p$. We denote a message m signed by a node p holding c descriptors as:

$$\begin{aligned}\langle m \rangle_p = \langle &\langle id_1, ts_1, PK_1 \rangle_1, \\ &\langle id_2, ts_2, PK_2 \rangle_2, \\ &\ldots \\ &\langle id_c, ts_c, PK_c \rangle_c \rangle_p.\end{aligned}$$

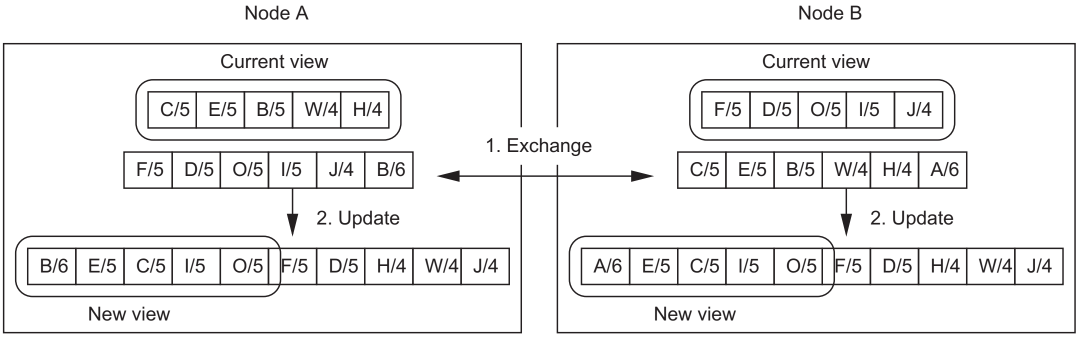


Fig. 2. A Newscast gossip exchange between node A and B. The size of partial view is $c = 5$. A descriptor is represented by a capital letter along with its timestamp. The exchange time is cycle 6.

We assume that cryptographic primitives – such as digital signatures and one-way hashes – cannot be subverted.

3. Attack model

Our attack model is generic and independent of the specific PS implementation adopted. The main objective of an attacker is to silently subvert the random overlay topology and cause the formation of *hubs*, i.e., nodes with a large number of incoming links.

Being a hub represents a leading position from a structural point of view. As soon as the attackers become hubs, they can exploit this position according to their particular aim. For example, they could manage the information flux or they could just disappear, achieving a massive DoS attacks as they leave the overlay topology in a partitioned state. Any service that relies on the PS can be severely affected.

To reach this goal, attackers must *pollute* the views of other nodes by installing as many descriptors of attackers as possible. If a view is completely polluted (i.e., all the descriptors point to malicious nodes), the node is defeated: it has no chance to recover to a regular situation, apart from rebooting.

Our attack model expects attackers to operate *rationally* and not allow themselves to be trivially exposed as the source of the infection. Therefore, messages sent by attackers should be indistinguishable from regular messages. This requirement is quite easy to achieve in PS services, as the only mandatory constraints are that messages should not contain more than c distinct descriptors; no other check is performed. Surprisingly, weak integrity constraints like this can be found, for example, in real-world file-sharing applications [12,13], where nodes do not verify the validity of received item advertisements.

While the goal and the principal methods of attack are fairly generic, actual attacks may adopt features that are specific to the underlying protocol implementation. A reasonable taxonomy of possible malicious actions that may be carried by an attacker are summarized as follows:

1. *Discard*: an attacker may decide to discard specific node descriptors from its local view and its messages, irrespectful of their fresh timestamp. The goal could be to “eclipse” a node from the rest of the system, removing all the information about it from partial views.
2. *Replay*: attackers can avoid discarding specific node descriptors, even when their timestamps identify them as (really) old. The goal could be to transform a unaware node into an hub, to have it detected and blacklisted.
3. *Corrupt*: attackers can corrupt in-transit descriptors, to maliciously trigger an anomalous behavior in the specific PS implementation. Apart from the node address, the only information that could be corrupted is the timestamp, for example, to keep an old descriptor alive.
4. *Forge*: attackers can populate the messages sent to benign nodes with descriptors which are explicitly forged to pollute the network.

This set of actions can be performed against existing correct or malicious nodes, or even against nonexisting nodes that are not member of the system.

The first two actions, discard and replay, do not pose any serious threat even to a nonsecured PS service; in fact, the gossip paradigm provides enough redundancy to avoid that a given node becomes completely eclipsed from other nodes; and even if an attacker tries hard to push an old, unmodified descriptor to other nodes, they will unilaterally discard it based on its timestamp.

In case of corruption or forging of descriptors pointing to correct nodes, the solution is simple: descriptors should be signed with the private key of the node contained in the descriptor. In this way, no attacker can impersonate any existing correct node. The same comment applies to the action of forging descriptors for nonexisting nodes.

The only possible actions left is the replaying, corruption and forging of descriptors of colluding attackers. In general, the attack algorithm works as follows. Attackers do not pose any restriction on the size of their partial view c and let them grow to potentially include all the correct nodes of the network. The well-known attacker set is not included in the partial view. The attackers run the standard PS algorithm with the following exceptions:

- the message sent to p is populated with malicious descriptors based on a specific attack strategy,
- the timestamps of malicious descriptors are manipulated in order to postpone their droppings as late as possible.

Several strategies are then possible, based on the number of attackers $k \leqslant f$ that are inserted in the messages sent to gossip partners, and how they are selected from the malicious population (e.g., uniformly or normally at random).

Less aggressive strategies (with small *f*) are more stealthy and difficult to detect, but also less effective; more aggressive strategies (with *f* approaching or equal to *c*, the view size) are more efficient in polluting the system, but also easier to detect.

4. Prestige-based SPS

The secure peer sampling service (SPS) is designed to be transparent to applications using the PS service: the API is not modified, and the secure portion of the service runs as an extension. In fact, the basic structure of the SPS service is to (i) play the nonsecure PS implementation as usual, and (ii) monitor the overlay and react to structural changes when required.

The goal of the SPS module running at each node is to identify and *blacklist* potentially malicious nodes, removing them from the local partial view and excluding them from future interactions. The process of identification takes inspiration from *social network analysis* theory; in particular, we consider the notion of *prestige* in directed graphs, where the "most popular" nodes are considered prestigious [9]. We adopt a simple technique where the structural prestige of a node is represented by its indegree. Intuitively, since the expected topology should be uniformly random, detecting a node showing a large indegree value could mean that node is a network hub – and so it is potentially malicious.

As the infection of a pollution attack can spread exponentially fast (a property inherent to gossip-based dissemination), the main concern for an SPS service is to quickly build a suitable knowledge base about the prestige of other nodes present in the system. Information about other nodes is collected through gossip exchanges with random partners; the collected information is not shared with other nodes [12] to avoid further issues, such as the corruption of the exchanged knowledge.

