Detection and Removal of Malicious Peers in Gossip-Based Protocols*

Márk Jelasity, Alberto Montresor, Ozalp Babaoglu

Department of Computer Science, University of Bologna, Italy
e-mail: jelasity,montreso,babaoglu@cs.unibo.it

1 Introduction

In addition to the popular structured peer-to-peer (P2P) overlays [2], a class of P2P protocols rely purely on gossip-based communication in an *unstructured* communication topology [6]. Examples of problems that can be solved through these protocols include membership management, information dissemination (as in lpbcast [5] and newscast [7]), and computation of aggregate functions (such as average and maximum) over distributed collections of numeric values [8, 10].

Since they do not rely on specific topologies such as trees, rings, butterflies, etc., gossip-based protocols over unstructured topologies are potentially more robust to massive benign failures. They are also extremely responsive and can adapt rapidly to changes in the underlying communication structure. And finally, they can be used as robust components to build other protocols. For example, [12] describes a technique for repairing or jump-starting a structured overlay using newscast.

Properties that make unstructured gossip-based protocols attractive (in particular responsiveness and adaptivity) when all peers cooperate correctly become a detriment when even a small number of peers may be *malicious* and compromise the integrity of the system.

In our opinion, a broader and more significant exploitation of the gossip paradigm will be possible only if we can develop a better understanding of the issues related to security in such protocols. This short paper is aimed at raising the problem and suggesting some initial ideas towards solutions. We first give a brief overview of the gossip paradigm. Then, we introduce a general framework for solving the security problem in such systems. Finally, we illustrate some preliminary results for the case of an aggregation protocol.

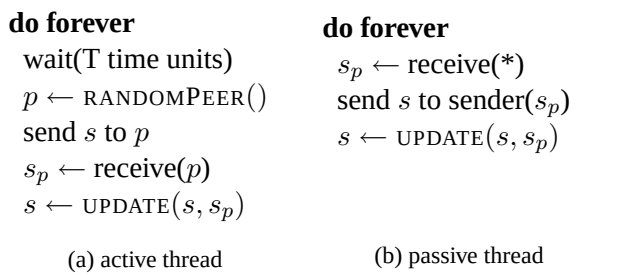
*This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923).

2 Gossip-Based Protocols

Figure 1 illustrates the skeleton of a generic gossip-based protocol. Each node possesses a local state and executes two different threads. The *active* one periodically initiates a *state exchange* with a random peer by sending it a message containing the local state and waiting for a response. The *passive* thread waits for messages sent by an initiator and replies to them with the local state. The random peer selection is based on the set of neighbors as determined by a membership protocol [5, 7].

Method UPDATE builds a new local state based on the previous local state and the state received from the random peer. Semantics of the node state and the method UPDATE define the behavior and function of the protocol. For example, in *membership management*, a fixed-size set of peer addresses called the *partial view* constitutes the state. Method UPDATE can be implemented (as in lpbcast) by computing the new state through a random sampling of the union of the old view and the received view. In this case, selection of the random peer is done using the old view itself. In *epidemic broadcasting*, the state is a flag that records if the node is *infected* or not. Method UPDATE sets the state to infected if the received state is infected.

To make our reasoning concrete, the discussion that follows is based on *aggregation*. Here, the state of a node is a numeric value. In a practical setting, this value can be any attribute of the environment, such as the current load or storage capacity. The task of the protocol is to calculate an aggregate value over the set of all numbers stored at nodes. Several aggregate functions are possible, including extremal values, average, sum, counting, etc. For example, if we are interested in the *average*, method UPDATE simply returns the arithmetic mean of the two states. The resulting protocol is know to result in exponential convergence to the correct global average at each node [8, 10].



(a) active thread (b) passive thread

Figure 1: The skeleton of a gossip-based protocol where s denotes the local state and s_p the state of the peer p.

3 Defense against Malicious Attacks

A malicious node can deviate from its protocol in essentially three ways: First, it can communicate more frequently than prescribed by the protocol, "pumping" unwanted information into the system. In a broadcast application, this may lead to denial-of-service attacks. Second, it can communicate less frequently than prescribed. This behavior can be considered as a general benign failure; gossip-based protocols are extremely robust with respect to this, so no new measures are needed. Third, a malicious node can respect communication frequency but it might apply an incorrect UPDATE method, which can arbitrarily change the collective behavior. For example, in average aggregation, a malicious node can maintain a large constant value, which eventually causes the global average to incorrectly grow instead of converging.

We can identify four subproblems that need to be solved in order to maintain system integrity in the presence of malicious peers. The first problem is *identity assignment*: how to guarantee that a real-world *entity* (person, organization) obtain only a limited number of virtual *identities* [4], that these identities be unforgeable and that the source of information in the system be attributable to a valid identity. These goals can be achieved through public-key cryptography and a trusted certificate authority (CA) as described in [1]. The central CA is not a performance bottleneck because it does not participate in the protocol and it can even be offline.

