Towards Secure Epidemics: Detection and Removal of Malicious Peers in Epidemic-Style Protocols

Márk Jelasity **Alberto Montresor** **Ozalp Babaoglu**

Technical Report UBLCS-2003-14

November 2003

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in gzipped PostScript format via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`. All local authors can be reached via e-mail at the address *last-name*`@cs.unibo.it`.

Recent Titles from the UBLCS Technical Report Series

2002-8 *User Untraceability in the Next-Generation Internet: a Proposal*, Tortonesi, M., Davoli, R., August 2002.

2002-9 *Towards Adaptive, Resilient and Self-Organizing Peer-to-Peer Systems*, Montresor, A., Meling, H., Babaoglu, O., September 2002.

2002-10 *Towards Self-Organizing, Self-Repairing and Resilient Distributed Systems*, Montresor, A., Babaoglu, O., Meling, H., September 2002 (Revised November 2002).

2002-11 *Messor: Load-Balancing through a Swarm of Autonomous Agents*, Montresor, A., Meling, H., Babaoglu, O., September 2002.

2002-12 *Johanna: Open Collaborative Technologies for Teleorganizations*, Gaspari, M., Picci, L., Petrucci, A., Faglioni, G., December 2002.

2003-1 Security and Performance Analyses in Distributed Systems (Ph.D Thesis), Aldini, A., February 2003.

2003-2 Models and Types for Wide Area Computing. The calculus of Boxed Ambients (Ph.D. Thesis), Crafa, S., February 2003.

2003-3 MathML Formatting (Ph.D. Thesis), Padovani, L., February 2003.

2003-4 Performance Evaluation of Mobile Agents Paradigm for Wireless Networks (Ph.D. Thesis), Al Mobaideen, W., March 2003.

2003-5 Synchronized Hypermedia Documents: a Model and its Applications (Ph.D. Thesis), Gaggi, O., March 2003.

2003-6 Searching and Retrieving in Content-Based Repositories of Formal Mathematical Knowledge (Ph.D. Thesis), Guidi, F., March 2003.

2003-7 Intersection Types, Lambda Abstraction Algebras and Lambda Theories (Ph.D. Thesis), Lusin, S., March 2003.

2003-8 Towards an Ontology-Guided Search Engine, Gaspari, M., Guidi, D., June 2003.

2003-9 An Object Based Algebra for Specifying A Fault Tolerant Software Architecture, Dragoni, N., Gaspari, M., June 2003.

2003-10 A Scalable Architecture for Responsive Auction Services Over the Internet, Amoroso, A., Fanzieri F., June 2003.

2003-11 WSSecSpaces: a Secure Data-Driven Coordination Service for Web Services Applications, Lucchi, R., Zavattaro, G., September 2003.

2003-12 Integrating Agent Communication Languages in Open Services Architectures, Dragoni, N., Gaspari, M., October 2003.

2003-13 Perfect load balancing on anonymous trees, Margara, L., Pistocchi, A., Vassura, M., October 2003.

2003-14 Towards Secure Epidemics: Detection and Removal of Malicious Peers in Epidemic-Style Protocols, Jelasity, M., Montresor, A., Babaoglu, O., November 2003.

2003-15 Gossip-based Unstructured Overlay Networks: An Experimental Evaluation, Jelasity, M., Guerraoui, R., Kermarrec, A-M., van Steen, M., December 2003.

2003-16 Robust Aggregation Protocols for Large-Scale Overlay Networks, Jelasity, M., Montresor, A., Babaoglu, O., December 2003.

Towards Secure Epidemics: Detection and Removal of Malicious Peers in Epidemic-Style Protocols[1]

Márk Jelasity[2] **Alberto Montresor**[2] **Ozalp Babaoglu**[2]

Technical Report UBLCS-2003-14

November 2003

Abstract

In this paper we address the problem of maintaining integrity in the presence of malicious peers in a large class of epidemic-style protocols. This class includes membership management, information dissemination and proactive aggregation. We present solutions to the problems of identifying non-cooperating (malicious) nodes and removing them from the system. Our solutions are fully decentralized other than requiring an offline certificate authority to bind identity information to joining entities. We also present empirical evidence to illustrate the performance of our solutions.

