Self-stabilizing and Byzantine-Tolerant Overlay Network*

Danny Dolev[1], Ezra N. Hoch[1], and Robbert van Renesse[2]

[1] School of Engineering and Computer Science
The Hebrew University of Jerusalem, Israel
{dolev,ezraho}@cs.huji.ac.il
[2] Dept. of Computer Science
Cornell University, Ithaca, NY
rvr@cs.cornell.edu

Abstract. Network overlays have been the subject of intensive research in recent years. The paper presents an overlay structure, *S-Fireflies*, that is self-stabilizing and is robust against permanent Byzantine faults. The overlay structure has a logarithmic diameter with high probability, which matches the diameter of less robust overlays. The overlay can withstand high churn without affecting the ability of active and correct members to disseminate their messages. The construction uses a randomized technique to choose the neighbors of each member, while limiting the ability of Byzantine members to affect the randomization or to disturb the construction. The basic ideas generalize the original *Fireflies* construction that withstands Byzantine failures but was not self-stabilizing.

1 Introduction

Network overlays have become a basic technique for routing among a dynamic set of participants. The literature studies various efficiency measures and availability issues. Recent papers address the issues of stabilization [1,2,3] and overcoming Byzantine faults [4,5,6]. In the current paper we extend the *Fireflies* construction [6], making it self-stabilizing. We call the resulting system *S-Fireflies*, for "stabilizing" *Fireflies*. *S-Fireflies* provides robust support (middleware) for various peer-to-peer and distributed applications, including Distributed Hash Tables ([7,8]) and reliable broadcast. For example, *Fireflies* has been used for Byzantine video streaming [9] and secure dissemination of software patches [10], and *S-Fireflies* can make the same applications significantly more robust.

S-Fireflies provides a *dissemination structure* along which members can exchange messages. The structure is an overlay graph among currently active members, having logarithmic diameter and adapting to churn (members coming and going). It overcomes

* This material is based in part upon work supported by ISF, ISOC, by the AFOSR under Award No. FA8750-06-2-0060, FA9550-06-1-0019, FA9550-06-1-0244, and by the National Science Foundation under Grant No. 0424422. Any opinions, findings, and conclusions or recommendations expressed in this publications are those of the author(s) and do not necessarily reflect the views of the AFOSR, NSF or ISF.

E. Tovar, P. Tsigas, and H. Fouchal (Eds.): OPODIS 2007, LNCS 4878, pp. 343–357, 2007.

Byzantine members who may try to prevent correct members from reliably communicating, or to cause them to send a large amount of useless messages. We assume that the networking facility underneath the overlay allows any two correct and active members to establish a communication channel, resembling networking over the Internet.

Each member maintains communication channels to a subset of the active members. The dissemination structure is composed of a dynamic number of "random subgraphs" that determine neighboring members with whom a member communicates. The sequence of subgraphs is used also for constructing additional structures that monitor availability of neighboring members.

Members may leave or crash, some may be Byzantine, and some or all may face transient faults that arbitrarily change values stored in their memory. *S-Fireflies* guarantees that the system will maintain its robustness as long as the number of correct and active members is sufficiently larger than the number of Byzantine members and the number of members that have recently recovered. Moreover, if the system loses the required ratio among correct and failed members, it will converge to a robust overlay once the ratio is restored and remains so for a long enough period of time.

When the system recovers from a transient fault members may not know which are the currently active members. Moreover, it may be that the system may find itself in several disjoint components. In [1] the authors assume the existence of some failure detection subsystem, and when instability is identified, members flood the network to form a new stable membership. In [3] a gossiping style is used, with members occasionally probing potential neighbors to identify whether they are active or not. Our technique resembles this later one, though we reduce the ability of Byzantine members to probe all members all the time.

A significant challenge of overcoming churn and facing Byzantine failures is to find ways to limit the ability of faulty members to take advantage of high churn to destroy the system's structure. Mechanisms that deal with churn and transient faults may make the system prone to replay attacks by Byzantine members. A typical use of counters becomes problematic in the environment we envision, and special care need be given to the use of digital signatures. While we address these issues in the *S-Fireflies* system, the technique presented in this paper is general and can be applied to improve the robustness of other overlay networks. Our protocols use randomization and the results are achieved with high probability (whp).