The second problem is the *real-world interface*, which is present in any setting where peers in the system inject information derived from their external environment. Typical examples are sensor networks where unstructured gossip-based protocols may find applications. Preventing such attacks is an application-specific problem where techniques like smoothing, interval bounding, background knowledge, machine learning, etc. can be applied. In this work we do not address this problem.

In the following we focus on the two remaining problems: detection of nodes that do not behave according to the prescribed protocol and their removal from the system. Space limitations force us to give only sketches of our solutions. Additional information and details may be found in [9].

3.1 The Detection Problem

The goal of detection is to find a *proof* that a node deviated from the behavior prescribed by its protocol. Our solution to this problem is not unlike the random auditing technique described in in [11], in that it involves participating nodes to periodically verify each other's actions. Verification is achieved through *communication logs* that nodes maintain locally and provide to others when asked for. A communication log is a list of records corresponding to all communication events (both initiated and incoming) at a node within a given time window of fixed size that ends with sending the log.

We augment the protocol in Figure 1 so that during each state exchange, the peers agree on a unique exchange ID and exchange local timestamps. With this additional information, each log record is built to contain the local timestamp, the remote timestamp, the exchange ID and the identity of the remote peer involved in the communication. All information originating at remote peer p is signed by p. Additionally, each record may contain application-specific data related to the communication event.

A given log is said to be *legal* if it conforms both to the syntactic structure defined above and to the semantic structure defined by the application. Two logs are said to be mutually *inconsistent* if it is logically impossible for both of them to be legal at the same time.

Each time node A contacts a peer B according to the protocol in Figure 1, it asks B for its log, denoted L_B, in addition to its state. Node A subsequently picks a random peer of B, say C, from B's log and asks C for its log, L_C, as well. The key idea is that if we define the semantics of the log properly, then if either B or C is malicious, then at least one of the following is true with significant probability: (a) one or both of L_B and L_C are provably illegal, (b) both logs are legal but

they are provably inconsistent. As described below, these conditions can be verified by A so it can make a decision whether B or C or both are malicious. All information included in the logs being signed by specific identities, the pair of logs constitute a proof that one or more malicious identities have behaved incorrectly. These proofs cannot be forged.

To provide an example of the log verification process, we discuss the case in which a malicious node applies a wrong UPDATE method to its state. Application-specific data are necessary to detect this kind of cheating, so we concentrate on average calculation.

In average calculation, we require nodes to attach to each log record their current local value and the value received from the peer. In a legal log, the local value recorded has to be the average of the previous local value and the previous received value. Furthermore, a node cannot forge a received value (recall that we require each piece of information to be signed by its originator in the system). Using these properties, it is clear that for C to forge its log, it has to either forge at least one record by changing its local value in it, or C has to remove all records preceeding the last cheating so as to avoid detection of the inconsistency.

In the first case, if the forged record is the one under examination, the verification process will discover the inconsistency, because the other log under examination will contain a completely different record. The removal of multiple records in the log will be detected more easily, because several other nodes may contain records in their logs proving that the node had actually communicated during the log time window.

It is important to note that a single isolated malicious act could pass undetected by this mechanism: it is possible that two logs whose comparison would lead to a proof, are never randomly selected. This is not always a problem: for example, in the aggregation protocol, a single false value in a large scale system does not modify the average in a significant way. Furthermore, if a malicious node stops acting maliciously after some point, it is no longer necessary to remove it anyway. Nevertheless, a malicious node that keeps acting maliciously will eventually be detected by some peer reasonably rapidly, depending on the frequency of malicious acts and depending also on the specific application and the semantics of the log.

Clearly, to assess the correctness of this detection mechanism, a complete analysis of all possible attacks would be needed. Several details did not find their way in the brief description above. For example, we have not discussed what happens if a node refuses to provide its log; this problem can be easily solved through an indirection mechanism, that forces nodes to make their logs available if they want to participate in the protocol. Other problems to be considered include frequency attacks (where nodes start more exchanges than prescribed by the protocol), cooperation among nodes, etc. For additional details, please consult [9].

3.2 The Removal Problem

It is possible to remove detected malicious nodes through a centralized solution. Yet, in our framework, we make full use of the power of gossip-based database updates [3]. Let each node maintain a *blacklist* of identities. Nodes will not accept connections from blacklisted identities and will not initiate connections to them, effectively cutting them off from the network.

We can think of the blacklist as a database of malicious node identities. When a node detects a malicious node, it simply inserts it in its blacklist and propagates this information as a database update using a classical gossip-based protocol of choice [3].

Malicious nodes may of course want to propagate correct nodes as blacklist updates. Since this requires proof, which they cannot forge, it is not possible.

4 An Illustrative Example

To prove the effectiveness of our approach, we present an example scenario based on our aggregation protocol for averaging [8]. In this example, aggregation is used to compute the network size as follows. Exactly one node sets its initial value to 1, while all the others set it to 0. The averaging is then started and the values held by all nodes converge exponentially to $1/N$, where N is the network size. Since convergence is very fast, the protocol is restarted every 20 cycles (an *epoch*), and the converged value at the end of each epoch is used as a dynamic approximation of $1/N$.