1. This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project BISON (IST-2001-38923).
2. Department of Computer Science, University of Bologna, Italy

do forever
  wait(T time units)
  p = getRandomPeer()
  send s to p
  receive s_p from p
  $s \leftarrow$ updateState(s, s_p)

Figure 1. The skeleton of a push-pull epidemic protocol. Notation: s is the state of this node, s_p is the state of the peer p.

1 Introduction

Beyond the popular structured peer-to-peer (P2P) overlays, there exists a class of P2P protocols that rely purely on epidemic-style communication in an *unstructured* communication topology [5]. Examples of problems that can be solved through these protocols include membership management, information dissemination (as in lpbcast [4] and newscast [6]), and calculating aggregates of numeric values (like average and maximum) [7].

Since they do not rely on a specific topology such as a tree, ring, butterfly etc., epidemic-style protocols over unstructured topologies are attractive since they 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 supporting other protocols. For example, [10] describes a technique for repairing or jump-starting a structured overlay using an unstructured low-level layer, newscast.

Properties that make unstructured epidemic protocols attractive (in particular responsiveness and extreme adaptivity) when all peers cooperate correctly become a detriment when even a small number of peers may be *malicious* and can compromise the integrity of the system. In this position paper we suggest fully distributed solutions for detecting malicious peers and removing them automatically.

Section 2 defines the class of protocols we are targeting. Section 3 describes our approach and finally in Section 4 experimental results are presented.

2 Epidemic-Style Protocols

Figure 1 illustrates the skeleton of an epidemic protocol based on the *push-pull* interaction model where both parties participate symmetrically in each exchange. According to this protocol, each node periodically initiates an information exchange with a randomly-selected peer and subsequently updates its state based on the information received. The receiving (passive) peer executes a symmetric algorithm. The random peer to exchange information with is normally selected using a membership protocol [4, 6] and we ignore its details here.

Semantics of the node state and the method updateState define the behavior and function of the protocol. We give several examples below to illustrate the generality of this paradigm.

In *membership management*, a fixed-size set of peer addresses called the *partial view* constitutes the state. Method updateState could be implemented (as in lpbcast) by computing the new state through a random sampling of the union of the old view and the view of the peer. In this case, selection of the random peer is done using the old view itself.

In *broadcasting*, the state is a flag that records if the node is *infected* or not. Method updateState sets the state to infected if the received state is infected. For a range of more practical implementations see [2].

Aggregation is yet another interesting instance of this paradigm. Let the state be an arbitrary numeric value. If we are interested in the *average* of these values over all members, method updateState returns the mean of the two states. This protocol will result in exponential convergence to the correct global average at each node [7]. Note that if method updateState returns the maximum or minimum, then the system converges to the global maximum or minimum, respectively.

3 Defense against Malicious Attacks

We can identify four subproblems that need to be solved in order to maintain system integrity in the presence of malicious peers. The first is the *identity assignment* problem: how to guarantee that a real-world *entity* (person, organization) be able to obtain only a limited number of virtual *identities* [3], 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 to be solved is what we call 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 some of the unstructured epidemic protocols can 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 concentrate 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 and leave out much of the detail.

3.1 The Detection Problem

We say that a malicious node M is *detected* if at least one correct node obtains *proof* that M deviated from the behavior prescribed by its protocol.

A malicious node can deviate from an epidemic-style protocol in essentially two ways. First, it can communicate more frequently that prescribed by the protocol and in this way it can pump misleading information into the system more rapidly. Second, a malicious node can respect communication frequency but it might apply an incorrect updateState method, which can arbitrarily change the collective behavior in many applications. For example, in aggregation to compute the average, a malicious node can maintain a large constant value, which eventually causes the global average to incorrectly grow instead of converging. In a membership protocol, a malicious node could keep reporting a set containing other malicious nodes in order to increase their presence in the views of correct peer nodes.

Note that epidemic protocols are extremely robust to faulty nodes that communicate *less* frequently than prescribed since this behavior is included in general benign failures, which can be tolerated without any extra measures [7].

Our solution to the detection problem is not unlike the random auditing technique described in [8], in that it involves participating nodes to periodically verify each other's actions. Verification is achieved through histories that nodes maintain locally and provide to others when asked for.

Communication histories: are lists 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 history.