1.1 Related Work

The structures that we create are intended to simulate random graphs, in contrast to ring-based Distributed Hash Tables (DHTs) like Chord [7]. In Chord, members are organized in a single ring, with each member having $\log N$ "fingers" pointing across the ring that provide routing shortcuts. Instead, in *S-Fireflies* each member has $\log N$ pseudo-random neighbors, from which we construct the various structures. In both cases members end up with $\log N$ neighbors, but an important difference is that in *S-Fireflies* the neighbor relation is easily verifiable, preventing a Byzantine member from claiming to be a neighbor of an arbitrary other member.

[4] describes defenses against various Byzantine behavior for Pastry [8], another ring-based DHT. The paper suggest remedial approaches to impersonation [11], as well

as to attacks on overlay routing table maintenance and message forwarding. An *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. [5] suggests thwarting this attack by enforcing bounds on the in- and out-degrees of P2P members. None of these approaches consider self-stabilization however.

There has been a variety of work on Byzantine-tolerant epidemic protocols, apparently starting with [12]. These protocols consider the problem of correct members not accepting any malicious updates without using unforgeable signatures, and use a form of voting instead.

Drum [13] 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 *S-Fireflies*.

The issue of self-stabilization was studied in the context of group membership [1]. That work assumes that there is failure detection—once the system detects failure it switches to a stabilization phase. The main objective is to reduce communication overhead. The paper does not deal with permanent presence of Byzantine faults.

One observation in the current paper is that in order to overcome Byzantine members there is a need for a high connectivity underlying graph. One could consider using Harary graphs, or even Logarithmic Harary Graphs [14]. Unfortunately, despite its high connectivity, such structures are fragile and cannot be built on-the-fly in the presence of Byzantine members.

2 The Model

The system consists of a set $\mathcal{P}$ of members. Each member, m, has an identifier $m.id \in \mathcal{P}$, that is randomly assigned by a central authority (CA). To simplify notations we assume that all identifiers are in $[1, \ldots, |\mathcal{P}|]$, and we use the convention $m \in \mathcal{P}$. Members can be *active* or *passive*. An active member participates in the protocol; a passive one may be dead or detached. Some of the active members may be *Byzantine.* Members may go through transient periods resulting in an arbitrary state of the various variables, though the protocols (consisting of code and constants) are hard-coded and unaffected by transient faults. We assume that members may dynamically fail; failed members may recover and need to be re-integrated into the system.

We assume the existence of a public key cryptography scheme that allows each member to verify the signature of each other member. We further assume that non-Byzantine members never reveal their private keys,[1] such that faulty members cannot forge signatures. Member identifiers and their keys are part of their hard-coded state. The keys (as well as the signed identifier) are acquired via a trusted CA.

We assume that after going through a transient failure period the system eventually recovers, and at steady state the probability of an active member being Byzantine is bounded by p_{byz}.[2] We also assume that the communication network allows any two

[1] A Byzantine member that reveals its private key can never recover and be considered correct.

[2] One can assume that this holds only when the number of active members is more than some small n_0. One may generalize p_{byz} to be a distribution over the number of active members.

active members to establish a secure communication channel. Moreover, there is a constant δ that bounds, with high probability, the time it takes messages among active members to reach their destination. Informally, we are interested in establishing an overlay among a given set of members over the Internet.

Correct members have internal timers that run at a bounded drift from real time, which enable them to measure periods of time with relative precision. We do not require clock values to be synchronized.

Members that face transient failure may find themselves in an arbitrary state. Therefore it may take some time to integrate them back into the system.

Definition 1. *An active member is* non-faulty *if it follows its protocols, processes messages in no more than π real-time units and has a bounded drift of its internal timer. An active member that is not* non-faulty *is considered* Byzantine.

A member will be called *faulty* or Byzantine, interchangeably.

Definition 2. *A member is* correct *if it has been* non-faulty *for Δ_{memb}.*

The value of Δ_{memb} is determined in Theorem 2. The communication network itself may face periods of time during which it deviates from its assumed properties.

Definition 3. *A communication network is* non-faulty *if messages arrive at their destination within δ real-time, and the content of the messages as well as the identity of the sender are not tampered with.*

Definition 4. *A communication network is* correct *if it has been* non-faulty *for Δ_{net} real-time.*

The value of Δ_{net} is chosen such that all messages that were sent before t_1 or were forged due to transient faults in the network are removed by $t_1 + \Delta_{net}$.

Definition 5. *A system is* coherent *if there is group $\mathcal{G}$ of* correct *members, such that $|\mathcal{G}| \geq N \cdot (1 - p_{byz})$, where N is the number of the currently active members, and the network connecting the members in $\mathcal{G}$ is* correct.

