Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays

Håvard Johansen
University of Tromsø
Norway

André Allavena
University of Waterloo
Canada

Robbert van Renesse
Cornell University
USA

ABSTRACT

This paper describes and evaluates *Fireflies*, a scalable protocol for supporting intrusion-tolerant network overlays.[1] While such a protocol cannot distinguish Byzantine nodes from correct nodes in general, *Fireflies* provides correct nodes with a reasonably current view of which nodes are live, as well as a pseudo-random mesh for communication. The amount of data sent by correct nodes grows linearly with the aggregate rate of failures and recoveries, even if provoked by Byzantine nodes. The set of correct nodes form a connected submesh; correct nodes cannot be *eclipsed* by Byzantine nodes. *Fireflies* is deployed and evaluated on PlanetLab.

1. INTRODUCTION

Network overlays provide important routing functionality not supported directly by the Internet. Such functionality includes multicast routing, content-based routing, and resilient routing, as well as combinations thereof. In recent years, Distributed Hash Tables (DHTs) have been proposed to support network overlays. While it is often straightforward to support network overlays on DHTs, this choice can be questioned. DHTs dictate routes that are not optimal [25], and DHTs are hard to secure [30]. As network overlays are starting to be deployed for critical applications, efficiency and security are becoming important attributes.

In this paper we present an alternative support structure called *Fireflies*. *Fireflies* provides each of its members with a complete view of its live peers.[2] A small subset of these peers are marked as *neighbors*. With high probability, the mesh formed by the members and their neighbor links has a diameter logarithmic in the number of live members and connects all the reachable members that are not Byzantine.

An obvious disadvantage of providing members with a view of the full membership, compared to only a partial view as provided by a DHT, is decreased scalability. Memory requirements per member will grow linearly in the number of members. Given the availability of cheap memory this is not necessarily a problem. More alarmingly, the rate of member join and leave events will likely grow linearly with the number of members as well, possibly leading to an unmanageable amount of network bandwidth or latency. Many of these concerns have been addressed previously [2, 14, 15, 28]. We believe *Fireflies* can scale easily to thousands of members and that this is sufficient for many applications.

Providing each member with membership is a form of agreement. Previous works on Byzantine fault-tolerant agreement establish invariants that are impossible to invalidate. Even the most practical of these protocols (*e.g.*, [5, 19]) require several rounds of all members broadcasting state to all other members, and can consequently not scale up to more than perhaps a few dozen members. In order to scale to thousands or more members, we had to come up with a radically different approach. *Fireflies* makes use of epidemic techniques to form a probabilistic agreement, which can only establish invariants that hold with a certain probability. Because invariants never hold for certain, defense against adversaries trying to break agreement can never rest. The main contribution of this paper is this novel approach towards tolerating intrusions. A formal specification and correctness proof has been completed as well and is the subject of a future publication.

We distinguish three *states* of members: *correct*, *stopped*, and *Byzantine*. Correct members execute the protocols described in this paper faithfully. Stopped members are not executing any protocol steps. Byzantine members are not bound to the protocols. For convenience, we also refer to members that are either correct or Byzantine as *live members*. Members can switch between states at any time, which is commonly referred to as *churn*. Informally, *Fireflies* provides its correct members with a membership view that includes all members that have been correct for sufficiently long and excludes all members that have been stopped for sufficiently long.

A group membership protocol that tolerates Byzantine behavior of its members is said to be *intrusion-tolerant*.

[1] This work is supported by the DARPA/IPTO SRS program, the AFRL/Cornell Information Assurance Institute, NSF grant 0430161, and the Research Council of Norway IKT 2010 program.

[2] Fireflies, the beetles, model not only the on/off behavior of members, but like Byzantine members they are also known for their aggressive mimicry in order to dupe and devour related species.

EuroSys'06, April 18–21, 2006, Leuven, Belgium.

There are limitations to group membership, particularly if intrusions are allowed. Correct members may be unreachable and appear stopped to other members. Recently stopped members may not yet have been detected and appear correct. A Byzantine member can disguise itself as a flaky correct member. Nonetheless, intrusion-tolerant network overlay routing protocols may be built using *Fireflies* as a building block. *Fireflies* currently supports a DHT and a multicast protocol, both of which are intrusion-tolerant as well.

We start with a description of related work in Section 2. Section 3 presents an informal specification of *Fireflies*. The membership protocol is the subject of Section 4. Section 5 describes the epidemic protocol that supports the membership protocol. An evaluation of *Fireflies* appears in Section 6. We discuss current applications of *Fireflies* in Section 7. Section 8 concludes.

2. RELATED WORK

The first paper that describes concrete defenses against Byzantine behavior in peer-to-peer (P2P) network overlays is [4]. While this paper addresses the problem of impersonation attacks, many of the problems discussed have to do exclusively with overlay routing table maintenance and message forwarding. In our protocol, the members do not need to route messages, while in [4], the problem of membership is not considered.

Some peer-to-peer storage services like OceanStore [18] and FARSITE [1] use traditional Byzantine consensus protocols among small groups of machines. These groups do not protect the integrity of the peer-to-peer network as a whole, but protect Byzantine failures among replicas of individual files or meta-files. Also, as the system scales up, the likelihood that one of these groups fail as a whole grows, potentially endangering the entire system. Recent work [10] on FARSITE attempts to isolate such groups from one another, at the cost of weakening the system's semantics.

In [12], the authors propose a mechanism called *Link Attestation Groups* to increase the robustness of overlay networks. A link attestation is a certificate stating that a particular participant x trusts that it has a good path to another participant y. Using a link-state protocol, groups of dozens of participants are formed. The authors argue that by providing access to the graph of attestations, applications will be able to build more reliable protocols, but so far none of these have been evaluated.

The problem of impersonation attacks was first considered in [9]. The paper proposes secure identifier generation as a solution, and both our protocol and [4] have embraced this solution.

The problem of intrusion-tolerant membership in P2P protocols is considered in [29]. The *Eclipse attack* is an attack where malicious members isolate correct members by filling the neighbor table of a correct member with addresses of malicious members. The paper suggest thwarting this attack by enforcing bounds on the in- and out-degrees of P2P members.

Many group membership protocols are based on providing members with consensus on membership views. Note that in such systems views may still be stale. Versions that tolerate Byzantine members have been designed and implemented (*e.g.*, [26, 27]), but the overhead of consensus renders such systems unscalable beyond a few dozen members.

In [28], Rodrigues and Blake argue that in many environments multi-hop routing is not cost-effective, and full membership maintenance is both possible and desirable. One-hop peer-to-peer routing protocols that maintain full membership for fail-stop environments are presented in [14, 15]. CONGRESS [2] is a non-P2P solution based on a scalable server hierarchy. Neither tolerates Byzantine behavior.

Most closely related to our work is the SWIM protocol [7]. As in *Fireflies*, SWIM has a separate failure detection protocol and an epidemic dissemination protocol. Unlike *Fireflies*, SWIM's failure detection protocol does not adapt to varying message loss, and SWIM is not tolerant of Byzantine behavior.

