Group Spreading: A Protocol for Provably Secure Distributed Name Service

Baruch Awerbuch[⋆] and Christian Scheideler[⋆⋆]

Department of Computer Science, Johns Hopkins University, 3400 N. Charles Street,
Baltimore, MD 21218, USA, {baruch,scheideler}@cs.jhu.edu

Abstract. This paper presents a method called Group Spreading that provides a scalable distributed name service that survives even massive Byzantine attacks. To accomplish this goal, this paper introduces a new methodology that essentially maintains a random distribution of all (honest and Byzantine) peers in an overlay network for *any* sequence of arrivals and departures of peers up to a certain rate, under a reasonable assumption that Byzantine peers are a sufficient minority. The random distribution allows to proactively protect the system from *any* adversarial attack within our model.

1 Introduction

The Internet was originally designed for the purpose of being extremely robust against hardware attacks, such as natural disasters or wars. However, software attacks (such as viruses, worms, or denial-of-service attacks) have become increasingly severe over the past few years, whereas hardware attacks are negligible. Thus, for any distributed application to run reliably on the Internet, it is of utmost importance that it is robust against adversarial software attacks. This is especially important for critical applications such as name service, i.e. a service that translates names such as “machine.cs.school.edu” into IP addresses so that machines can communicate with each other.

The current way name service is provided in the Internet is server-based. However, server-based architectures are vulnerable to attacks. A much more robust alternative appears to be the recently emerged peer-to-peer paradigm with its strictly decentralized approach. Unfortunately, despite the appeal of a decentralized approach, it appears to be a daunting task to develop peer-to-peer networks that are robust against adversarial attacks. Obviously, in an open environment any attempt to keep adversarial peers out of the network is doomed to failure because a priori there are no trust relationships that would allow to distinguish adversarial peers from honest peers. So one approach has been to at least limit the number of identities adversarial peers can obtain. Here, the use of a certification authority was suggested that requires credentials, sensitive information, or a payment to obtain an identity that allows the peer to join the

[⋆] Supported by NSF grant ANIR-0240551 and NSF grant CCR-0311795.

[⋆⋆] Supported by NSF grant CCR-0311121 and NSF grant CCR-0311795.

system (e.g., [3]). However, being overly restrictive here would not only prevent adversarial peers but also many honest peers from joining the system, either because they cannot provide the necessary credentials or they are not willing to reveal sensitive information or to pay for their membership. Thus, it should be clear that without being overly restrictive, a certification authority will be ineffective in limiting the number of identities adversarial peers may obtain, allowing them to start so-called Sybil attacks [6] that can cause severe problems to all structured peer-to-peer systems that have been suggested so far. Hence, new designs are needed that provide reliability despite adversarial peers with a potentially unlimited number of identities.

The goal of this paper is to demonstrate that it is possible, under certain simplifying assumptions, to design *completely open* peer-to-peer systems that are *provably* robust against adversarial peers of *arbitrary behavior* with an *unlimited* number of identities, as long as the adversarial peers in the system (or more precisely, their currently active identities) are in a sufficient minority.

1.1 Distributed name service

A *peer* is defined as an entity with a unique identity, i.e. each peer p is uniquely identified by a tuple $(\mathsf{Name}(p), \mathsf{IP}(p))$ where $\mathsf{Name}(p)$ represents the name of p and $\mathsf{IP}(p)$ represents the IP address of p. In order to provide a distributed name service, the following operations have to be implemented:

- $\mathsf{Join}(p)$: peer p wants to join the system.
- $\mathsf{Leave}(p)$: peer p wants to leave the system.
- $\mathsf{Lookup}(\mathsf{Name})$: returns the IP address of the peer p in the system with $\mathsf{Name}(p) = \mathsf{Name}$, or NULL if there is no such peer.

These operations must be implemented so that they can be run *concurrently* and *reliably* in an *asynchronous* environment without any trust relationships in which adversarial peers have an unlimited number of identities at their disposal and behave in an arbitrary way (i.e. we allow Byzantine peers). To formalize this goal, we need a model (see also [1] for further details and motivation).

1.2 Security model

We consider a peer to be *adversarial* if it belongs to an adversary or it is simply unreliable. Otherwise, a peer is called *honest*. We do not assume any prior trust relationships between the peers. Hence, a priori honest peers cannot be distinguished from adversarial peers.

Certification authority. A necessary requirement for a name service as defined above to work correctly is that every possible name x has at most one peer p with $\mathsf{Name}(p) = x$, i.e. the Lookup operation provides a unique peer for a given name (if such a peer is currently in the system). To guarantee this property, an authority is needed that resolves conflicts among the peers and that prevents peers from taking over names of other peers. Thus, we assume that a certification authority is available that issues certified names to peers that want to enter the system and that only provides such a name if no peer has registered under that name before. Certified names allow peers to prove that they are the rightful owner of a name, which prevents peers from taking over the identities of other peers.

Semantics of Join, Leave, and Lookup. Join, Leave, and Lookup are operations acting on a name service relation $\mathsf{DNS} \subseteq \mathsf{Names} \times \mathsf{IPs}$ in the following way:

- $\mathsf{Join}(p)$: if this operation was initiated by $\mathsf{IP}(p)$ and p is correctly certified then $\mathsf{DNS} \leftarrow \mathsf{DNS} \cup \{(\mathsf{Name}(p), \mathsf{IP}(p))\}$
- $\mathsf{Leave}(p)$: if this operation was initiated by $\mathsf{IP}(p)$ then $\mathsf{DNS} \leftarrow \mathsf{DNS} \setminus \{(\mathsf{Name}(p), \mathsf{IP}(p))\}$
- $\mathsf{Lookup}(\mathsf{Name})$: if there is a peer q with $(\mathsf{Name}, \mathsf{IP}(q)) \in \mathsf{DNS}$ then return $\mathsf{IP}(q)$ and otherwise return NULL

Given that the certification authority maintains a mapping $\mathsf{CA} : \mathsf{Names} \rightarrow \mathsf{IPs}$ that is well-defined at any time (i.e. each name is associated with at most one IP address), also the lookup operation will be well-defined. Indeed, if the operations above are correctly implemented and executed, then $\mathsf{DNS} \subseteq \mathsf{CA}$ at any time and DNS consists of all identities currently present in the peer-to-peer system.