In order to build this knowledge base, the SPS service does not limit itself to one gossip exchange per cycle, as the PS service does. Instead, multiple *explorative gossip exchanges* are performed, the goal being (i) to identify structural hubs, i.e., nodes that are over-represented in the partial views of other nodes, and (ii) to collect "safe" descriptors that may become useful if and when the local partial view becomes polluted. As there is no way for a (malicious) node to detect if a gossip partner is executing the regular protocol or an explorative exchange, this behavior generates a *dilemma* that could severely limit the attackers' power.

4.1. Algorithm structure and notation

The execution of the SPS algorithm is organized in *cycles* of length δ: periodically, each node initiates some gossip exchanges with random partners and waits for their reply, following a push–pull style [1]. The code is organized around these exchanges, with the *active thread* that periodically sends messages and performs clean-up operations, and the *passive thread* that waits for incoming communications and appropriately updates local data structures, replying to the original sender when needed. The pseudo-code of the active and passive threads is contained in Figs. 3 and 4, respectively, while Fig. 5 describes *updateStatistics*( ), a support function for the passive thread.

Each node maintains three tables whose task is to associate keys (node identifiers) to values (potentially, complex tuples):

- PTABLE is the *prestige table*, which associates each node *q* with prestige information in the form of a tuple $\langle ts, hits, ttl \rangle$, where *ts* is the most recent timestamp known for *q*, *hits* counts the number of times *q* has been included in PS messages received from other nodes, and *ttl* is a time-to-live value expressing the time validity of the tuple.
- WLIST is a whitelist data structure, which contains nodes that are believed to be correct. In this table, each node is associated with just the latest timestamp known for it; prestige information is not needed in this case. For the same reason, nodes in WLIST are removed from PTABLE.

```
repeat periodically every $\delta$ time units                 % Code executed at each cycle of length $\delta$
    $done \leftarrow$ false
    $R \leftarrow sample(\text{VIEW}.keys(), size)$                  % Explorative exchanges
    foreach $q \in R$ do
        if $q \notin blacklist(\text{PTABLE})$ then
            send $\langle \text{REQUEST}, \text{VIEW} \cup \langle myID, now(), PK \rangle_{myID} \rangle_{myID}$ to $q$
    $check \leftarrow sample(blacklist(\text{PTABLE}), 1)$           % False positive check
    send $\langle \text{REQUEST}, \text{VIEW} \cup \langle myID, now(), PK \rangle_{myID} \rangle_{myID}$ to $check$
    foreach $q \in \text{PTABLE}.keys()$ do                       % Clean the PTABLE and add nodes to the whitelist
        $tuple = \text{PTABLE}.get(q)$
        $tuple.ttl \leftarrow tuple.ttl - 1$
        if $tuple.ttl = 0$ then
            $\text{PTABLE}.remove(q)$
            $\text{WLIST}.put(q, tuple.ts)$
```

Fig. 3. Prestige-based SPS algorithm – the active thread.

```
on receive $\langle type, msg\rangle$ from $q$ do
    if $type = \text{REQUEST}$ then                                   % We are not the initiator; we must reply
        send $\langle \text{REPLY}, \text{VIEW} \cup \langle p, now(), PK\rangle_{myID}\rangle_{myID}$ to $q$
        if $q \notin blacklist(\text{PTABLE})$ then
            $\text{VIEW} \leftarrow merge(\text{VIEW}, msg)$
    else
        if $q \in R$ and $q \notin blacklist(\text{PTABLE})$ and not $done$ and $toss(\frac{1}{size})$ then    % This is a reply to
            $done \leftarrow$ true                                    % an explorative exchange
            $\text{VIEW} \leftarrow merge(\text{VIEW}, msg)$
        else
            $updateStatistics(msg)$;
        if $q = check$ then                                           % This is a reply to a false positive check
            if $msg.keys() \cap blacklist(\text{PTABLE}) = \emptyset$ then
                $tuple \leftarrow \text{PTABLE}.get(q)$
                $\text{PTABLE}.remove(q)$
                $\text{WLIST}.put(q, tuple.ts)$
```

Fig. 4. Prestige-based SPS algorithm – the passive thread.

```
method $updateStatistics(msg)$
    foreach $(q, ts) \in msg$ do                                      % Updates statistics based on $msg$
        $tuple \leftarrow \text{PTABLE}.get(q)$
        if $tuple =$ null then
            $\text{PTABLE}.put(q, \langle ts, 1, ttl_0\rangle)$
        else
            $\text{PTABLE}.put(q, \langle \max(ts, tuple.ts), tuple.hits + 1, tuple.ttl + 1\rangle)$
    $hits_\mu \leftarrow \frac{1}{|\text{PTABLE}|} \sum_{(q,tuple)\in\text{PTABLE}} tuple.hits$
    $hits_\sigma \leftarrow \sqrt{\frac{\sum_{(q,tuple)\in\text{PTABLE}} (tuple.hits - hits_\mu)^2}{|\text{PTABLE}|}}$
    foreach $q \in blacklist(\text{PTABLE}) \cap \text{WLIST}.keys()$ do    % Remove new suspected nodes from WLIST
        $\text{WLIST}.remove(q)$
    foreach $\langle id, ts\rangle \in \text{VIEW}$ do                      % Remove new suspected nodes from VIEW
        if $id \in blacklist(\text{PTABLE})$ then
            if $\exists\, q \in \text{WLIST}.keys() - \text{VIEW}.keys()$ then
                $\text{VIEW}.remove(id)$
                $ts \leftarrow \text{WLIST}.get(q)$
                $\text{VIEW}.put(q, ts)$
```

Fig. 5. Prestige-based SPS algorithm – support function.

- VIEW is the partial view managed by the selected PS protocol (NEWSCAST); as WLIST, it associates nodes with timestamps; note that VIEW has a fixed size *c*.

These data structures are managed by methods *put*(*id*,*tuple*) which associates the tuple *tuple* with the node identifier *id*; *get*(*id*) which returns the tuple associated with *id*, or **null** if none is present; *remove*(*id*) which removes existing associations of identifier *id*. Finally, method *keys*( ) returns the collection of the node identifiers that are currently associated to tuples in the data structure. In the pseudo-code, tuples are conveniently composed using the ⟨⟩ notation.

A small collection of "predefined" functions are used in our algorithm without showing their pseudo-code. Functions *now*( ) and *merge*( ) represent basic actions of a standard PS. The former returns an up-to-date timestamp, used to create fresh descriptors of itself; while the latter fuses together a local partial view and a standard PS message, keeping the *c* freshest descriptors. Function *sample*(*S*,*k*) returns *k* random objects from the set *S*. Function *toss*(*p*) tosses a random coin, returning the value "true" with probability *p*.