The SCAMP protocol [13] is another epidemic-style membership algorithm that, like *Fireflies*, uses a small number of gossip partners per member in order to increase scalability. SCAMP is not intrusion-tolerant, and does not have a failure detection component. Members have to leave the group explicitly by gossiping a message. SCAMP may converge to a non-random graph. CYCLON [32] presents an improvement over SCAMP, maintaining randomness even with high node churn.

There has been a variety of work on intrusion-tolerant epidemic protocols, apparently starting with [21]. These protocols consider the problem of correct members not accepting any malicious updates without using unforgeable signatures, and use a form of voting instead. Perhaps the earliest epidemic membership protocol is reported in [31], while epidemic dissemination was pioneered in the Clearinghouse system [8].

Drum [3] is a DoS-resistant multicast protocol. It uses a combination of gossip techniques, resource bounds for certain operations, and random UDP ports in order to fight DoS attacks, especially those directed against a small subset of the correct members. These techniques are orthogonal to the ones used by *Fireflies*, and can be used to make *Fireflies* less susceptible to DoS attacks.

3. MEMBERSHIP SPECIFICATION

Each member m has a unique identifier $m.id$. A member cannot choose nor modify its identifier. Each member m has a state that is either *correct*, *stopped*, or *Byzantine*.

Each correct member has a *view* $m.view$, which is a subset of all members. Informally, $m_2 \in m_1.view$ means that m_1 believes that m_2 is, at least until recently, not stopped. As well, $m_2 \notin m_1.view$ means that m_1 believes that m_2 is, at least until recently, not live.[3]

A correct member also has a set of *neighbors* $m.neighbors$, which is a subset of its view. The number of neighbors is logarithmic in the size of the view, *i.e.*, $|m.neighbors| = O(\log |m.view|)$. The mesh consisting of correct members and the links to their correct neighbors is connected and logarithmic in diameter, and thus forms a usable routing substrate.

We also want, again informally, that all correct members send a limited amount of data in that Byzantine members cannot cause correct members to send large amounts of data rendering the protocol unscalable. We cannot prevent a Byzantine member from sending large amounts of data.

[3] We do not provide Virtual Synchrony properties such as consensus on views among correct members.

Fireflies does not exactly provide these properties, as it is a probabilistic protocol. For example, it is possible that long time correct members are sometimes evicted from the views of other correct members, and it is possible that long time stopped members are included in the views of correct members. Also, correct members may be partitioned. The *Fireflies* protocol makes such inconsistent states infrequent, with probabilistic guarantees.

4. MEMBERSHIP PROTOCOL

In this section we describe the membership protocol that correct members[4] follow. For now we will assume that members have at their disposal a gossip channel that reliably broadcasts a message to all members within a time Δ. Section 5 will present a protocol that provides such a guarantee with high probability.

The basic idea of the membership protocol is that members monitor one another and use the gossip channel to disseminate *accusations* (failure notices). When a member m_1 receives an accusation for a member m_2, m_1 waits a time period of length 2Δ before removing m_2 from its view. Should m_2 receive, through gossip, an accusation about itself, then m_2 has the opportunity to gossip a *rebuttal* before 2Δ expires. There is an overhead associated with gossip, so we have to prevent Byzantine members from submitting frequent accusations about correct members. This is a complicated issue because correct members may accidentally accuse other correct members due, for example, to transient link failures. Thus not every false accusation is from a Byzantine member.

4.1 Assumptions

While we allow the network to lose messages, we do assume that all correct members can run a ping protocol effectively and apply a gossip-style broadcast protocol that can deliver messages to all correct members in a timely fashion. Details on the ping protocol appear in Section 4.6, while details on the implementation of the gossip protocol are presented in Section 5. While we do not require clocks to be synchronized, we do assume that clock rates among correct members are identical, although rates only need to be "similar" in practice.

Byzantine members have few restrictions. They know the exact state of every other member, and have zero-latency connections between each other. However, they do not have sufficient computational power to break cryptographic building blocks, and in particular they cannot forge public key certificates, or public key signatures of correct or stopped members.

We assume that there is a bounded probability P_{byz} that a live member is Byzantine. Note that this is a stronger condition than a bound on the probability that *any* member is Byzantine. Such a weaker condition would not suffice, as in the case that most non-Byzantine members are stopped, the few remaining correct members could be overwhelmed by Byzantine members. Nonetheless, the assumption that among all live members only a fraction is Byzantine is reasonable, particularly since we do not limit the fraction of stopped members among all members.

We assume that each member m has a unique identifier $m.id$ (*e.g.*, tax identifiers, passport numbers, etc.), and that a shared Certificate Authority (CA) does a thorough background check on each potential member before handing the member m a private key $m.priv$ and a corresponding public key certificate $m.cert$ that binds the member's identifier and network address to its public key.[5] Correct and stopped members never reveal their private key.

note	most recently known note of the peer
accusations	accusations, at most one per ring
nPings	#pings sent since last "pong" response
avgLoss	smoothed average of #pings lost + 1

Table 1: Fields in an *info* structure, one for each peer. The last two entries are used only for successor peers.

We also assume that trivial Denial-of-Service attacks by flooding can be detected and suppressed (see [3] for how this might be accomplished).

4.2 Data Structures

The members are organized on $2t + 1$ rings, which are circular address spaces (the value t is discussed below). Each member m attains a position on each ring by evaluating a secure collision-resistant hash function $\mathcal{H}$:

$$m.pos[ring] = \mathcal{H}(m.id \mid\mid ring)$$

(where '$||$' is the concatenation operation). The ordering of members on each ring is different (with high probability) as a result. Member m ranks members on each ring clockwise, with itself being 0, its first successor being rank 1, and so on. The basic idea of the protocol is that each member m_1 monitors, on each ring, the lowest ranked successor m_2 that m_1 believes to be live.

The members use two data structures that are gossiped: *notes* and *accusations*. A member creates notes in order to notify and convince the other members that it is live. A note is a tuple (*cert, epoch, enabled*), signed using the private key of the member. Here *cert* is the public key certificate of the member, *epoch* a number used to order its notes, and *enabled* a bitmap with $2t + 1$ bits, controlling which of the $2t + 1$ predecessors are allowed to issue accusations against the member.

If a member m_1 suspects a successor m_2 on a particular ring of having stopped, then m_1 accuses the note of m_2 last known to m_1 by creating an accusation. An accusation is a tuple (*note, ring, accuser*), signed by m_1, where *note* is the note of m_2, *ring* is the ring on which m_2 is a successor of m_1, and *accuser* is the identifier of m_1. A requirement on the accusation is that *note.enabled*[*ring*] is set. Thus a member can use the *enabled* bitmap in the notes it generates to restrict which predecessors can accuse the member, an ability that we will use to defeat repeated false accusations by Byzantine members.