Size estimation has two remarkable characteristics. First, it is highly sensitive to malicious attacks. In the initial cycles, if a node that started with value 1 communicates with a malicious node that always reports 0 as its current value (irrespective of the values it receives from other peers), it is possible that a large share of the initial 1 value is removed from the network, thus increasing the network size estimate. Second, in this problem the real-world interface problem is not present

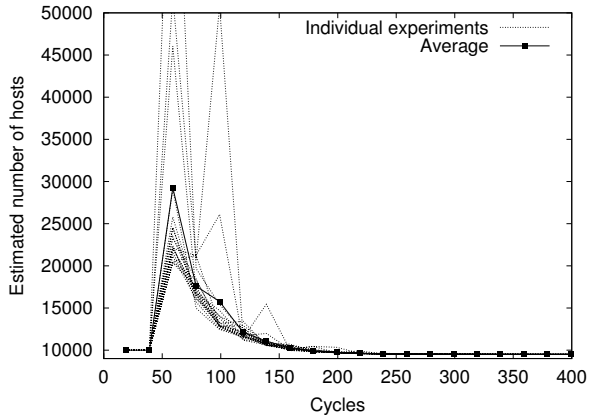


Figure 2: Response to sudden collective attack. 500 nodes in a network of 10^4 nodes start to behave maliciously at cycle 40 by reporting 0 as their current value. The network size estimated during 20 independent experiments is shown using dashed lines. The thick line represents the average computed over those experiments.

because the numbers to average are defined by the protocol itself; only values included in the interval $[0, 1]$ are possible.

Figure 2 shows the results. In the attack we consider, many malicious nodes enter the system at once. Although the system we consider is not synchronous, it is convenient to subdivide the computation into cycles — consecutive wall clock intervals during which every node has the chance to perform a state exchange.

In a network of 10^4 nodes, 500 (random) nodes become malicious at cycle 40. They communicate with the correct frequency, but during each value exchange, they always report 0 as their value. We can see that as the malicious nodes gradually get blacklisted the approximation approaches the correct value of 9500.

5 Conclusions

In this paper, we have discussed the security problem in gossip-based protocols. We have suggested a framework for maintaining integrity in a class of unstructured gossip-based P2P protocols in a fully autonomic fashion (except for the identity problem).

It is interesting to note that non-cooperative, malicious nodes are a problem not only for gossip-based protocols, but in general for any kind of peer-to-peer system in which the correct functioning of the protocol is based on the cooperation among nodes. We believe that the solution we have sketched in this paper can potentially be applied also to other, more structured protocols.

References

[1] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proc. 5th Usenix Symp. on Op. Sys. Design and Implementation (OSDI 2002)*, Boston, Massachusetts, 2002.

[2] F. Dabek and B. Zhao. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proc. of the 2nd Int. Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, Feb. 2003.

[3] A. Demers et al. Epidemic Algorithms for Replicated Database Management. In *Proc. 6th ACM Symp. on Principles of Dist. Comp. (PODC'87)*, pages 1–12, Vancouver, August 1987. ACM.

[4] J. R. Douceur. The sybil attack. In *Proc. 1st Int. Workshop on P2P Sys. (IPTPS'02)*, March 2002.

[5] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight Probablistic Broadcast. *ACM Trans. on Comp. Sys.*

[6] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. From Epidemics to Distributed Computing. *IEEE Computer*, 2003.

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

[8] M. Jelasity and A. Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks. In *Proc. of the 24th Int. Conference on Distributed Computing Systems (ICDCS 2004)*, Tokyo, Japan, 2004.

[9] M. Jelasity, A. Montresor, and O. Babaoglu. Towards secure epidemics: Detection and removal of malicious peers in epidemic-style protocols. Technical Report UBLCS-2003-14, University of Bologna, Dept. of Computer Science, Bologna, Italy, Nov. 2003.

[10] A. Montresor, M. Jelasity, and O. Babaoglu. Robust Aggregation Protocols for Large-Scale Overlay Networks. In *Proc. of the Int. Conf. on Dependable Systems and Networks (DSN 2004)*, Firenze, Italy, 2004.

[11] T.-W. Ngan, D. S. Wallach, and P. Druschel. Enforcing Fair Sharing of Peer-to-Peer Resources. In *Proc. 2nd Int. Workshop on P2P Sys. (IPTPS'03)*, Berkeley, CA, USA, February 2003.

[12] S. Voulgaris and M. van Steen. An Epidemic Protocol for Managing Routing Tables in Very Large Peer-to-Peer Networks. In *Proc. 14th IFIP/IEEE Int. Workshop on Dist. Sys.: Operations and Management, (DSOM 2003)*, number 2867 in LNCS. Springer, 2003.