Predefined function *blacklist*(PTABLE) returns the subset of identifiers contained in PTABLE that are currently suspected to be faulty. A precise definition of its behavior is postponed to Section 4.3.

4.2. The dilemma mechanism

As described above, each node periodically initiates multiple explorative gossip exchanges (Fig. 3, lines 2–6): a set of *size* random partners are selected from the partial view by the *sample*(*size*) function. A standard PS message is sent to all of them (unless they are included in *blacklist*(PTABLE), i.e., are suspected to be malicious). These messages are tagged with REQUEST, meaning that they are the first part of a request–reply interaction.

The passive thread handles such messages (Fig. 4). If the received message is tagged with REQUEST (Fig. 4, lines 2–5), the node immediately replies to it (tagging the message with REPLY); furthermore, if the node is not blacklisted, the message is treated as a normal PS message and the partial view is updated through function *merge*.

If the received message is tagged with REPLY (Fig. 4, lines 7–16), there are two cases: either the message is a reply to an explorative gossip exchange (lines 7–11), or it is a reply to a false positive check (lines 12–16); the latter is described in Section 4.4.

Regardless of the number *size* partners selected by *sample*( ), the PS state update is executed at most once; this is implemented by the "if" condition at line 7 of Fig. 4, with the help of the variable *done* (which can be set only once per cycle). Function *toss*(1/*size*) is used here to select a random node among those which reply. If the "if" condition is not satisfied, function *updateStatistics* is called.

This mechanism produces the dilemma discussed above: a malicious node *p* sending a message to a partner *q* (either in reply to a REQUEST message received from *q*, or when initiating an exchange with *q*) never knows how *q* is going to use the information contained in the message. The dilemma is the following: if *p* includes malicious descriptors in the message, it is possible that they will not actually be inserted into the partial view, but instead used to collect statistical information about the prestige of nodes. In fact, the more they pollute, the larger is the probability of appearing as "suspect" in terms of prestige. The other possible choice is to run the protocol correctly, but in this case no pollution will be performed.

4.3. Prestige mechanism

The information collected during explorative gossip exchanges is used to build the knowledge base required to detect, with high accuracy, attackers and to eventually repair the partial view when it becomes polluted by the presence of malicious descriptors.

This functionality is enclosed in function *updateStatistics*(*msg*), described in Fig. 5. Here, PTABLE is updated based on the list of descriptors contained in *msg*. For each node *q* contained in *msg*, we first check whether the node is already included in PTABLE or not; in the former case, the existing tuple is updated, while in the latter a new one is created (lines 3–7). An update is performed as follows: the timestamp is substituted with the most recent one, while both *hits* and *ttl* are incremented by 1. A new tuple is created by taking the timestamp included in the message, setting *hits* to 1 and setting an initial *ttl*.

If an attacker tends to acquire a network-centric position, its prestige (*hits* value) is likely to dominate over the other entries and its value will be much larger than the average prestige of nodes, which is stored in variable $hits_\mu$. In order to discriminate how far a node's *hits* value can be from the current average, we adopted the standard deviation of the *hits* value, which is stored in variable $hits_\sigma$. Function *blacklist*(PTABLE) uses $hits_\mu + hits_\sigma$ as a threshold to distinguish between potential attackers and correct nodes: if $hits \geqslant hits_\mu + hits_\sigma$ at node *p*, the node *p* is included in the set returned by *blacklist*:

$$q \in blacklist(ptable) \iff p \in ptable.keys() \wedge hits \geqslant hits_\mu + hits_\sigma,$$

$hits_\sigma$ is required to improve the quality of the suspicions and to limit the production of false positives. We verified through experiments the effectiveness of the threshold to achieve our goal.

Although we do not pose any size restriction to PTABLE, which could hence eventually grow to the size of the system, this is very unlikely, as the entries are purged according to an aging policy, which decrements the *ttl* field of all nodes in PTABLE by 1 at each cycle (lines 9–14 in the active thread, Fig. 3). When an entry expires (i.e., $ttl = 0$), it is removed from PTABLE and inserted in WLIST, as it can be considered (with high probability) a correct node. Essentially, the WLIST can be considered to be a "possibly trusted" set of peers.

Once the prestige table has been updated, new nodes may become blacklisted. These nodes need to be removed from the whitelist (Fig. 5, lines 10 and 11) and from the partial view (Fig. 5, lines 12–17). In the case of a partial view, to avoid the risk of excessively reducing the size of the partial view, removed descriptors are substituted with descriptors taken from WLIST.

4.4. False positives check

It is possible that correct nodes are falsely blacklisted, at which point the *SPS* reacts by isolating them from the system. We introduce a mechanism that helps to recover from this situation. Essentially, each correct node *p* chooses a blacklisted node *q* from *blacklist*(PTABLE) and makes an explorative PS exchange with *q*. If the received message contains one or more of the other suspected nodes contained in *blacklist*(PTABLE), the suspicion is considered correct. Otherwise, the suspicion is considered incorrect and the node is removed from PTABLE and inserted in WLIST as a new available candidate to substitute a malicious node in the current view when required. 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 mechanism is implemented in the active thread (Fig. 3, lines 7 and 8) and in the passive one (Fig. 4, lines 12–16) and comes at the cost of an extra PS gossip exchange. To limit the message traffic, every node is allowed to perform this check only once per cycle. The required extra gossip is indistinguishable from any other PS exchange; therefore, it is impossible for a malicious node to detect the intent of the node that has started the gossip.

A partial solution for a malicious node would be to randomly mix malicious and regular views over time, but this would greatly reduce the aggressiveness and effectiveness of any hub-like attack. Alternatively, an attacker may profit by the check mechanism in the following manner: at the beginning it plays maliciously for being blacklisted by its neighbors and then starts playing nicely in order to be whitelisted and hence considered a trusted peer. Then the malicious node can restart to pollute. However, this approach will fail for the same reason of the previous one: it will take too long to have any effect on the overlay.

4.5. Cryptography

Each node p cryptographically secures its own descriptors $\langle p, ts_p \rangle$ (containing the identifier and the timestamp) by signing them with its own private key: $\langle p, ts_p \rangle_p$. Partial views and messages exchanged between nodes thus contain a list of descriptors signed by the respective nodes.