The value t governs the number of Byzantine predecessors a member can tolerate. We choose t so that the probability of a member having more than t out of $2t+1$ live predecessors being Byzantine is small (see Section 4.5). Members set exactly $t+1$ bits in their notes' *enabled* bitmaps. If a member m has at most t Byzantine predecessors, then m can disable all Byzantine predecessors that make repeated false

[4] We shall omit the adjective "correct" where obvious.

[5] Note that the CA can provide access control as well.

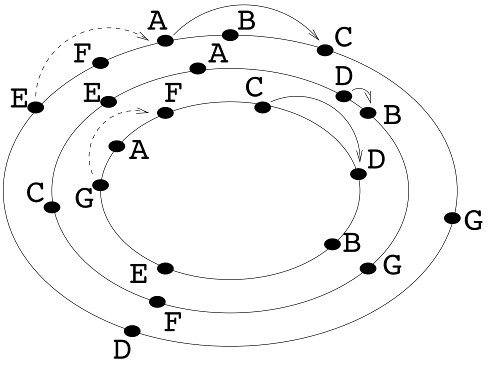

Figure 1: 3 rings with 7 members A through G. Valid accusations are shown with solid arrows, while invalid ones are shown in dashed arrows.

accusations while m is live. If, by mistake, m disables only correct predecessors and leaves t Byzantine predecessors enabled, then there is still at least one correct predecessor that can accuse m should m fail.

Each member m maintains a mapping $m.info$ of peer identifiers to information about these peers. ($m_1.info(m_2) = \perp$ means that m_1 does not have any information about m_2.) The fields in the *info* structure are listed in Table 1. The *Fireflies* protocol strives to ensure that the set of accusations is empty for a correct member and non-empty for a stopped member.

4.3 Valid Accusations

As we have not bounded the probability that a member is stopped, all predecessors of a member may be stopped with non-negligible probability. In order to allow such members to be accused in case they fail as well, a member must be able not only to accuse its immediate successor, but must also be able to make accusations skipping over stopped successors. Doing so may allow a Byzantine member to accuse any of its successors simply by claiming that it believes that the more immediate successors are all stopped.

In order to counter such attacks, we create rules that govern which accusations are considered *valid*. Informally, m_1 only allows the *highest ranked* live member to make *valid* accusations of m_2, and only on those rings that are enabled by m_2. Validity is defined recursively. Member m_1 considers an accusation for m_2 valid *iff*

- the accusation is correctly signed; *and*
- the note contained in the accusation corresponds to $m_1.info(m_2).note$; *and*
- the ring in the accusation is enabled in the note's *enabled* bitmap; *and*
- m_1 holds valid accusations for all members it ranks (on the given ring) between the accuser and m_2 itself, if any.

In Figure 1, we show a schematic depiction of how one of the members observes a group with 7 members, A through G. In this case, $t = 1$, and for simplicity we ignore ring deactivation. An accusation of B by D on the middle ring is a valid accusation (assuming the accusation refers to the note of B and is correctly signed by D) because there are no nodes in between D and B. This accusation is valid even though the accuser D is validly accused by C. The accusation by A of C on the outer ring is valid because there is a valid accusation against B, the node in between A and C. The accusation of A by E is invalid as there is no valid accusation of F.

A *Fireflies* member only maintains accusations that it considers valid, and associates a timer with each valid accusation that is set to 2Δ when the member first learns of the accusation. A valid accusation may depend on other valid accusations, and in case such a dependency changes, the timer needs to be reset. When the timer expires, the accusation leads to the removal of the accusee from the member's view.

4.4 Protocol Steps

Each correct member runs the same protocol. There are four events that trigger protocol transitions.

m_1 ***receives a note for a peer member*** m_2. If m_1 has a note for m_2 that is as recent as the one that arrived, then m_1 ignores it. Otherwise m_1 updates its note for m_2, removes any accusations that it has for m_2, cancels m_2's view removal timer if any, and includes m_2 in its view. In addition, the removal of accusations for m_2 may invalidate accusations that m_1 holds for other members. These accusations are removed as well.

m_1 ***suspects*** m_2. On each ring, m_1 monitors the lowest ranked successor m_2 for which m_1 does not hold valid accusations (unless m_2 has disabled the ring, in which case m_1 does not monitor anybody on that ring). Should m_1 suspect that m_2 has stopped, then it creates an accusation of m_2 that is subsequently gossiped to the other members.

m_1 ***receives an accusation for*** m_2. If m_1 does not consider the accusation valid, then m_1 ignores it. If $m_2 = m_1$, then m_1 replaces its note with a new one to act as a rebuttal, which is subsequently gossiped to the other members. If $m_2 \neq m_1$ and m_1 already has an accusation for m_2 on the same ring as the new accusation, then m_1 replaces its accusation only if the new one is from a higher ranked accuser.

At m_1***, the timer of an accusation of*** m_2 ***expires.*** m_1 removes m_2 from its view.

4.5 Calculating t

We now turn to calculating a suitable value for t. If there are too few rings, correct members may not be able to fight Byzantine behavior. However, more rings result in higher overhead. Since we cannot give deterministic bounds on the number of Byzantine predecessors, we use a probabilistic approach. We want the minimum value of t so that the probability of a member having more than t out of $2t + 1$ live predecessors being Byzantine is smaller than ε:

$$\min_t : B(t; 2t + 1; 1 - P_{byz}) < \varepsilon$$

where $B(x; n; p)$ is a cumulative binomial distribution. We may want to make ε smaller for larger N so that the expected number of members for which this condition is violated does

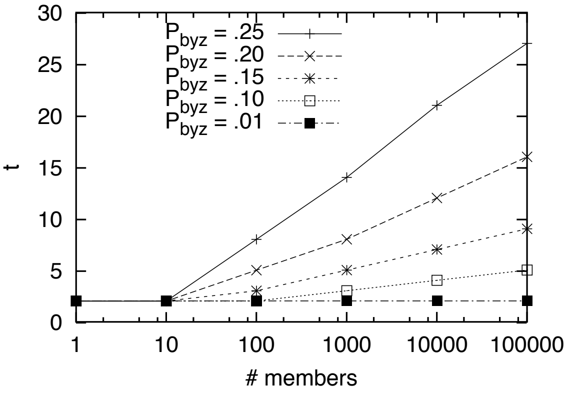

Figure 2: The magnitude of t for various N and P_{byz}.