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 history 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 a remote peer is signed by the remote peer. Additionally, each record may contain application-specific data related to the communication event.

In the history, the first and the last records must be marked as such, and the history must contain the signed local time corresponding to the request for the history (which also defines the beginning of the history since the window length is fixed). A given history is said to be *legal* if it conforms both to the syntactic structure defined above and to the communication structure defined by the application. Note that it is possible for a legal history to have "holes" (the node could be temporarily offline) in it or be empty (the node has just joined the network). Two histories are said to be mutually *inconsistent* if it is logically impossible that both are true at the same time.

History verification. Each time node A contacts a peer B according to the protocol in Figure 1, it asks B for its history, which is H_B, in addition to its state. Node A subsequently picks a random peer of B, say C, from B's history and asks C for its history, H_C, as well. The key idea is that if we define the semantics

of the history 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 H_B and H_C are provably illegal, (b) both histories 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.

Although the detection of a single isolated malicious act cannot be achieved with absolute certainty using this method, 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 history. This is not a severe restriction since if a malicious node stops acting maliciously after some point, it is no longer necessary to remove it anyway.

In the following, we continue to use the notation A, B and C to refer to the three peers of the above scenario and assume that C is malicious while A and B are correct. Note that the roles of C and B are symmetric in the verification process because B and C do not know each other. So the case where B alone is malicious is identical to C being malicious. The other cases (A or both B and C malicious) are dealt with later.

The frequency attack. If the "sin" of C is that it communicated too frequently, then it has to remove entries from its history in an effort to pass the legality check by A. Suppose C initiates two communication events (rather than one) per cycle of the protocol and removes half of them from its history in an effort to hide this (faulty) behavior. The probability that A will detect this (malicious) behavior of C during the consistency check is at least 0.5 since C cannot predict that A will pick B for comparison, thus the probability of having removed the wrong communication record from its history is given through a coin toss. In general, frequency attacks are very easily detected: the more frequently a node communicates, the more likely it will fail verification. The two histories H_B and H_C possessed by A serve as a proof because H_B contains an exchange ID and remote timestamp, both signed by C, that are missing from H_C. This can be proven using the time information included in H_C and the (signed) timestamp held by B about the local time at C during the exchange. For this we also have to make sure that the exchange in question is well within the time window of the histories (e.g., A can always pick a record from the first half).

The forged state attack. If C communicates with the correct frequency, then H_C will correctly include B, so application-specific data is necessary to detect cheating. The point is that an incorrect updateState should provoke inconsistencies between the histories of B and C. Let us examine average calculation as an example.

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

In the first case, if the record corresponding to the exchange under examination is forged, A will detect the inconsistency with the record of the exchange in H_B. These two conflicting records can be used as proof that C is malicious. In the second case, the situation looks like a frequency attack by C so that is detected as well, in fact more easily because the removal of each record counts as an independent malicious act.

What if A or both B and C are malicious? If A is malicious it has no incentive to do the verification correctly. It might even wish to declare an otherwise correct node malicious. However, not being able to forge proofs, A cannot do that.

If both B and C are malicious, they have little chance to cooperate, even if they otherwise wanted to, because the verification process is anonymous, in the sense that C does not know and has no control over exactly on whose behalf it is being verified.

Other attacks. A node can simply play dead and not answer any requests, including those for its history. Being down is perfectly legal, so it is not suspicious by itself, but in the meantime, the node can actively contact peers and act maliciously.

To solve this we introduce an *indirection* mechanism, by replacing the direct contact step by a request for callback. The contacted peer then selects a random peer and asks it to callback the original caller. This way, malicious nodes have to answer all requests (they cannot play dead) because each call might be a callback.

However, a node can still refuse to send its history, and this fact cannot be proven. (Otherwise it could count as detecting a malicious node.) Furthermore, it can send a faulty history unsuitable for being used as proof (missing signatures, etc.). Enforcing the history-contract, and the very fact that each node can get a history from each other node can be solved by having each node send its history *regardless* of request, in the ping-phase of building a connection. This way, for *any* communication a node has to provide a proper history first. This does not introduce additional overhead because during each connection histories are exchanged anyway.

3.2 The Removal Problem

A centralized solution is always possible for the removal of nodes. However, in our framework we can make full use of the power of epidemic database updates [2].