Unfortunately, the adoption of cryptography considerably increases the size of views. Considering a PS gossiping views of size $c = 20$, the actual message size is in the range of a few hundreds of bytes:

$$|\langle id, ts \rangle| \cdot 20 = ((4 + 2) + 4) \cdot 20 = 200;$$

here, *id* is represented by a pair (IP address, port), and the timestamp is represented by a 32-bit integer. The SPS, in contrast, can easily reach a few KBytes if we consider default 2048-bit RSA keys. As the small message size is one of the strengths of the PS, a real deployment of the SPS should consider this issue. As a possible optimization, we could let each node collect the identities of the nodes it discovers until the corresponding identity is valid; this behavior would not require to send all credentials for each descriptor in the view. The credentials could be sent along with each descriptor at a much lower rate according to a specific distribution function, reducing the average message size. When a node receives a descriptor for which it does not already know the credentials, it can ask the sender for the corresponding credentials. It is assumed that a correct node knows the credentials for all the descriptors it has encountered, therefore failing in the request would explicitly mean not complying to the rules.

5. Evaluation

We conducted an extensive set of experiments in a simulated environment to evaluate our SPS. We adopted PeerSim [14] as our simulation platform.

The main figure of merit considered in this section is the average level of pollution, measured as the percentage of malicious descriptors that are present in the partial views of the overall system. In particular, we analyze: (a) how much time is required to achieve a stable (possibly low) pollution in the views, (b) how the protocol's basic parameters (e.g., the number of explorative gossip exchanges per cycle, or the initial TTL value) affect its performance, (c) the resulting organization of the PS overlay at the global and local level, and (d) the pollution in a dynamic scenario. Finally (e), we analytically show the communication overhead induced by our solution.

We simulate a network of 10,000 participants running the SPS service on top of Newscast. The local view size is $c = 20$. If not stated otherwise, the hub attack is struck by a set of $f = 20$ malicious nodes; in other words, the set of malicious nodes is as large as the node's view size. Malicious nodes collude in order to share their identity credentials; in such a manner, they can sign forged descriptors on behalf of each other without running the risk of being discovered due to an obvious violation.

We consider three distinct strategies with which the attackers inject the malicious descriptors in their messages: (i) *standard*, where a constant number of malicious descriptors ($f = 20$) are continuously injected, (ii) *random* and (iii) *normal* in which the number of malicious descriptors is chosen at every exchange uniformly in the range $[0 \ldots c]$ or according to a normal distribution having parameters $\mu = 15$ and $\sigma = 2$, respectively.

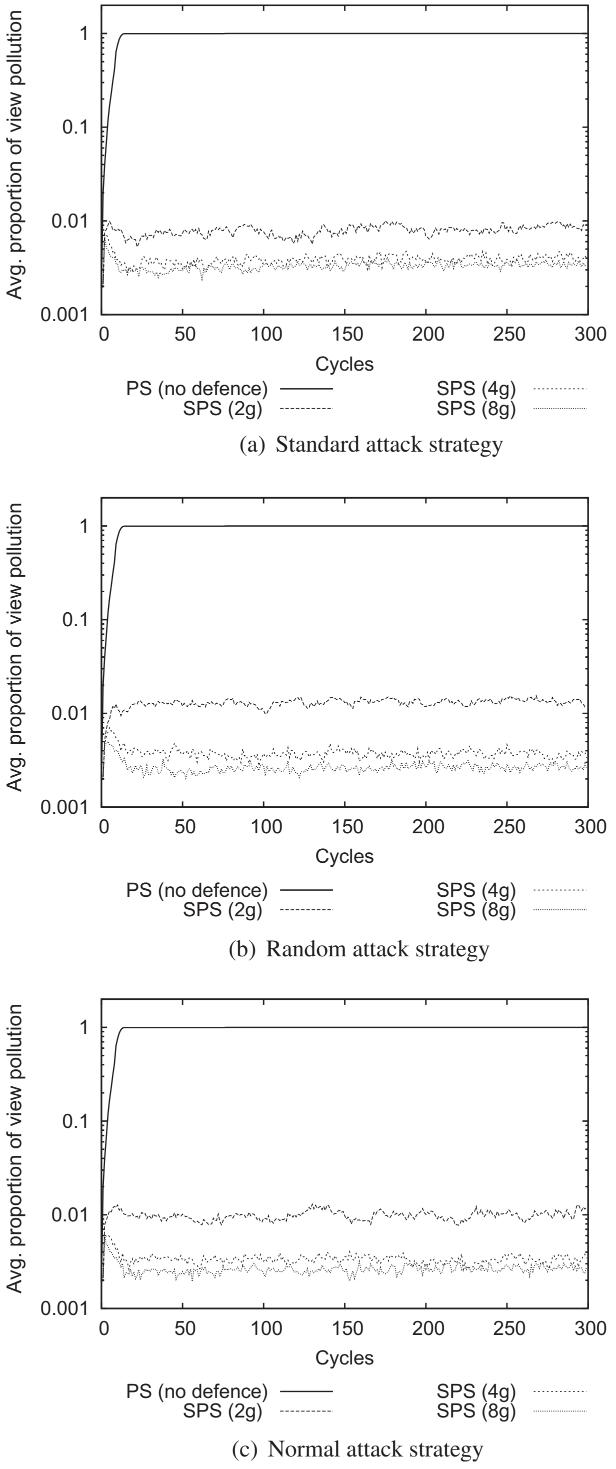
5.1. Pollution

In Fig. 6, we show the average pollution level over time. Each subfigure, from top to bottom, corresponds to the standard, random and normal attack strategy. Each plot represents a distinct number of gossip exchanges performed by the SPS: 1, 2, 4 and 8. We considered a static scenario, in which no node joins or leave the network during the lifespan of the experiment. When a single gossip per cycle is performed, the SPS service is indistinguishable from the ordinary PS service in which no defense mechanism is in place. In fact, about 20 cycles are sufficient to fill all node's views with malicious-node descriptors and achieve an overlay organization equivalent to the one depicted in Fig. 1b.

Starting from the 2-gossip setup, the situation changes dramatically. The average pollution level drops to a negligible 1%. Increasing the number of gossip exchanges to 4 or 8 has a marginal benefit that probably does not justify the extra communication costs.

When the attack strategy is random or normally distributed, the pollution level reaches 3%, higher than the previous strategy, but still low enough to not pose any serious threat to the overlay. Again, increasing the number of gossip exchanges has a marginal benefit. The higher level of pollution achieved by these strategies proves their better efficiency. Essentially, the knowledge base collected by the SPS service is accurate when there is small variance in the attack pattern; smarter patterns are more difficult to be identified and the attackers can be more effective with a less aggressive and more "stealthy" behavior.