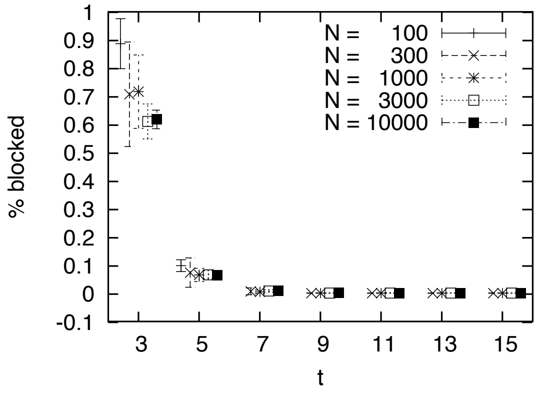

Figure 3: The likelihood of blocked accusations (99% confidence intervals) as a function of t for various N.

not grow linearly with N. For example, if we set $\varepsilon = 1/N$, then the expected number of such unfortunate members is 1, independent of N altogether. In Figure 2 we used the normal approximation to the binomial distribution to solve this equation for various N and P_{byz}, using $\varepsilon = 1/N$.

The protocol described above assumes a static t, implying that a maximum membership should be anticipated and enforced. It is, however, possible for a member to specify a number of rings that depends on the size of its view. Care should be taken to deal with Byzantine members specifying an *enabled* map in their note with a very large number of rings in order to try to consume all memory of correct members. The resulting extensions to the protocol are trivial and are not discussed in this paper.

Even if a stopped member m_1 has fewer than $t+1$ Byzantine live predecessors, it is possible that an enabled correct live predecessor m_2 cannot accuse m_1. Consider the stopped members between m_2 and m_1 on the corresponding ring and call them $s_1, \ldots, s_m$. If the ring is enabled for all s_i, then m_2 can accuse all s_i and thus m_1. Assume there is a member s_j that disabled the ring. Then m_2 cannot accuse m_1 until m_2 has received a valid accusation for s_j on a different ring. We say that the accusation of m_1 is blocked by s_j. Unfortunately, m_2 may never receive an accusation for s_j. Member s_j may have fewer than $t+1$ correct live predecessors, all of which being disabled. It may even be that the accusation of s_j is blocked by m_1, thus creating a loop preventing both m_1 and s_j of being accused.

Fortunately, it can be shown that the probability of blocked accusations is negligible. Formally, the probability of a stopped node not being accused is upper bounded by:

$$P \approx P_{\text{byz}} + (1 - P_{\text{byz}})P_{\text{stopped}}^{\ln n})^{t+1}$$

While the proof is outside the scope of this paper, a simulation bears this out as well. In Figure 3, we show the results of the following simulation. Initially, all members are correct and are present in all the views. At time $T = 0$, 75% of the members stop, 20% of the remaining members become Byzantine. In order to generate a worst-case scenario, the correct members disable as many correct predecessors as possible, and the Byzantine members do not emit any accusations. The graph shows that even for large N, a relatively small value for t makes blocked accusations highly unlikely.

4.6 Pinging

Members use pinging to detect failures. Essentially, a member m_1 monitors a member m_2 by sending "ping" messages to m_2 at regular intervals. Member m_2 returns a "pong" message for each ping that it receives. If m_1 does not receive pong messages from m_2 for more than some time period, m_1 considers m_2 stopped and issues an accusation.

A tricky detail is determining how long to wait before issuing an accusation. Using a static global timeout is not a good choice, as this will not scale well and can cause correct members to accuse other correct members more often than necessary. In particular, the timeout period has to adapt to the message loss characteristics between monitor and monitoree.

In Figure 4, we present a simple but effective protocol based on unreliable message passing. The members individually estimate the probability of message loss. We model pinging as a negative binomial experiment with parameters $r = 1$ and the probability of success p. Then the expected number of consecutively failed ping exchanges is $E(X) = (1-p)/p$. It follows that $p = 1/(E(X)+1)$. We estimate $E(X)+1$ by *avgLoss* using exponential smoothing. (The smoothing factor α is set to .999 in our current implementation.)

Having p, we can calculate the probability of making τ mistakes: $(1-p)^{\tau}$. We want this probability to be smaller than a configured constant $P_{mistake}$. It follows that $\tau > \log(P_{mistake})/\log(1-p)$.

If message loss is very low, τ would be set unrealistically low, and if $p = 1$, the expression would be undefined. We address both problems by having a minimum threshold τ_{min}. It follows that p should be set no higher than $1 - P_{mistake}^{1/\tau_{min}}$.

Byzantine members could potentially prevent detection of stopped members by forging pong messages. This is prevented by having each ping message contain a nonce that has to be signed by the monitoree and returned in the corresponding pong message. This strategy prevents both forging of pong messages and replay attacks, and this is why we chose pinging over a heartbeat protocol.

Byzantine members can, however, generate a modest amount of overhead on the system by not responding to ping messages from correct members, and rebutting the ensuing accusations. Such "nuisance attacks" are easily identifiable,

```
on time to ping $m$ on ring $r$:
    $p = \min(1/\mathit{info}(m).\mathit{avgLoss}, 1 - P_{\mathit{mistake}}^{1/\tau_{\mathit{min}}})$;    // est. probability of successful ping exchange
    $\tau = \log(P_{\mathit{mistake}})/\log(1-p)$;    // calculate threshold
    if $(\mathit{info}(m).\mathit{nPing} > \tau)$
        $\mathit{info}(m).\mathit{accusation} =$ new Accusation$(\mathit{info}(m).\mathit{note}, r, \mathit{self.id})$;
    else
        send$(m,$ new Ping$(\mathit{self.id}))$;
        $\mathit{info}(m).\mathit{nPing}++$;

on receive Pong$(m)$:
    $\mathit{info}(m).\mathit{avgLoss} = (\alpha * \mathit{info}(m).\mathit{avgLoss} + (1-\alpha) \cdot (\mathit{info}(m).\mathit{nPing}))$;
    $\mathit{info}(m).\mathit{nPing} = 0$;
```

Figure 4: Pinging protocol. $P_{mistake}$ (probability of making a mistake), τ_{min} (minimum threshold), and α (smoothing factor), are configuration constants.

and such members can be manually removed by revoking their public key certificates.[6]

5. GOSSIP PROTOCOL

A gossip protocol is a simple group communication protocol whereby each member periodically picks a random member from its view and exchanges state information. Such protocols are known to be highly robust, as they are essentially flooding protocols. But unlike flooding protocols, they are efficient with probabilistic bounds on delivery latency [17]. In our particular situation, we have to concern ourselves with Byzantine members.

Say we have two members m_1 and m_2 exchanging notes and accusations. All notes and accusations are signed, and because we assume that Byzantine members cannot break the cryptographic building blocks, we do not have to worry about impersonation attacks [9]. We have also assumed that trivial Denial-of-Service attacks can be detected and suppressed.

Byzantine members can still attack the gossip protocol in the following two ways. In order to slow down dissemination, they can neglect to forward recent updates. This slow-down can be incorporated in the calculation of Δ. Byzantine members can also pretend that they have no information, causing correct members to transmit their entire state to them and thus causing unnecessary load on the correct members and on the network. In order to reduce the opportunities of Byzantine members to launch this attack, we will consider gossip protocols in which each member can only gossip with a small subset of the membership.