In the following, we will discuss only members from $\mathcal{G}$, thus, when stating "correct member m," we actually mean "correct member m s.t. $m \in \mathcal{G}$."

Once the system is coherent, a message between any two correct members is sent, received, and processed within d real-time units, where d includes δ, π, and drifts of local timers. For simplicity we will assume that $\Delta_{net} = 2d$, though one can choose different values.

3 The BSS Overlay Service Specification

Each correct member has a *view*, $m.view$, which is a subset of all members, $\mathcal{P}$. Informally, $m_2 \in m_1.view$ means that m_1 believes that m_2 is, at least until recently, neither stopped nor exhibiting Byzantine behavior. Conversely, $m_2 \notin m_1.view$ means that m_1

believes that m_2 is stopped or faulty.[3] The subset $m.neighbors$ of a member's view represents its *neighbors*.

An *overlay* G, $G = (V, E)$, is a directed graph whose members, $V \subseteq \mathcal{P}$, are the active members and $E = \{(m_i, m_j) | m_i, m_j \in V, m_j \in m_i.neighbors\}$. A *Byzantine-Self-Stabilizing (BSS) overlay* G, is an *overlay* G that is Byzantine-tolerant and self-stabilizing. Thus, a BSS-overlay forms a usable routing substrate among the active members that is highly robust. We refer to the graph spanned by the BSS-overlay in which each directed edge is replaced by an undirected one as *underlying*-BSS-*overlay* . For maintaining the BSS-overlay each member has an additional list: $m.detect$ contains members used for failure detection.

Our goal is to design a protocol that guarantees that when the system is coherent the following properties hold with high probability:[4]

P1: There is a directed path in the BSS-overlay from any correct member to any other correct member composed of only correct members;
P2: The diameter of the underlying-BSS-overlay is bounded by $O(\log(V))$;

Our aim is to develop protocols that converge from any arbitrary initial state, once the system stabilizes and there are enough correct members; i.e., the protocols spans a BSS-overlay among the active members satisfying the properties above. Moreover, we wish to reduce the time it takes for the system to converge and for a recovering member to be considered correct, i.e., to obtain Δ_{memb} as small as possible.

4 Data Structures

4.1 The Sequence of Subgraphs

The randomization used in the various structures of *S-Fireflies* are derived from a sequence of permutations. Each member m has a list $m(r_1), \ldots, m(r_j)$ of permutations, where $m(r_i)$ is a permutation over $\mathcal{P} - \{m\}$. For each permutation m should connect to the first member in that permutation, and if that member is down, m will connect to the next member in the permutation, and so on.

The solution proposed in this paper requires the list of permutations to be chosen independently and uniformly at random. This list should have at least $|\mathcal{P}| \cdot log(|\mathcal{P}|)$ permutations in it, since each member should have at most $\log N$ neighbors. According to experiments done by [6], a collision-resistant hash function can provide the required "randomization".[5] Therefore, we will assume the existence of a hash function

$$\mathcal{H} : (\mathcal{P}, \mathcal{N}) \rightarrow permutations([1, \ldots, |\mathcal{P}|]),$$

where $\mathcal{N}$ is the set of natural numbers and $permutations()$ is the set of all permutations over some group, such that for each member and subgraph index there is a permutation over the set $\mathcal{P}$. It is assumed that each member knows all permutations.

[3] We do not provide Virtual Synchrony properties such as consensus on views.
[4] The probabilities can be tuned to any desired probability.
[5] One can have the CA randomly select the permutations and send them to each member, exchanging practical performance with theoretical robustness.

Table 1. Additional View Lists

List	Description	Duration	Action
rcnt_suspected	members that were recently suspected	2Δ	remove from $m.view$ move to *rcnt_removed*
rcnt_removed	members that were recently removed	Δ	remove
to_be_joined	recently accepted members	Δ	move to $m.view$

Members are aware of their successors and predecessors on the various subgraphs, where a successor m of m' on the i^{th} subgraph is the first active member along the r_i permutation of m' as perceived by the view of m'; if m is a successor of m' then m' is the predecessor of m (on subgraph i). The actual number of subgraphs that a member uses may differ in different structures and will be specified for each one accordingly. On subgraph i:

$$succ_i(m) = \min_j\{m(r_i)_j | m(r_i)_j \in m.view\} \ ,$$
$$pred_i(m) = \{m' | m \in succ_i(m')\} \ ,$$

Recall that $m \notin m(r_i)$. Note that $pred_i(m)$ might contain several members. When the specific subgraph is clear from context we omit the subscript i. Our notations resembles those of [2].