However, when more attackers are involved, the efficiency of the random and normal attack strategies decreases. Fig. 7 shows a comparison of the average pollution levels achieved at the end of the simulation according to the proportion of malicious nodes and the number of gossip exchanges (i.e., 2, 4, 8). In particular, we consider four distinct percentages of malicious nodes in the system: 1%, 2.5%, 5% and 10% of the network size and each attack strategy is represented in the corresponding subfigure. We limited the attackers to a maximum of 10% of the population; given that we excluded the possibility of a Sybil attack, we believe that larger proportions of

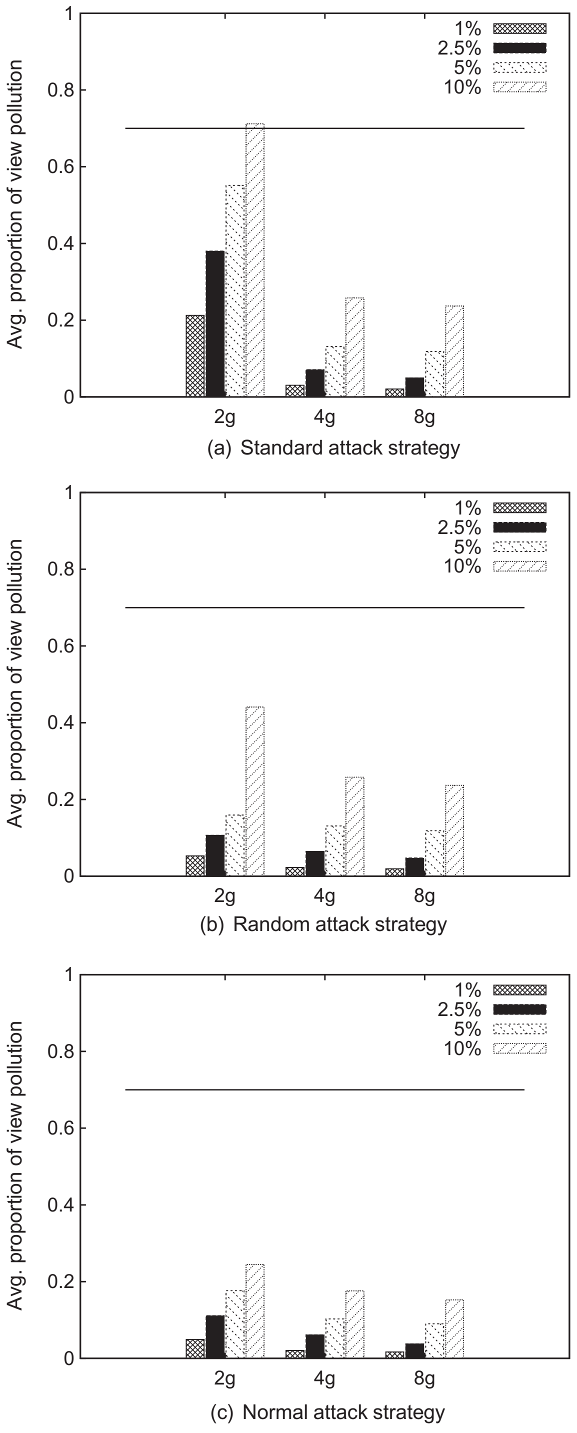


(a) Standard attack strategy

(b) Random attack strategy

(c) Normal attack strategy

Fig. 6. The average pollution level over time. Distinct attack strategies [i.e., (a) standard, (b) random and (c) normal] are represented in each corresponding subfigure. Each plot represents a distinct number of gossip exchanges (i.e., 1, 2, 4, 8, respectively); $f = c = 20$ malicious nodes are present. Network size is 10,000.



(a) Standard attack strategy

(b) Random attack strategy

(c) Normal attack strategy

Fig. 7. Comparison of average pollution levels at the end of the experiment according to the proportion of malicious nodes (i.e., 1%, 2.5%, 5% and 10% of the network size) and value of *size* (i.e., 2, 4, 8). Each subfigure represent a distinct attack strategy. The thick horizontal line represents the maximum tolerable pollution level. Network size is 10,000.

colluding, malicious attackers are unlikely to happen. The thick horizontal line represents the maximum tolerable pollution level over which the overlay runs the risk of being partitioned if and when the malicious nodes leave the network (see Fig. 1c and d).

Only the most aggressive strategy – the standard one – can be a threat for the system when 10% of the network behaves maliciously and two gossip exchanges are adopted. Essentially, when the number of attackers is larger than the view size, promoting the popularity of random sets of malicious nodes is useless as the popularity is uniformly increased over the whole population of attackers, but too slowly. Correct nodes have thus enough time to detect and isolate the malicious ones.

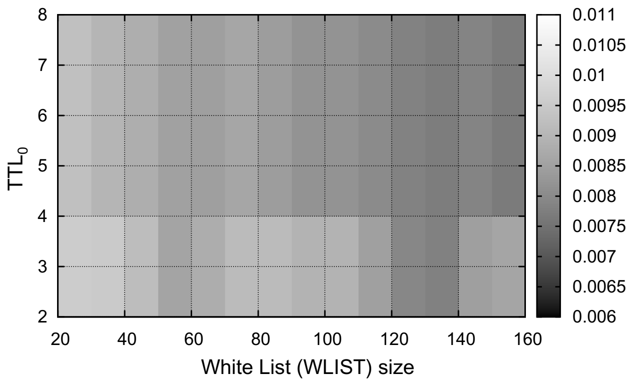


Fig. 8. The average pollution level as a function of ttl_0 and WLIST size. Two gossip exchanges per cycle are considered; $f = 20 = c$ malicious nodes are present. Network size is 10,000.

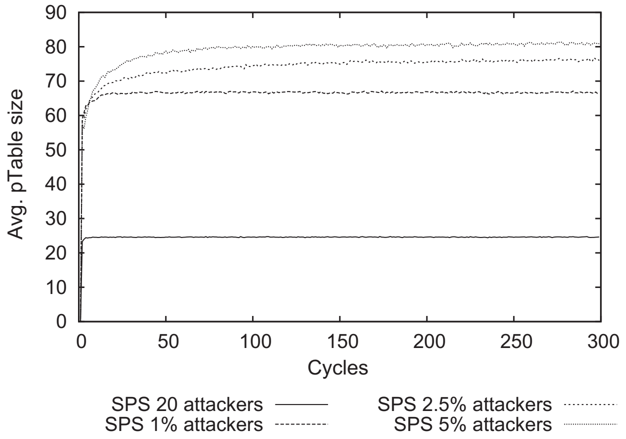


Fig. 9. The average PTABLE size over time. Two gossip exchanges per cycle are adopted; each plot corresponds to a distinct number of attackers in the system (i.e., 20%, 1%, 2.5% and 5% of the network size). Network size is 10,000.