5.1 Partial Membership Gossip

Kermarrec et al. [17] shows that it is possible to build effective gossip protocols if each member only has a small set of uniformly chosen members it gossips with. Each member m selects k gossip *neighbors* from its view uniformly at random where k be large enough to create a connected graph of correct nodes. A classic result of Erdös and Rényi [11] shows that in a graph of n nodes, if the probability of two nodes being connected is $p_n = (\log n + c + o(1))/n$, then the probability of the graph being connected goes to $\exp(-\exp(-c))$.

The number of correct nodes in the view, n, is expected to be at least $(1 - P_{byz}) \cdot N$, where P_{byz} is a configured upper bound on the probability that a live node is Byzantine, and N is the total of the correct and the Byzantine nodes. Then the probability that one node is connected to another is $1 - (1 - 1/N)^k \approx k/N$. Thus $p_n \approx 2k/N$.[7]

In order for the correct nodes to be connected with probability φ, we obtain

$$k \geq \frac{N}{2n} \cdot \left(\log \frac{-n}{\log \varphi} + o(1)\right)$$

Next we determine the resulting Δ, the time to disseminate an update in this random graph. To better preserve resources, each member does not update all its k gossip neighbors in each round, but instead selects one neighbor for each round in a round-robin fashion. In order to simplify calculations, we will assume conservatively that it takes k rounds to update all gossip neighbors, and thus the dissemination runs a factor k slower than if all neighbors were updated in each round. If d_n is the diameter of the graph of correct nodes, then the expected amount of time to disseminate an update reliably among the correct nodes is therefore $\Delta = k * d_n$.

An asymptotic value for d_n can be determined. A recent result of Chung and Lu [5] shows that if $np_n \to \infty$ (which in our case it does), then the expected diameter of our graph is $(1 + o(1)) \frac{\log n}{\log np_n}$. Unfortunately, it does not provide the constants needed to tune the gossip protocol.

In order to find suitable constants, we ran simulation experiments with N ranging from 16 to 16,384 for varying P_{byz} and with k chosen as above (ignoring the $o(1)$ term), to determine if the resulting graphs of correct nodes are indeed connected and to obtain values for Δ. We ran each experiment 100 times. We encountered no disconnected graphs in any of our 3000 experiments. In Figure 5 we report the maximum Δ we observed for each N and P_{byz}.

5.2 Pseudo-Random Mesh

The analysis of the gossip protocol above tacitly ignores the possibility of a Byzantine member selecting more than k neighbors in order to increase the overall load on the correct

[6] Not discussed in this paper, a CRL can be reliable disseminated using the gossip protocol.

[7] We assume here that every correct node can connect to every other correct node. This assumption can be relaxed, but p_n has to be adjusted accordingly.

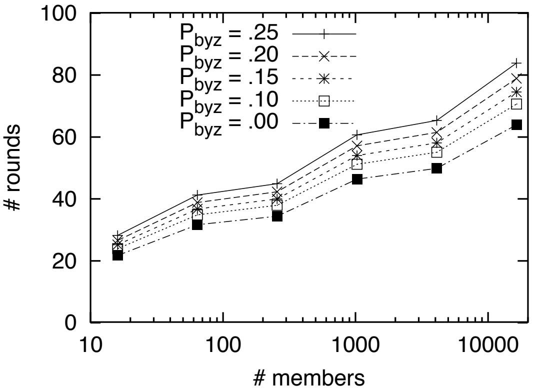

Figure 5: The maximum of 100 simulation experiments of the number of rounds required to disseminate an update to all correct members as a function of the total number of live members for various P_{byz}. In these experiments, $\varphi = .99999$.

members. Also, Byzantine members could "gang up" on a small set of correct members, overwhelming them with gossip load [3]. In order to fight such attacks, we introduce a rule that determines who can gossip with whom. We use the same technique that we used in Section 4.2 to determine who monitors whom, except that we use k rings.

On each ring, a member initiates gossip only with the first successor in its view. For ring i, a member m sets up a secure mutually authenticated connection with the successor m_i using their private and public keys. Member m then sends $m.note$ and i to m_i, so that m_i can add m to its view if necessary and possible (existing accusations of $m.note$ may prevent this). Member m_i checks that $1 \leq i \leq k$ and that m_i is m's successor in m_i's view.

One complication is that even when m and m_i are both correct, they may have different views. In particular, m_i may know a "better" gossip neighbor n_i for m that is not in m's view. If such is the case, m_i sends n_i's note to m. Should m have plausible accusations for n_i, then it returns those to m_i and terminates the attempt to gossip. If no such accusations exist, then m was unaware of n_i. In that case m adds n_i to its view and tries to gossip with n_i instead.

If at any point in time m should determine a better gossip neighbor for ring i than m_i, then m terminates the existing connection. Note that newly joining and recovering members should gossip with at least $t+1$ different members before they can be reasonably certain that they will be integrated into the "true" membership (as opposed to a fake membership created by Byzantine members [29]).

The neighbors thus chosen form a convenient low-diameter mesh that connects the correct members. *Fireflies* exposes the set of neighbors in its API, so that applications can gossip about information other than membership or use the mesh for multicast dissemination (as discussed in Section 7.2).

6. EVALUATION

With some moderate assumptions, such as having the graph of correct nodes be connected, we can formally prove that all correct nodes will be included in the view of correct members and that stopped nodes will almost certainly be removed from the views. In this section, we present the results of experiments that validate the performance of *Fireflies*.

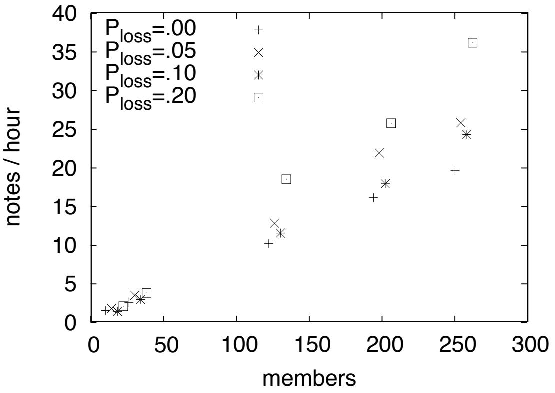

Figure 6: Number of notes sent per correct member per hour as a function of the number of members for various P_{loss}.

Our prototype implementation is written in Python. The code can run both on a simulated network, or on a real network. In all experiments below, we used $t = 12$, resulting in 25 rings. We used $k = 8$ (corresponding to $\varphi = .999$), meaning that each member had about 16 neighbors for gossiping. Each member gossiped once every 3.75 seconds on average (resulting in one gossip per minute with every neighbor), although the *Fireflies* code randomizes the intervals at which a member gossips in order to prevent synchronized "waves" of gossip. The probabilistic upper bound on the time for gossip to spread, Δ, was chosen to be 5 minutes. Members pinged each of their monitorees every 30 seconds.

We will first describe some experiments performed on the simulated network, and then present experience gained from a deployment of the code on PlanetLab [24].