We introduce operators that represent the segment of potential successors or predecessors of a member in a subgraph:

$$seg\_succ_i(m) = \{m(r_i)_j | j < location(succ_i(m), m(r_i))\} \ ,$$
$$seg\_pred_i(m) = \{m' | m' = succ_i(m), location(m', m(r_i)) \neq 1)\} \ ,$$

where $location(m, perm)$ returns the index of m in the permutation $perm$. Observe that these operators depend on the current view of the member, and different members might have different views.

4.2 The Views

High churn in the system and uncertainty about the time at which various members update their views require that each member maintains temporary lists of additional members, as described in Table 1. *S-Fireflies* members gossip on the BSS-overlay *dissemination structure* (Section 5), and the connectivity is chosen so with high probability all members learn of new gossip within Δ time units.

A member that is suspected as failed (as described in Section 6) is listed on the *rcnt_suspected* list and if it does not rejoin within 2Δ (as described in Section 7) it is removed from $m.view$ (and from *rcnt_suspected*). During the uncertainty period members in this list are still considered as potentially connected. The process of joining is also a double step. A joiner that will be accepted (as described in Section 7) is first placed in the *to_be_joined* list, and will be integrated into $m.view$ only after being in the list for Δ time units, giving the rest of the members a chance to identify the new addition.

S-Fireflies does not reach agreement on views, therefore views can always differ. At steady state the difference among the views of two correct members is due to the lists above and due to Byzantine behavior. To accommodate for that flexibility, when a member m considers its view, $m.view$, to check whether m' is allowed to connect to m, it will actually consider *rcnt_removed*'s and *to_be_joined*'s effect on $m.view$; that is, m will accept m' to connect to it if there is an update of $m.view$ with members from *rcnt_removed* $\cup$ *to_be_joined* such that $m' \in pred_i(m)$ (for some $i \leq r'_m$, as defined in Section 5). In such a case we say that the view of m' is close to the view of m. However, when m considers which members to connect to, it will consider $m.view$ only. Such a behavior will allow the required flexibility for m and m' to connect to each other in the presence of joining and leaving members.

4.3 The Epoch and Epoch List

The assumed existence of digital signatures reduces the ability of Byzantine members to mislead correct members. But since members may fail and recover, Byzantine members can replay old signed messages. In a self-stabilizing environment it is challenging to identify replayed messages.

In order to reduce the ability of Byzantine members to perform a convincing replay attack, a member needs a mechanism that produces some randomization to its new identity when it recovers. To achieve that, a member chooses periodically (and during recovery) a new random incarnation number. A new *epoch* of a member is the signed pair (*prev_inc*, *new_inc*), where *prev_inc* is its previous incarnation value and *new_inc* is its new incarnation value. The incarnation values are random numbers from a large enough space (much larger than the memory space of the faulty members), so that the probability that a member repeats the pair (*prev_inc*, *new_inc*) is negligible, and the ability of a faulty member to replay such a pair is even smaller. We will ignore this small probability of error.

The introduction of a random epoch is similar to choosing a random id. Therefore, in our protocols, whenever a member sends a signed message it should include its current epoch. We assume that members send signed messages, and members ignore any message that is not signed properly or does not carry the matching epoch. We will ignore these details when describing the protocols.

Each member maintains as part of its view the latest epoch of each member in the view ("the epoch list"). A receiver of a signed message will consider the message *current* only if the epoch matches the last epoch the receiver knows of. If the recent epoch was received less than Δ ago, it can still accept signed messages containing the previous epoch value. The message is current also when the member did not receive the new epoch yet, but its latest copy of the epoch matches the *prev_inc* part of the epoch of the received message. When a member m_1 updates its epoch (done once in Δ_{epoch}) it will send a special message containing the new epoch; this message is disseminated the same as other messages in the system. If a member m_2 receives such a message and m_2's current epoch is equal to m_1's *prev_inc* then m_2 updates its view of m_1's epoch.

5 The Dissemination Structure

The dissemination structure is defined according to the neighbor relations induced by the set of subgraphs as determined by the size of the views of individual active members. Let $N_m = |m.view|$ and define g_m, the number of active members m establishes a connection to, as

$$g_m = g_0 + \lceil \frac{1}{1 - p_{byz}} \ln N_m \rceil,$$

where g_0 is a minimal number of neighbors (defined in Theorem 1). Member m establishes a secure channel with the g_m different members of $m.view$ that are *succ*(m) in one of the first $r(m.view)$ subgraphs, where $r(m.view)$ is the minimal number of such subgraphs satisfying

$$r_m(m.view) = \min_i(\Big| \bigcup_{j \leq i} \{m' | m' = succ_j(m)\} \Big| = g_m)$$

$r(m.view)$ and r_m will also be used. These g_m members are the members of the $m.neighbors$ set, and member m will gossip its messages along these channels. Note that whp $r_m = g_m$.