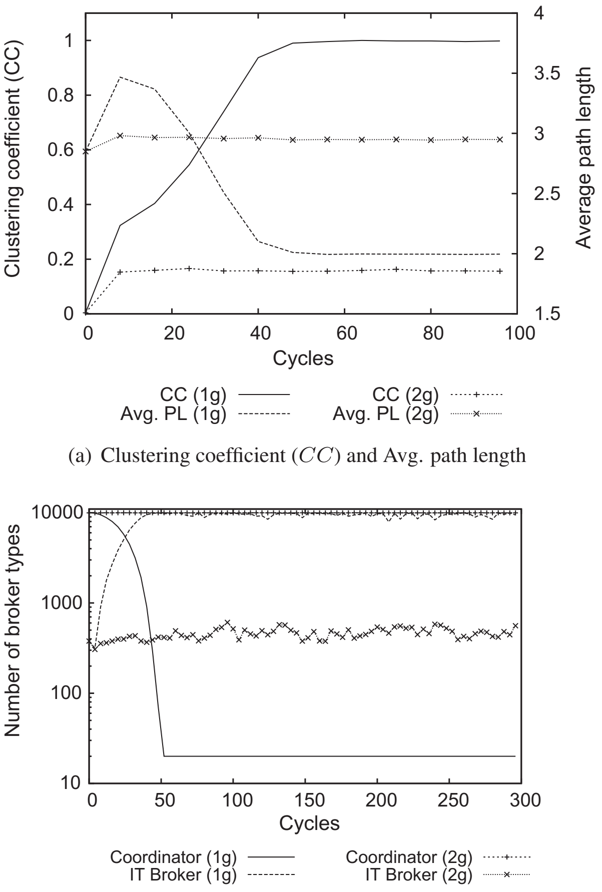
5.2. Parameter evaluation

Fig. 8 shows the impact over the view pollution of the main SPS parameters, such as the initial ttl_0 parameter and the WLIST size. Two gossip exchanges per cycle are considered. The plot refers to the standard attack strategy. Essentially, increasing ttl_0 has a marginal benefit; the maximum allowed size of WLIST instead has a larger impact. According to these data, we adopted $ttl_0 = 4$ and —WLIST— = 100 as basic parameters for our evaluation.

Fig. 9 shows the average size of PTABLE over time. Each plot corresponds to a distinct number of attackers in the system (i.e., 20% and 1%, 2.5% and 5% of the network size). The size of the structure scales very well with the increasing number of attackers and the number of items collected is lower than 1% of the size of the overlay. This means that, when $f \gg c$, a node never gets to know every malicious node, but only those with which it gets in touch more often. Note also that not every descriptor collected in PTABLE links to a malicious node.

5.3. Graph and structure properties

In Fig. 10, we show how the overlay structure evolves over time due to the effect of the hub attack and with the presence of the SPS. We analyze the overlay structure both from a global and a local point of view. The global point of view is shown in terms of clustering coefficient and average path-length, while the local point of view is expressed by the brokerage roles (see Fig. 11).



(a) Clustering coefficient (CC) and Avg. path length

(b) Brokerage roles

Fig. 10. Structural properties of the overlay. (a) The global characteristics in terms of clustering coefficient and average path-length, while (b) the local characteristics in terms of brokerage roles. Network size is 10,000.

The concept of brokers is a common tool in social network analysis; the general idea is that having lots of links within a group exposes a node to the same information over and over again, whereas ties outside one's group yield more diverse information that is worth passing on or retaining to make a profit.

A node connected to other nodes which are themselves not directly connected has opportunities to mediate between them and profit from its mediation. Essentially, this is the ideal condition for a malicious node running the hub attack.

Brokerage roles are calculated over triads of nodes. A triad in which node v mediates transactions between node u and w can display five different patterns of group affiliations. Each pattern is known as a brokerage role. The affiliation we consider is the malicious and nonmalicious group of nodes. We consider just two basic brokerage roles: the *coordinator* and the *itinerant broker* (see Fig. 11). Both roles involve mediation between members of one group. In the former role, the mediator node v is also a member of the same group. While with the latter role, two members of a group use a mediator node v from an outside group.

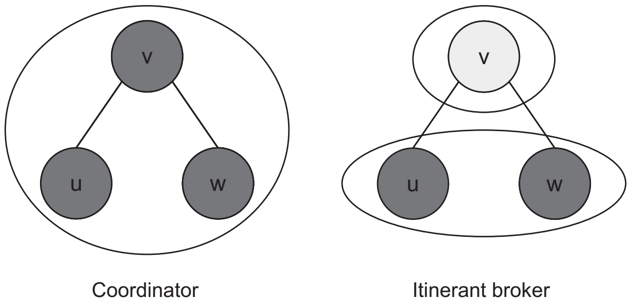


Fig. 11. Brokerage role patterns. Only the subset represented by the *coordinator* and the *itinerant broker* pattern are shown. Links are undirected.

In Fig. 10a, when the PS is not secured (e.g., 1 gossip exchange) the clustering coefficient achieves the unit value in about 50 cycles, meaning that the overlay is extremely dense as all the node's views are completely polluted by malicious descriptors (e.g., as in Fig. 1b). The average path-length value is about 2, which is expected as the hub topology is basically a star: roughly speaking, malicious nodes are in the middle of every path between any pair of well-behaving nodes.

When the overlay is secured by the SPS, the cluster coefficient is much lower (about 0.16) and so is the tendency to generate a dense cluster. However, this value is far away from the clustering coefficient that a real random graph can exhibit (which is in general lower than 0.01), but it is about half of the clustering coefficient usually achieved by Newscast. The overlay induced by the SPS exhibits an average path- length of about 2.9, which is still a very low value considering the size of the network.

In Fig. 10b, when no action is taken to counterstrike the hub attack (1 gossip) the number of coordinators drops dramatically and around cycle 50 the malicious nodes ($f = 20$) are the only coordinators in the system (i.e., attackers are all connected to each other). The number of itinerant broker roles instead has the opposite behavior as they quickly increase their number; this means that correct nodes quickly play the role of bridges between two malicious nodes as they start having their views polluted with malicious descriptors.

When the SPS is switched on, as shown by the dotted plots in Fig. 10b, almost every node can still play the role of coordinator; a feature that is crucial in order to maintain the original relations of the (pseudo) random graph. The relations among the two groups instead are quite limited as the number of itinerant brokers is low. This fact is basically directly related to the low amount of pollution in the views.

5.4. Dynamic scenario

Finally, Fig. 12 shows the behavior of the system in a dynamic scenario. Two gossip exchanges per cycle are adopted, and we show the results for two distinct concentrations of malicious nodes: (a) $f = 20$ and (b) $f = 100$ – i.e., 1% of the network size.