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 epidemic (or in fact gossiping) protocol of choice [2].

Even if we assume that malicious nodes do not forward blacklist updates, this mechanism works as long as the sub-network of non-malicious nodes forms a connected graph. For example, with newscast graphs of degree (i.e., view size) 20, it is possible to remove approximately 60% of the nodes while the remaining graph remains connected. With degree 100, approximately 90% of the nodes can be removed [6].

Malicious nodes can of course wish 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 [7]. 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 $1/N$.

To apply this protocol in practice, several implementation details need to be sorted out. For example, the single-value assumption may be relaxed by having multiple average computations that run concurrently, each of them appropriately tagged by the identifier of one of the nodes that started with a value 1. Due to space reasons, the reader is referred to [7] for details.

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 estimate. Second, in this problem the real-world interface problem is not present because numbers to average are defined by the protocol itself; only values included in the interval $[0, 1]$ are possible.

In Figure 2, we plot these converged values at the end of each epoch. The epoch length and the time window of the communication history are both 20 cycles.

In the attack we consider, many malicious nodes enter the system at once. In a network of 10^4 nodes, 500 (random) nodes become malicious at cycle 40. They communicate with the correct frequency, which makes their detection harder, but during each value exchange, they always report 0 as their current value.

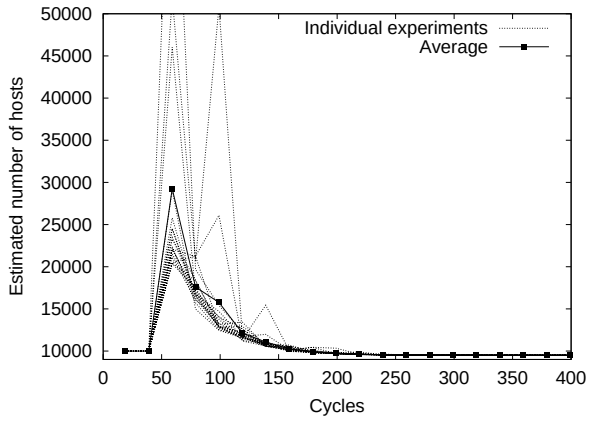

Figure 2. Response to sudden collective attack. 500 nodes in a network of 10^4 nodes start to behave maliciously at cycle 40 by reporting 0 as their current value. The size estimated during 20 independent experiments is shown using dashed lines. The thick line represents the average computed over those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

This choice of value by the malicious nodes is their "best bet" for three reasons. First, 0 is always a legal value, so it cannot be rejected by the correct peers. Second, sending a too large value would raise suspicions, especially when the system is already close to the global average. Third, with this value, malicious nodes hope to achieve maximal inconsistency since the network size estimate is the reciprocal of the average value (which is $1/N$) and forcing it closer to 0 would result in a larger error for the network size estimate.

In Figure 2 we can see that as the malicious nodes gradually get blacklisted the approximation shows the correct value, 9500.

5 Related Work

As we mentioned, our approach shares some basic ideas from secure structured overlays [1, 8].

Another approach for secure epidemic protocols is implemented in Astrolabe [9]. The security approach is based on the Astrolabe hierarchy in which each zone belongs under a separate administrator, which provides central control over the zone. Our approach however is autonomic and applicable to fully unstructured epidemic protocols without assuming any administration activity, apart from the CA used to solve the identity problem.

6 Conclusions

We suggested a framework for maintaining integrity in a class of unstructured epidemic-style P2P protocols in a fully autonomic fashion (apart from the identity problem). We gave an example implementation of the framework for the case of aggregation. Our illustrative example of network size estimation using this implementation suggested that the framework is potentially applicable even in applications that are very sensitive to misbehavior.

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] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proc. 6t ACM Symp. on Principles of Dist. Comp. (PODC'87)*, pages 1–12, Vancouver, August 1987. ACM.

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

[4] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probablistic broadcast. *ACM Trans. on Comp. Sys.* to appear.

[5] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. From epidemics to distributed computing. *IEEE Computer*. to appear.

[6] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. submitted for publication.

[7] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks, 2004. to appear in Proc. 24th Int. Conf. on Dist. Comp. Sys. (ICDCS 2004).

[8] 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.

[9] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. on Comp. Sys.*, 21(2), May 2003.

[10] 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.