Since views of different members may differ, member m accepts a connection request from each active member m' who is *pred*(m) in one of r'_m permutations of m', defined as:

$$r'_m = r(m.view \cup rcnt\_removed \cup to\_be\_joined) \cdot (1 + p_{byz}).$$

Thus, m estimates the number of subgraphs of m' within which it appears as a *succ*(m') in order to accept the connection. In such a case we say that the view of m' is *close* to the view of m. This notion of "close" intends to allow for two correct members to differ by the potential presence of current Byzantine members and view changes that are in transit. As in [6], if the request arrives from a member that is not *pred*(m), a message is returned containing an update.

The dissemination structure that defines the BSS-overlay is composed of the active members and the secure channels they establish with their neighbors. Each member m has g_m outgoing links.

Theorem 1. *A gossip protocol over* BSS*-overlay completes with high probability within* $\Delta = (\ln N + c_0) \cdot d$*, where* N *is the number of currently active and correct members and* c_0 *is a constant that depends on the probability of message loss and on* p_{byz}*.*

Proof. Sketch: Kermarrec et al. [15] show that it is possible to build effective gossip protocols if each member only has a small set of uniformly chosen members it gossips with. In the dissemination structure, each member m effectively gossips with some g_m *neighbors* from its view uniformly at random, where g_m is large enough to create a connected graph of correct members; if we have k neighbors, then $p_{byz} \cdot k$ neighbors will by Byzantine (in expectation), hence we would like to have $(1 + p_{byz}) \cdot k$ neighbors; now we have an additional $p^2_{byz} \cdot k$ Byzantine neighbors, and so on. Using $\sum_{i=0}^{\infty} p^i_{byz} = \frac{1}{1-p_{byz}}$, we obtain the definition of g_m.

Detection of a Crashed Member /* *executed at member* m */
/* *others members act upon receiving appropriate message* */
/* *all gossip along the* BSS-*overlay* */

Monitoring m**:**
If suspects crashing of $m' = succ_i(m)$ then
add m' to *rcnt_suspected*;
disseminate "suspect$(m, epoch_m, m', epoch_{m'})$",
where $epoch$ is m's epoch and $epoch_{m'}$ is m''s;

Member m''**:**
when received "suspect$(m, epoch_m, m', epoch_{m'})$" from m and $m' \neq m''$ do
if $epoch_m$ and $epoch_{m'}$ current and $m = pred(m')$ and $m \notin banned(m')$ then
add m' to *rcnt_suspected*;
disseminate "suspect$(m, epoch_m, m', epoch_{m'})$",
when received "suspect$(m, epoch_m, m', epoch_{m'})$" from m and $m' = m''$ do
if $epoch_m$ and $epoch_{m'}$ current and $m = pred(m')$ then
add m to $banned(m'')$
if $|banned(m'')| > f_{m''}$ then $banned(m'') = \perp$
invoke a new incarnation of m''

Fig. 1. Handling Suspicions

A classic result of Erdös and Rényi [16] shows that in a graph of n members, if the probability of two members 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 proof follows this line of arguments, using the potential difference between views of different members, their additional lists, and their estimate of the number of currently active members. The value of g_0 is determined by c and the initial n for which the estimates hold. □

6 Membership Maintenance

The membership maintenance draws upon ideas presented in [6]. The basic idea is that members exchange accusations regarding suspected misbehavior of other members. They keep track of other members using the detection structure (defined below) and gossip their accusations using the dissemination structure.

In the *detection structure*, each member maintains outgoing links (some of which may overlap with the links of other structures) with its successors in a number of subgraphs such that each member in its view has g_m different $pred(m)$ members. The number of detection subgraphs that an active member considers is determined as follows: Increase the number of subgraphs until for every $m' \in m.view$ there exist at least g_m different members of $m.view$ as their predecessors on the different subgraphs. More formally:

$$detect(m) = \min_r(|\bigcup_{i \leq r} \{pred_i(m)\}| \geq g_m).$$

Each member maintains secure channels with the set of its successors along these subgraphs. When a member is requested to establish a secure channel it checks that the requester is in *pred* for one of the subgraphs implied by that view (while considering *to_be_joined* and *rcnt_removed*).