We allowed three distinct churn set sizes: 1%, 5% and 10%, respectively; these percentages of nodes leave the network at every cycle and are substituted by an equal number of new participants. The malicious nodes, instead, stay in place and try to pollute partial views for the whole duration of the experiment. Note that these values are actually exceptionally high [15], demonstrating the feasibility of our solution under harsher conditions.

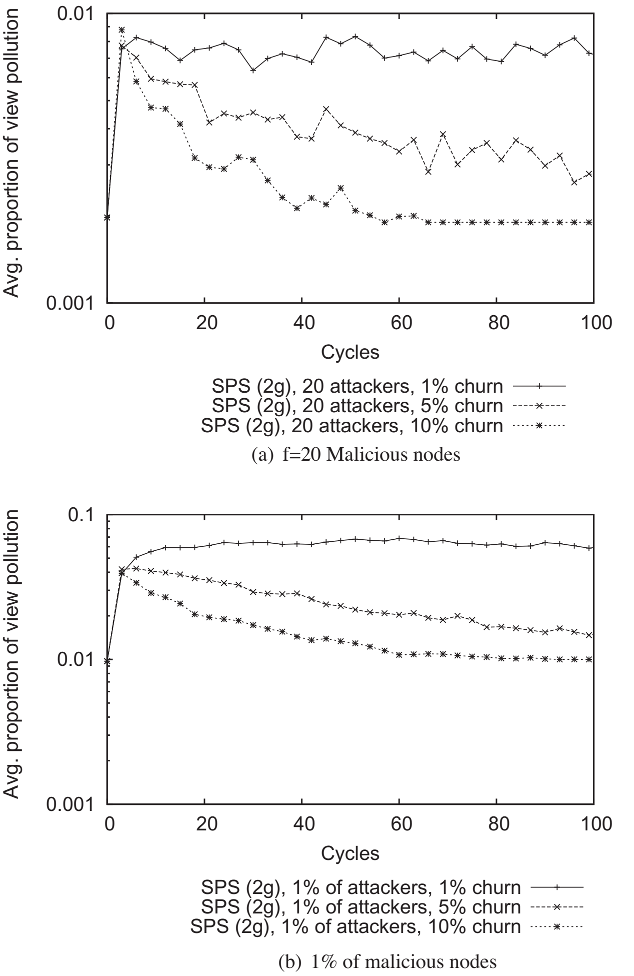


Fig. 12. The average pollution level in a dynamic scenario. (a) The results when $f = 20$ malicious nodes are in the system, while (b) the results when 1% of the overlay is malicious. Two gossip exchanges per cycle are adopted; each plot corresponds to a distinct churn ratio (i.e., 1% and 5% of the network size). Network size is 10,000.

It is surprising to see that actually the network churn helps the SPS to keep the node's views clean. In fact, a higher level of dynamism leads to a lower level of view pollution with respect to the static scenario. 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-node descriptor in their view. The correct nodes that stay in the overlay for a longer time will hardly diffuse the malicious-node descriptors as they have already detected them with high probability. Therefore, it becomes harder and harder for the malicious nodes to diffuse the infection.

5.5. Communication overhead

In comparison to the actual underlying peer sampling adopted, the SPS does not need any extra message. Essentially, the SPS mechanisms are transparently achieved by *multiple*, standard sampling interactions.

In our particular case, as we adopted a Newscast implementation, the cost is $g + 1$ times the cost of each Newscast interaction, where g is the number of distinct explorative gossips per cycle, plus the extra gossip required to minimize the false positive checks. As the average number of exchanges per node can be modeled by a random variable $1 + \phi$ (see [7]), where ϕ has a Poisson distribution with parameter 1, the overall node cost is $(g + 1) \cdot (1 + \phi)$.

However, due to the presence of cryptography, the size of each message could be substantially larger than the adopted peer sampling implementation as we previously discussed in Section 4.5.

6. Related work

Poisoning attacks are closely related to the hub attack. In particular, index poisoning [13] 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 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 [16], the authors mix index poisoning with poisoning routing tables in distributed hash tables (DHT). This combination leads to an effective DoS attack. In this case, a selected victim host is referenced by many other (poisoned) overlay participants, significantly increasing the probability that a message will be routed through the victim.

The BAR model [17] is an interesting approach to deal with Byzantine, Altruistic and Rational (BAR) nodes. The BAR model has been applied to decentralized backup systems [18] and to multimedia streaming applications [19]. BAR gossip relies on two main primitives: (i) verifiable pseudo-random node 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 cooperation among selfish nodes. In our context, we cannot let nodes verify the random selection as partial views are continuously changing and their size is limited. BAR gossip achieves this feature sacrificing dynamic membership: each participant must register 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 every node knows every other node. Again, as each partial view 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 estimates have settled, attackers could already have subverted the network.

In [20], the authors propose a protocol – Fireflies – for intrusion-tolerant overlays. Fireflies is meant as a building block for overlays construction. It provides each node with a complete view of the live nodes in the network. A small portion of these nodes are used as neighbors. The neighborhood relations define a mesh having a diameter logarithmic in the number of live members and connect all reachable members that are not Byzantine. Two possible concerns are represented by the requirement of providing each node with a full membership view and by dealing with the dynamism of the network. However, the system can scale to thousands of nodes. Unfortunately, when the malicious behavior comes into play, the tradeoff between scalability and global knowledge is a key issue and the SPS is no exception as it requires extended knowledge in the form of a knowledge base, compared to the traditional PS. As the malicious population grows, it is necessary to increase more and more the communication effort (i.e., gossips) in order to aggregate the data about the overlay graph and to build solid local knowledge; essentially, this can be seen as mimicking global knowledge.

In [21], the authors present Brahms, a random sampling algorithm suitable to deal with malicious behavior. The design of Brahms is based on the following ideas: (a) to use a gossip-based membership protocol with extra features to deal with an adversarial environment, (b) to understand that this kind of membership protocol induces a bias in the view samples and (c) to correct this bias locally at each node. The authors quantified mathematically the extent of the bias. To achieve an unbiased sample, a specific component is designed to achieve a uniform sample using minwise independent permutations. The unbiased sample is derived from the biased history of gossiped descriptors. Brahms achieves very good results in terms of reducing the pollution of the partial views in the presence of a large proportion of malicious nodes, while keeping sublinear membership view and without requiring cryptography nor a certification authority. However, the convergence to a uniform random sample is achieved when the churn ceases, which is a very unlikely scenario in a real-world setting.

Social inspirations and social techniques are gaining popularity in network protocols. In [22], the authors introduce 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 validate 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 relies on the presence of a working DHT facility that adds complexity and may become an extra security vulnerability as it can be attacked as well.