6.1 Simulation

In order to trigger frequent mistaken failure detections, we set $P_{mistake}$ to .001. Both the MTTF (Mean Time To Failure) and MTTR (Mean Time To Recovery) of the correct members was set to 6 hours. The intervals between stopping and going were exponentially distributed. The total number of members N ranged from 16 to 256. Each experiment ran for six simulation hours, and each experiment was run at least eight times. The graphs below show averages and 95% confidence intervals, except where the intervals were too small.

First we looked at the overhead in the absence of Byzantine members. Figure 6 shows the average number of notes sent (created or forwarded) per correct member per hour as a function of the number of members for various P_{loss}, the probability of message loss. In each case, we see a clear linear trend as a function of the number of members, as expected. Without loss, the expected number of notes is $N/12$, as on average there is one recovery every 12 hours. With loss, mistakes are made, leading to an increase in the number of notes generated. Due to adaptive pinging this is almost, but not completely, independent of P_{loss}. For example, for 5% loss the rate of notes sent is higher than for 10% loss. The

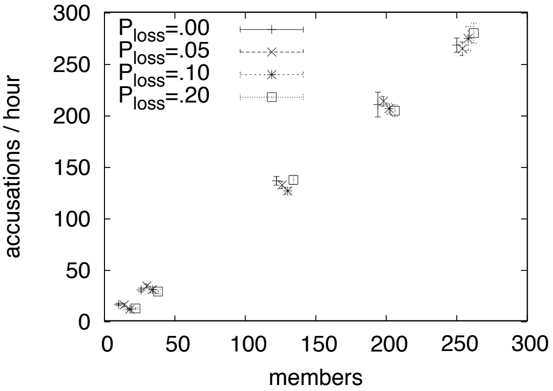

Figure 7: Number of accusations sent per correct member per hour as a function of the number of members for various P_{loss}.

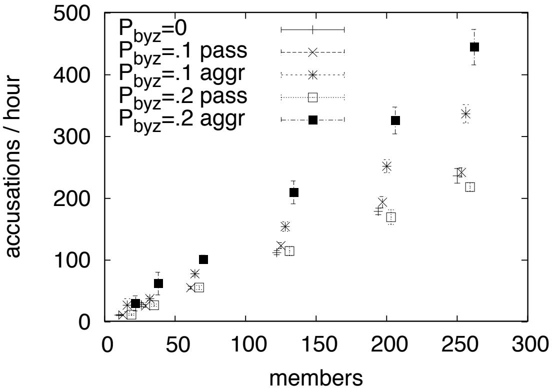

Figure 8: Number of accusations sent per correct member for various P_{byz} and styles of attack.

differences are due to rounding of τ, the pinging threshold above which a failure is assumed. The effect is that significantly fewer mistakes may be made than specified with $P_{mistake}$.

Figure 7 shows the average number of accusations sent per correct member per hour as a function of the number of members for various P_{loss}. Because accusations can be made on multiple rings, these rates are higher than for notes, but they do not depend much on the message loss probability. Partially this is due to the success of our adaptive pinging protocol, but it is also due to the dissemination of a mistaken accusation being squelched by the ensuing rebuttal.

Next we introduced Byzantine members. We varied P_{byz} from 0 to 0.2. We looked at two types of attacks. One is an "aggressive attack," where Byzantine members accused other members at any opportunity, and refrain from forwarding notes (rebuttals) from these members. The other is a "passive attack," where Byzantine members never accuse stopped members, and do not forward accusations of stopped members, in an effort to make stopped members appear correct. Neither style of attack was successful in any of our tests. Figure 8 shows the average number of accusations sent by correct members for various P_{byz} and styles of attack. The attacks had a moderate effect on traffic generated, but the amount stayed well within a factor of two of the case in which there were no Byzantine attacks.

6.2 Experience on PlanetLab

PlanetLab [24] is a world-wide collection of over 600 machines at over 275 sites connected to the Internet in 30 countries. PlanetLab can be used to test new scalable protocols and to deploy novel distributed services. We first deployed *Fireflies* on PlanetLab in early February 2005, and found the experience useful to find pragmatic problems and test solutions. However, the overheads we measured, some of which are presented below, are specific to PlanetLab only.

In this section we describe our experiments and indicate where further work on *Fireflies* is needed. Our prototype implementation uses TCP for gossip but UDP for pinging as our adaptive pinging protocol needs to determine when messages get lost.

While the majority of PlanetLab nodes tend to be fairly well-connected, some of the nodes are very heavily loaded, to the point of making some of these nodes effectively unreachable. Other nodes are only partially reachable, either due to configuration problems or due to heavy packet loss. For example, some nodes cannot send or receive UDP messages. This has two consequences for *Fireflies*. First, a node that cannot receive UDP packets will accuse its successors, even if they are correct. This is not a problem, as these successors will use their *enabled* bitmaps to disable the corresponding rings. Second, such a node will be accused by its predecessors. The accusations are effectively rebutted, and this accused member is not removed from the views as long as it is able to gossip new notes (using TCP). Unfortunately, the member cannot disable all rings (which would have its own problems), leading to a continuous background gossip of accusations and rebuttals. Besides a communication overhead on the network, these superfluous messages increase the load on machines.

6.3 Measurement Study

We now describe the results of one of our recent PlanetLab experiments, run February 24, 2006. The parameters were set the same as in the simulations above, except that $P_{mistake}$ was set to a more sensible .00001. For signatures we used SHA1 and RSA with 1024 bit keys. This resulted in a public key certificate of 163 bytes, a note of 49 bytes, and an accusation of 52 bytes.

Each time a member m_1 gossips with member m_2, they first exchange a collision-resistant hash of their sets of notes and accusations. If they are different, a full state reconciliation is done using the algorithm described in [22].

The experiment started with *Fireflies* agents running on 280 PlanetLab machines. During the experiment about 10 of those machines became unresponsive and fell out of the experiment. 10% of the *Fireflies* agents were configured to mount false accusations aggressively (chosen randomly), while another 10% where configured to mount a passive attack, not accusing and not forwarding accusations for failed members. At 7pm, we terminated 80 of the *Fireflies* agents, chosen randomly. At 7am, we restarted the agents.