The detection structure is the graph spanning the active members and all channels to their successors in one of the detection subgraphs, as defined above. Member m monitors $succ_i(m)$, for each $i \leq detect(m)$, and is expected to be monitored by $pred_i(m)$.

We use the pinging techniques of [6]. If a member suspects that the member it monitors fails, it gossips along the dissemination structure an accusation message as defined in [6], except that each such message carries the epoch as defined above. Observe that the pinging technique does not have long term state and therefore introduces no difficulty to stabilization of the system.

To prevent faulty members from continuously sending accusation messages about correct and active members, each member maintains a list *banned-members* flagging up to f_m of its predecessors as disabled. A predecessor that is disabled cannot disseminate any accepted accusation of a member. Figure 1 presents the schematic protocol that handles crash detection.

The view of each member includes not only the identities of members it assumes to be active and correctly operating, but also for each member the latest signed epoch and the vector of disabled predecessors.

Observe that the detection structure does not need to use the technique of skipping across accused members, used in [6], in order to guarantee that each member has a monitoring member. Our detection structure has that property by construction.

Members also track the activities of other members and if they can prove Byzantine behavior they can disseminate such a proof and members can remove the faulty member. We will not elaborate on that optimization.

Lemma 1. *A crashed member will be removed from the view of every correct and active member within 3Δ, whp.*

Proof. When the system becomes coherent correct members exchange messages within Δ whp. A crashed member has at least one correct and active predecessor that is not in its banned-members list. That member will detect the crash and will be able to disseminate that to all correct and active members whp. Within Δ it will reach all correct and active members, and within an additional 2Δ those members will update their views. □

7 Recovery of Members

Due to the self-stabilizing requirement, the system must cope with transient faults. Members can be subject to such faults, and may be able to identify them via inconsistencies in their internal state, or they may realize that other members suspect them as failed. In such cases, the member needs to recover and integrate back into the system.

Observe that the new epoch cannot be disseminated as is because simple gossiping will enable faulty members to gossip about past values, thus enabling replay attacks. The first step of recovering a member is to ensure it has an updated epoch list.

Sending epochs and views to all members */* *executed at member* m *every* $\Delta_1/|\mathcal{P}|$ *time units* */*

Member m**:**

Send epoch and $m.view$ to member i;

$i := i + 1(mod\ |\mathcal{P}|)$;

Member m'**:**

upon receiving epoch and view from m, update epoch list with m's epoch, and update $m'.view$ with $m.view$

Fig. 2. Background process: sending epochs and views to all members

7.1 Epoch Renewal and Epoch List Stabilization

The usage of signatures handicaps Byzantine members to some degree; however, the Byzantine members may replay signed messages. To prevent this, each member p has a counter that it includes in each message and increments for each message. Receiving members will not accept a message from a member with a lower counter than expected. Due to transient faults, members in the system may have invalid values for these counters. Moreover, Byzantine members may replay messages of higher counters in case of a transient fault that caused the correct members to think that there are lower counter values.

To overcome both these issues a member should select a new epoch every so often (depending on the system security requirements); with each new epoch the message counter is reset. However, there might be a mismatch between a receiving members' value of $prev\_inc$ of p and p's value of $prev\_inc$; this will lead to members not accepting the new incarnation.

We consider two scenarios: The first is a scenario in which a majority of correct members have undergone transient faults, and their epoch lists are not valid anymore; in the second scenario only a small portion of members have undergone transient faults (this is likely the more common case in practice).

To solve the first case – when many members have undergone transient faults – we use a background process that periodically sends the epoch and view to each member (see Figure 2). The second case is solved by contacting the immediate neighbors and updating the list of epochs and views according to their majority agreement (see Figure 3).

Transient faults may disturb other data structures as well as the epoch list. If the value of $m.view$ is too far off to even connect to other members to gather information about the epoch list then m has to wait Δ_1 time for the algorithm in Figure 2 to update its epoch list. In case $m.view$ approximately represents which members are up and which not then the algorithm in Figure 3 will operate correctly and "re-update" the epoch list of m within $\Delta_{1'}$ time.