Social network principles (e.g., reciprocity and structural holes) are also adopted in JetStream [10] to optimize and build robust gossip systems. The basic idea is to make a predictable node selection when gossiping in order to avoid unpredictable, excessive message overhead. In addition, the traditional scalability and reliability of gossip are maintained.

7. Conclusions

Peer sampling is a fundamental building block for gossip-based, large-scale distributed systems. The current state-of-the-art systems, with a few exceptions, are capable of dealing with churn and crash failures, but are vulnerable to malicious attacks. We believe that the ‘‘hub attack” is a simple, but realistic model that poses a real threat to current peer-sampling mechanisms.

This paper proposes an extension to existing peer sampling services (in particular, Newscast), which is capable of reducing the impact that malicious nodes may have on the quality of the sampling. An extensive experimental evaluation shows that our extension is efficient in dealing with the malicious threat, even when a large proportion of malicious nodes are present. In addition, this result is achieved with local and limited knowledge of the network.

As these result seems to point in the right direction, we plan to test a prototype implementation of our solution in a real distributed environment.

Acknowledgment

This work is supported by the Autonomous Security project, financed by MIUR Programme PRIN 2008.

References

[1] A.J. Demers, D.H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H.E. Sturgis, D.C. Swinehart, D.B. Terry, Epidemic algorithms for replicated database maintenance, in: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing Systems (PODC'87), 1987, pp. 1–12.

[2] M. Jelasity, O. Babaoglu, T-Man: gossip-based overlay topology management, in: ESOA, Lecture Notes in Computer Science, vol. 3910, Springer, 2005, pp. 1–15.

[3] M. Jelasity, A. Montresor, O. Babaoglu, The bootstrapping service, in: I.C. Society (Ed.), Proceedings of the 26th International Conference on Distributed Computing Systems Workshops (ICDCS WORKSHOPS), Lisboa, Portugal, 2006, p. 11.

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

[5] G.P. Jesi, A. Montresor, O. Babaoglu, Proximity-aware superpeer overlay topologies, in: SelfMan'06, Lecture Notes in Computer Science, vol. 3996, Springer, Dublin, Ireland, 2006, pp. 43–57.

[6] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, M. van Steen, Gossip-based peer sampling, ACM Trans. Comput. Syst. 25(3) (2007).

[7] M. Jelasity, W. Kowalczyk, M. van Steen, Newscast computing, Tech. Rep. IR-CS-006, Vrije Universiteit, Amsterdam, The Netherlands, November 2003.

[8] S. Voulgaris, D. Gavidia, M. van Steen, CYCLON: inexpensive membership management for unstructured P2P overlays, J. Network Syst. Manage. 13(2) (2005).

[9] S. Wasserman, K. Faust, Social Network Analysis: Methods and Applications, Cambridge University Press, Cambridge, 1994.

[10] J.A. Patel, I. Gupta, N. Contractor, JetStream: achieving predictable gossip dissemination by leveraging social network principles, in: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications (NCA 2006), Cambridge, MA, USA, 2006, pp. 32–39.

[11] J.R. Douceur, The Sybil attack, in: IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Springer-Verlag, London, UK, 2002, pp. 251–260.

[12] S.J. Nielson, S. Crosby, D.S. Wallach, A taxonomy of rational attacks, in: IPTPS, Lecture Notes in Computer Science, vol. 3640, Springer, 2005, pp. 36–46.

[13] J. Liang, N. Naoumov, K. Ross, The index poisoning attack in P2P file sharing systems, in: INFOCOM, Barcelona, Spain, 2006.

[14] M. Jelasity, A. Montresor, G.P. Jesi, S. Voulgaris, The Peersim simulator, <http://peersim.sf.net>.

[15] R. Mahajan, M. Castro, A. Rowstron, Controlling the cost of reliability in peer-to-peer overlays, in: IPTPS, Lecture Notes in Computer Science, vol. 2735, Springer, 2003, pp. 21–32.

[16] N. Naoumov, K. Ross, Exploiting P2P Systems for DDoS Attacks, in: InfoScale, ACM Press, New York, NY, 2006, p. 47.

[17] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, M. Dahlin, BAR gossip, in: OSDI, 2006.

[18] A.S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, C. Porth, Bar fault tolerance for cooperative services, SIGOPS Oper. Syst. Rev. 39 (5) (2005) 45–58.

[19] H.C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, M. Dahlin, Flightpath: obedience vs. choice in cooperative services, in: R. Draves, R. van Renesse (Eds.), Proceedings of the Eighth USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8–10, 2008, San Diego, California, USA, USENIX Association, 2008, pp. 355–368.

[20] H. Johansen, A. Allavena, R. van Renesse, Fireflies: scalable support for intrusion-tolerant network overlays, SIGOPS Oper. Syst. Rev. 40 (4) (2006) 3–13.

[21] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, A. Shraer, Brahms: Byzantine resilient random membership sampling, in: Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC), 2008.

[22] M. Sherr, B.T. Loo, M. Blaze, Veracity: a fully decentralized service for securing network coordinate systems, in: Seventh International Workshop on Peer-to-Peer Systems (IPTPS 2008), 2008.

Gian Paolo Jesi got his Ph.D. in Computer Science from the University of Bologna. His research interests are focused on large-scale P2P systems, emergent behavior, epidemic protocols and application of bio-inspired approaches to distributed systems problems. He received his M.Sc. in computer science from the University of Bologna in 2002; after a short work experience in an Italian IT company, he had the opportunity to move to the University of Bologna joining the BISON project. In 2009 he moved to the University of Trento.

Alberto Montresor is Associate Professor of Computer Science at the University of Trento, Italy. He received his Ph.D. in 2000 from the University of Bologna, Italy, where he designed Jgroup, a partition-aware group communication system. He served as Research Associate in Bologna until 2005, when he moved to Trento. He is author of more than 40 papers in international conferences and journals, and he has been active in several European projects in the field of distributed computing and complex adaptive systems.

Maarten van Steen is full professor in the Computer Systems Group at the Vrije Universiteit Amsterdam (VUA). He teaches modules and courses covering distributed systems, computer networks, operating systems, and complex networks to academics and professionals. He has co-authored two textbooks on networked computer systems. His research concentrates on large-scale distributed systems with a strong emphasis on adaptive techniques that support automatic replication, management, and organization of wired and wireless systems. Recently, he has been exploring gossip-based solutions to achieve decentralized autonomous systems, partly focusing on very large wireless sensor networks and pervasive computing. He is furthermore consultant for Philips Research, and closely participates with a collaboration of high-tech SMEs for developing and deploying real-world pervasive computing systems.