Each agents writes a checkpoint to its log every 10 seconds. If an agent has not written a checkpoint to its log during a 2 minutes period of the experiment it is considered

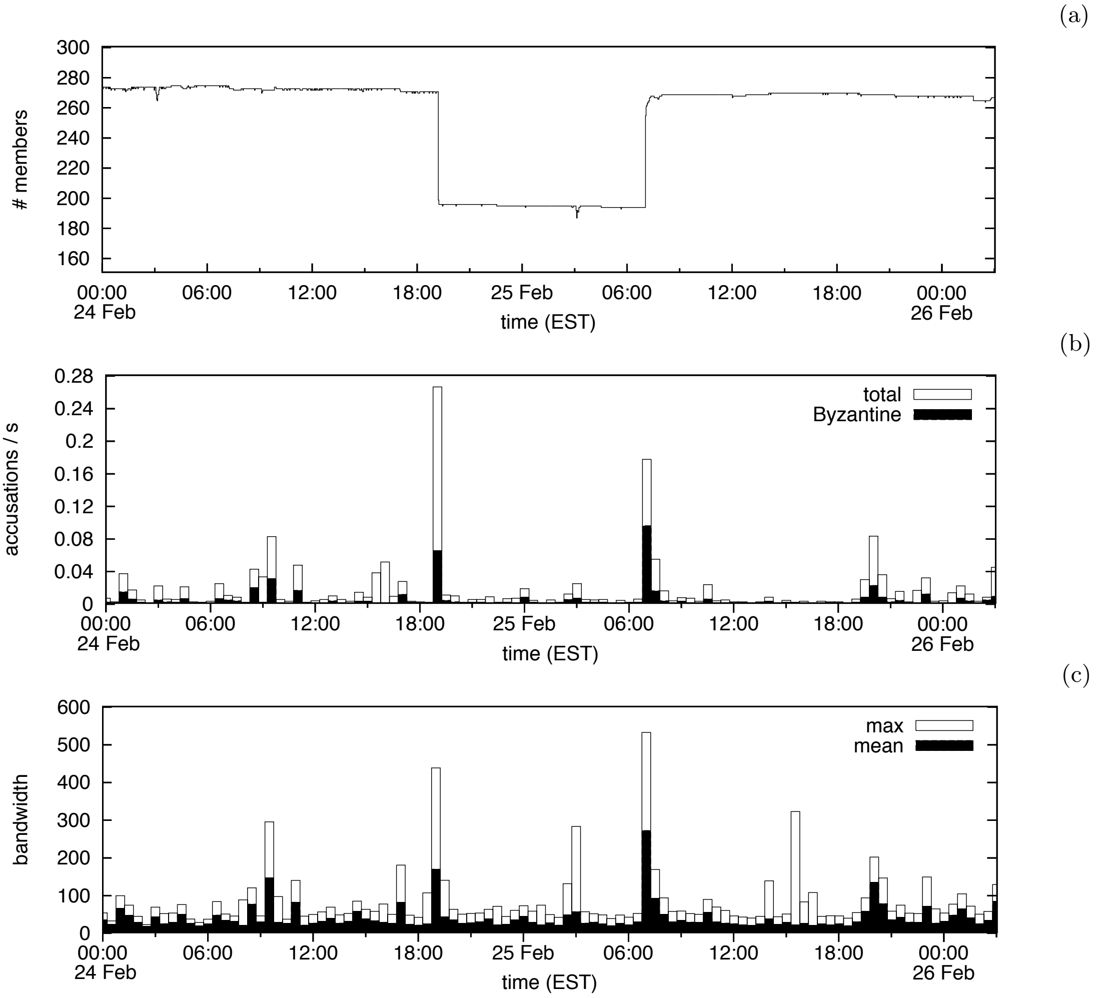

Figure 9: PlanetLab results: (a) size of the membership; (b) # accusations / second; (c) # bytes written per member per second.

failed in that period. The number of live agents in each time period is shown in Figure 9(a). We can observe that there is a fair bit of membership churn not under our control.

Figure 9(b) shows the aggregate rate of accusations created per second, divided into total and false accusations from Byzantine members. Two peaks can be clearly distinguished: when the agents are terminated, and when they are restarted. The first peak is obvious: the terminated agents are accused. The second peak is caused by several rejoining agents becoming unresponsive due to some heavily loaded PlanetLab machines's inability to accommodate the extra CPU and network overhead incurred when reintegrating the recovering agents into the membership. Accidental accusations from correct members gives aggressive Byzantine members opportunities to issue new false accusations, adding to the temporary flurry of communication.

Figure 9(c) shows the mean and maximum number of bytes sent per correct member per second. The bandwidth follows the rate of accusations, but is never above 500 bytes per member per second. The various peaks are caused by real failures. We have witnessed in our experiments various unexpected behavior on PlanetLab nodes. Sometimes the local file system on a node disappears or runs out of disk space, preventing logs to be written. Sometimes nodes are wrongly configured with an unroutable IP address. Sometimes nodes become unaccessible due to network outages, CPU overload, or, rarely, actual crashes. There have even been bugs in *Fireflies* agents causing them to crash or behave erratically. But the *Fireflies* infrastructure as a whole has survived all of these problems.

7. CURRENT APPLICATIONS

There are clear limitations to what *Fireflies* can offer. Byzantine members can disguise themselves as correct members by executing the protocol, or as stopped members by not executing at all, and so a correct member cannot determine which members are Byzantine unless they reveal themselves by sending messages that prove that they are not following the protocol. Also, views trail membership changes, and may be stale at any time. The question then is how *Fireflies* can be useful.

In this section, we provide examples of using *Fireflies* for building intrusion-tolerant network overlays. In particular, we show how *Fireflies* is used to support a Distributed Hash Table and a multicast protocol.

7.1 DHT

DHTs that are intrusion-tolerant are increasingly necessary. For example, the P6P overlay protocol is an IPv6 tunneling technology built over a DHT, and requires that links between correct members are fair [33]. An intrusion-tolerant DHT can be trivially implemented on *Fireflies*, simply by routing messages for an object identifier to the member in the view with the closest member identifier (assuming object and member identifiers are chosen from the same identifier space). Such an implementation is called a *One Hop DHT* (OHDHT),[8] as messages are not routed through intermediate members.

Other DHTs provide its members with only a partial view of the membership in order to increase scalability, and messages sent between members often take multiple hops. In a OHDHT, messages are less likely to get lost or intercepted along the way and encounter lower latency.

7.2 Multicast

A more interesting use of *Fireflies* is to build an intrusion-tolerant multicast protocol. For example, the Vigilante worm containment system [6] assumes a hypothetical intrusion-tolerant multicast primitive. Our protocol is heavily based on Chainsaw [23], which floods each message on the neighbor mesh. Flooding is done efficiently: when a member receives a large message, it notifies its neighbors only of the message identifier. Each member collects such notifications from its neighbors and requests the message from one. If no response is received within a short period of time, another neighbor is selected (see below). Measurements on Chainsaw have shown that this protocol is as efficient as the best multicast protocols based on DHTs [23], and the measurements of our version of the protocol support this as well.

Because the neighbor mesh connects all correct members, a message from a correct member is guaranteed to be delivered to all correct members. In order to prevent forging and prevent forwarding of illicit traffic, members sign messages and check signatures before accepting received messages. Correct members prefer uploading messages to neighbors from which they recently received a message. This strategy has two positive effects. First, it avoids using links to Byzantine members that are not forwarding messages. Second, it discourages freeloading. A paper that describes and evaluates our multicast protocol in the presence of Byzantine members and free-loaders is forthcoming.

As an alternative to using a pseudo-random mesh, *Fireflies*' complete membership information could be used to build Harary graphs [16], which can tolerate a fraction of Byzantine nodes and can thus form the basis of a secure broadcast protocol [20].