In Figure 3, the size of the group to request the epoch list from (N_{epoch}) affects tolerance to multiple transient failures. To increase tolerance this size can be increased. Let p_{trans} be the probability of having a transient fault at some member. If $(p_{byz} + p_{trans}) < 1/2$ then the larger $\mathcal{G}_{epoch}$ is, the higher the probability of "hitting" enough

Getting epoch list /* *executed at member* m *every* $\Delta_{1'}$ *time units* */
Member m**:**
$\mathcal{G}_{epoch}$:=randomly select N_{epoch} active members from $m.view$;
request epoch list and view from all members in $\mathcal{G}_{epoch}$;
upon receiving responses (wait at most $2d$ to collect responses):
for each member, select the epoch that appears most often.
for each member, select the state that appears most often.

Fig. 3. Background process: getting epoch list and views from members

Choosing a new epoch
Member m**:**
either once every Δ_{epoch} **or** if m has been accused:
randomly select a new epoch;
disseminate the new epoch;
Member m'**:**
upon receiving information about a new epoch of member m, if m' epoch list contains m's *prev_inc* then update the epoch list with the current epoch of m;

Fig. 4. Background process: choosing a new epoch

correct members in the search for epoch lists. Moreover, if $m.view$ was subject to some transient faults, then the larger $\mathcal{G}_{epoch}$ is, the larger group out of $m.view$ is examined, which increases the probability of reaching enough correct members.

Assume that each member renews its epoch to ensure that the epoch is always fresh every Δ_{epoch} period (as specified in the next subsection).

Lemma 2. *Starting from any state, each correct member has an updated epoch list within* $\Delta_1 + \Delta_{epoch}$.

Note that from this point on correct members can communicate safely among each other; also, Byzantine members cannot use replay attacks because all epochs have been changed.

7.2 Periodical Epoch Update

A member m creates a new epoch every Δ_{epoch} and disseminates the message among all members (see Figure 4).

Lemma 3. *An active and correct member that renews its epoch in less than* Δ *from the time the first correct and active member suspects it as failed succeeds to do so before it is removed from the view of any correct and active member, whp.*

Proof. Sketch: The renewal message carries the new epoch that matches the last epoch at all active and correct members in its previous view. Since dissemination takes less

than Δ, whp, its renewal message reaches and is accepted by each such member before it is removed from the *rcnt_suspected* list of any such member. Note that the renewal message may reach members that are not aware of the suspicion. If an old suspicion message will be received in such a case its epoch will not match and the message will be ignored. □

Note that the lemma also claims that a member that renews its epoch without it being in *rcnt_suspected* of any correct member also has its epoch accepted by all correct members within Δ whp.

Lemma 4. *A correct and active member will not enter the rcnt_suspected list of any correct and active member as long it remains active, whp.*

7.3 Stabilization of the Overlay Network Structures

In Section 7.1 it was shown that all correct members eventually agree on their epoch list. However, due to transient failures, members might disagree on the dissemination structures.

In the following the self-stabilization of the overlay network structures is considered (assuming the epoch list has stabilized). Consider all correct members in the system to be in an arbitrary state. That is, a member m has arbitrary values for $m.view$, *rcnt_suspected*, etc. By the algorithm in Figure 2 after Δ_1 all members will have similar view sizes; hence they will agree on the value of g_m.

Since each member continuously monitors the members in seg_succ_i (for all $i \leq r_m$) and in seg_pred_i (for all $i \leq r'_m$), then eventually each member will have connections with the members it should be connected too. (Note that the lists *rcnt_suspected*, *rcnt_removed*, *to_be_joined*, and *banned* are cleared when the items in them are old enough.)

From this point on all disseminations are performed correctly, as messages are sent and received along the "correct" connections; however, members still do not have valid views of all the network, and it will take time for this view to become consistent. Note that this view inconsistency is not an issue, as it only affects new connections in case some member leaves the overlay network and since *seg_succ* and *seg_pred* are always monitored, such failures will be detected.

In addition, members' states are continuously disseminated along the network (for example, due to periodic distribution of new epochs described in Section 7.2). Since each active member is connected to the overlay, once the overlay network has its connections set each member will disseminate its sate. Hence, after an additional Δ time units all members will receive each such dissemination and will have up-to-date views of the status of all other correct members.

Note that this stabilization will take no more than $O(\Delta + \Delta_{scan})$ time whp (where Δ_{scan} is the interval for scanning *seg_pred* and *seg_succ*, and is also the rate at which members refresh their *banned-members* list). Define the stabilization period $\Delta_2 = O(\Delta + \Delta_{scan})$.

Theorem 2. *Starting from an arbitrary state, each disjoint set of the system converges within* $\Delta_{memb} = \Delta_{epoch} + \Delta_1 + \Delta_2$.