8. CONCLUSION

We presented *Fireflies*, a weakly-consistent, scalable protocol that supports network overlays and tolerates Byzantine members with high probability. *Fireflies* may be used to support an intrusion-tolerant Distributed Hash Table, or for building intrusion-tolerant overlay routing networks, or simply to organize computer resources in, say, a wide-area computational or storage grid.

Fireflies consists of three subprotocols. First, an adaptive pinging protocol makes the probability of a mistaken failure detection independent of message loss. Second, an intrusion-tolerant gossip protocol disseminates information between correct members within a probabilistic time bound. Third, the membership protocol uses accusations and rebuttals to implement the membership information that *Fireflies* provides, and which in turn induces a pseudo-random mesh that can be used for overlay routing.

Our evaluation shows that, at least for moderately sized memberships, *Fireflies* performs well. As the rate of failures and recoveries tends to grow linearly with the size of membership, the amount of information received per member has to grow at least linearly as well. We show that the amount sent (point-to-point) per correct member grows linear with the churn rate, almost independent of the behavior of Byzantine members (with a relative membership of up to 20%). The observed overheads for memberships up to about 280 members are low, and we believe that *Fireflies* will be able to scale up to another order of magnitude without difficulty.

Availability

The code (including the simulation code) is available on SourceForge (http://sourceforge.net/projects/fireflies).

Acknowledgements

The paper was significantly improved using suggestions from the anonymous Eurosys reviewers, from Dag Johansen, and from our shepherd, Christof Fetzer.

[8] Not to be confused with an $O(1)$ hop DHT, although OHDHTs are a member of that class.

9. REFERENCES

[1] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, and R.P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proc. of the 5th Symp. on Operating Systems Design and Implementation*, Boston, MA, December 2002. USENIX.

[2] T. Anker, G.V. Chockler, D. Dolev, and I. Keidar. Scalable group membership services for novel applications. In M. Mavronicolas, M. Merritt, and N. Shavit, editors, *Proc. of the Workshop on Networks in Distributed Computing*, pages 23–42. DIMACS, 1998.

[3] G. Badishi, I. Keidar, and A. Sasson. Exposing and eliminating vulnerabilities to Denial of Service attacks in secure gossip-based multicast. In *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, pages 201–210, June – July 2004.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D.S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. of the 5th Usenix Symposium on Operation System Design and Implementation (OSDI)*, Boston, MA, December 2002.

[5] F. Chung and L. Lu. The diameter of random sparse graphs. *Advances in Applied Math*, 26(4):257–279, May 2001.

[6] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante:

End-to-end containment of Internet worms. In *Proc. of the 20th ACM Symp. on Operating Systems Principles*, Brighton, UK, October 2005.

[7] A. Das, I. Gupta, and A. Motivala. SWIM: Scalable Weakly-consistent Infection-style process group Membership. In *Proc. of the Int. Conf. on Dependable Systems and Networks DSN O2*, pages 303–312, Washington, DC, June 2002.

[8] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. of the 6th ACM Symp. on Principles of Distributed Computing*, pages 1–12, Vancouver, BC, August 1987.

[9] J. Douceur. The Sybil attack. In *Proc. of the 1st Int. Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002.

[10] J.R. Douceur and J. Howell. Byzantine fault isolation in the Farsite distributed file system. In *Proc. of the 5th Int. Workshop on Peer-To-Peer Systems*, Santa Barbara, CA, February 2006.

[11] P. Erdös and A. Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutato Int. Közl*, 5(17):17–61, 1960.

[12] M.J. Freedman, I. Stoica, D. Mazières, and S. Shenker. Group therapy for systems: Using link attestations to manage failures. In *Proc. of the 5th Int. Workshop on Peer-To-Peer Systems*, Santa Barbara, CA, February 2006.

[13] A.J. Ganesh, A.-M. Kermarrec, and L. Massoulié. SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In *Proc. of the 3rd International Workshop on Networked Group Communication*, London, UK, November 2001.

[14] A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Proc. of the 9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*, pages 7–12, Lihue, HI, May 2003.

[15] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *Proc. of the 1st Symp. on Networked Systems Design and Implementation*, San Francisco, CA, March 2004.

[16] F. Harary. The maximum connectivity of a graph. In *Proc. of the National Academy of Sciences*, volume 48, pages 1142–1146, 1962.

[17] A.-M. Kermarrec, L. Massoulié, and A.J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3), March 2003.

[18] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM, November 2000.

[19] C.S. Lewis and L. Saia. Scalable byzantine agreement. Technical report, CS Dept., University of New Mexico, 2004.

[20] M.-J. Lin, K. Marzullo, and S. Masini. Gossip versus deterministically constrained flooding on small networks. In *Proc. of the 14th Int. Conf. on Distributed Computing*, volume 1914 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2000.

[21] D. Malkhi, Y. Mansour, and M.K. Reiter. On diffusing updates in a Byzantine environment. In *Symposium on Reliable Distributed Systems*, pages 134–143, Lausanne, Switzerland, October 1999.

[22] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory*, 49(9):2212–2218, September 2003.

[23] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A.E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proc. of the 4th Int. Workshop on Peer-To-Peer Systems*, Ithaca, NY, February 2005.

[24] L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. In *Third Workshop on Hot Topics in Networking (HotNets-III)*, San Diego, CA, November 2004.

[25] P. Pietzuch, J. Shneidman, J. Ledlie, M. Welsh, M. Seltzer, and M. Roussopoulos. Evaluating DHT-based service placement for stream-based overlays. In *Proc. of the 4th Int. Workshop on Peer-To-Peer Systems*, Ithaca, NY, February 2005.

[26] K. Potter Kihlstrom, L.E. Moser, and P.M. Melliar-Smith. The SecureRing protocols for securing group communication. In *Proc. of the the 31st Hawaii Int. Conf. on System Sciences*, volume 3, pages 317–326, Kona, HI, January 1998.

[27] M.K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42, January 1996.

[28] R. Rodrigues and C. Blake. When multi-hop peer-to-peer routing matters. In *Proc. of the 3rd Int. Workshop on Peer-to-Peer Systems*, La Jolla, CA, February 2004.

[29] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against Eclipse attacks on overlay networks. In *Proc. of the 11th European SIGOPS Workshop*, Leuven, Belgium, September 2004. ACM.

[30] E. Sit and R.T. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proc. of the 1st Int. Workshop on Peer-To-Peer Systems*, Cambridge, MA, March 2002.

[31] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proc. of Middleware'98*, pages 55–70, The Lake District, UK, September 1998. IFIP.

[32] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2), June 2005. Special issue on Self-Managing Systems and Networks.

[33] L. Zhou and R. van Renesse. P6P: A peer-to-peer approach to Internet infrastructure. In *Proc. of the 3rd Int. Workshop on Peer-To-Peer Systems*, San Diego, CA, February 2004.