Proof. Sketch: Starting from an arbitrary state, after $\Delta_{epoch} + \Delta_1$ time all non-faulty and active members agree on each other's epochs. From this step on, secure communication can commence. In addition, all members agree (approximately) on the view size. Hence, they all consider the same g_m which leads to the construction of a working overlay network; after an additional Δ_2 time, all correct members will have up-to-date overlay structures in their connected subgraph. □

Theorem 3. *The* BSS*-overlay with the detection and the integration structure satisfies properties P1 and P2, with high probability.*

Proof. Sketch: We prove that once the system is coherent the system converges from an arbitrary state to a safe state and that once it is in a safe state it remains in such a state unless the system becomes incoherent.

Let $\mathcal{G}$ be the set of members that are active and non-faulty for $\Delta_{epoch} + \Delta_1 + \Delta_2$. Within $\Delta_{epoch} + \Delta_1$ after the system becomes coherent each one of them will go through its subgraphs and will end up learning about all possible connected components. Within Δ_2 each member m will connect to at least g_m members. Observe that in these bi-lateral exchanges members add to their view each member they found active.

Within Δ each one will establish connections with g_m and will connect the BSS-overlay. Members that crash disappear from views and correct and active members remain in views. □

8 Conclusion

The paper presents a robust and self-stabilizing overlay network. In order to establish self-stabilization while overcoming Byzantine faults some unique techniques are developed. These techniques can help turn other constructions into self-stabilizing systems that withstand Byzantine faults. The basic techniques are: 1) the use of randomization to create permutations of the list of members; 2) the use of a pair of random numbers to form a member's epoch, instead of an ordinal number; 3) the introduction of integration and detection structures that enable dealing with high churn without the need to reconstruct the overlay network when members fail and recover.

References

1. Dolev, S., Schiller, E.: Communicaiton adaptive self-stabilizing group membership service. IEEE Transactions on Parallel and Distributed Systems 14(7), 709–720 (2003)
2. Ghodsi, A., El-Ansary, S., Krishnamurthy, S., Haridi, S.: A self-stabilizing network size estimation gossip algorithm for peer-to-peer systems. Technical Report Technical Report T2005:16, SICS (2005)
3. Shafaat, T., Ghodsi, A., Haridi, S.: Handling network partitions and mergers in structured overlay networks. In: Proc. of the Seventh IEEE International Conference on Peer-to-Peer Computing, Galway, Ireland (September 2007)
4. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.S.: 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. Singh, A., Castro, M., Druschel, P., Rowstron, A.: Defending against Eclipse attacks on overlay networks. In: Proc. of the 11th European SIGOPS Workshop, Leuven, Belgium, ACM, New York (2004)
6. Johansen, H., Allavena, A., van Renesse, R.: Fireflies: Scalable support for intrusion-tolerant network overlays. In: Eurosys 2006, Leuven, Belgium (2006)
7. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F.: Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proc. of the 1995 Symp. on Communications Architectures & Protocols, Cambridge, MA, ACM SIGCOMM (August 1995)
8. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, Springer, Heidelberg (2001)
9. Haridasan, M., van Renesse, R.: Defense against intrusion in a live streaming multicast system. In: 6th IEEE International Conference on Peer-to-Peer Computing (P2P 2006), Cambridge, UK (September 2006)
10. Johansen, H., Johansen, D., van Renesse, R.: Firepatch: Secure and time-critical dissemination of software patches. In: IFIP International Information Security Conference (IFIPSEC 2007), Sandton, South-Africa (May 2007)
11. Douceur, J.: The Sybil attack. In: Proc. of the 1st Int. Workshop on Peer-to-Peer Systems, Cambridge, MA (March 2002)
12. Malkhi, D., Mansour, Y., Reiter, M.K.: On diffusing updates in a Byzantine environment. In: Symposium on Reliable Distributed Systems, Lausanne, Switzerland, pp. 134–143 (October 1999)
13. Badishi, G., Keidar, I., Sasson, A.: 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), pp. 201–210 (2004)
14. Jenkins, K., Demers, A.: Logarithmic harary graphs. In: Proceedings of the 21st International Conference on Distributed Computing Systems, ICDCSW (2001)
15. Kermarrec, A.-M., Massoulié, L., Ganesh, A.J.: Probabilistic reliable dissemination in large-scale systems. IEEE Transactions on Parallel and Distributed Systems, 14(3), (March 2003)
16. Erdös, P., Rényi, A.: On the evolution of random graphs. Magyar Tud. Akad. Mat. Kutato Int. Közl 5(17), 17–61 (1960)