Notice that there are many ways for adversarial peers to attack the correctness of DNS: adversarial peers may execute $\mathsf{Join}(p)$ for honest peers currently not in the system or $\mathsf{Leave}(p)$ for honest peers currently in the system, or may leave the system without notice. Also, adversarial peers may attempt to provide a wrong answer to a lookup operation. So countermeasures have to be taken to protect the system against these attacks.

Network model. Our basic approach is to organize peers in a scalable overlay network in which every peer may be represented by multiple logical units called *nodes*. We allow arbitrary adversaries with bounded resources, i.e. the number of adversarial nodes is at most an ϵ-fraction of the honest nodes in the system at any time. Such adversaries are called *ϵ-bounded*.

We consider asynchronous systems in which every honest peer has the same clock speed but the clocks are not synchronized and there is no global time. Since honest peers are considered reliable, we assume that at any point in time, any message sent by an honest peer p to another honest peer q will arrive at q within a unit of time. (Other message transmissions may need any amount of time.) Furthermore, honest peers have unbounded bandwidth and computational power, i.e. an honest peer can receive, process, and send out an unbounded number of messages in a unit of time. The latter assumption allows us to ignore denial-of-service attacks, but it does *not* simplify the task of securing an overlay network against legal attacks (i.e. attacks exploiting security holes in its protocols). As long as adversarial peers do not transmit unnecessary packets, the number of messages an honest peer will have to deal with in a time unit will normally be low so that we believe that our protocols are practical despite this assumption. Designing provably secure overlay networks for honest peers with bounded bandwidth is very challenging and needs further research.

Bootstrap peers. We assume that the certification authority provides a limited number of so-called *bootstrap peers* that are always part of the overlay network. This list of peers may be downloaded by a new peer when it registers its name so that it can contact one of the bootstrap peers without contacting the certification authority again. Bootstrap peers are like normal peers. For the Join protocol to work correctly we assume that at least one of the bootstrap peers is honest. Otherwise, there is no reliable way for a new peer to join the system. However, the Leave and Lookup protocols should *not* rely on the bootstrap peers so that the system is scalable and can work correctly under ϵ-bounded adversaries even if all bootstrap peers are adversarial.

In this paper, we will assume that *all* bootstrap peers are honest.

Messages. Finally, we need some assumptions about how messages are passed. We assume that the (IP address of the) source of a message cannot be forged so that adversarial peers cannot forge messages from honest peers (which can easily be achieved). Also, a message sent between honest peers cannot be deleted or altered by the adversary (because peers normally sit at the edge of the network).

1.3 Security goal

Recall that our security goal is to implement the Join, Leave, and Lookup operations so that they can be run concurrently and reliably in an asynchronous environment. More precisely, any of these operations executed by any of the honest peers in the system should be executed in a correct and efficient way. “In the system”, “correct” and “efficient” require precise definitions.

A $\mathsf{Join}(p)$ (resp. $\mathsf{Leave}(p)$) operation is called *completed* if any $\mathsf{Lookup}(\mathsf{Name}(p))$ operation executed afterwards by an honest peer in the system (and before another $\mathsf{Join}(p)$ or $\mathsf{Leave}(p)$ operation) returns $\mathsf{IP}(p)$ (resp. NULL). A peer p is called *mature* if $\mathsf{Join}(p)$ has been completed and $\mathsf{Leave}(p)$ has not been initiated yet. A $\mathsf{Lookup}(\mathsf{Name})$ operation is called *completed* once the peer initiating the request accepts the return value.

An overlay network operation is said to execute *correctly* if it completes within a finite amount of time. Furthermore, an overlay network operation is called

– *work-efficient* if it is completed using at most polylog(N) messages and
– *time-efficient* if it is completed using at most polylog(N) time,

where N be the current number of nodes in the overlay network. The following definition is central to this paper.

Definition 1. *We call an overlay network* survivable *if, when starting with a consistent system of n honest nodes (i.e. there are no pending join or leave requests) and $N \geq n$ at any time afterwards, it can guarantee the correct and (time and work) efficient execution of any overlay network operation initiated by an honest peer for* $\mathrm{poly}(n)$ *time steps, with high probability, for any* $1/\mathrm{polylog}(N)$*-bounded adversary and a join/leave rate of up to* $1/\mathrm{polylog}(N)$*, i.e. at least* $N/\mathrm{polylog}(N)$ *peers may join or leave the network in a time unit.*

Notice that we only require correct and efficient executions for honest peers, i.e. we do not care whether the semantics of Join, Leave, or Lookup are violated for adversarial peers. For example, a Lookup(Name) request for some Name owned by an adversarial peer q is allowed to give inconsistent answers, i.e. some honest peers may receive the answer IP(q) and others may receive the answer NULL.

Also, notice that we have to add the term "with high probability" above, because we said that a priori, it is not possible to distinguish between honest and adversarial peers. So no absolute guarantees can be given, unless we completely interconnect all peers, which is highly inefficient and therefore out of question.

1.4 Existing work

Classical distributed computing methods [12, 4, 13, 16] use Byzantine agreement and two-phase commit approaches with inherently *linear* redundancy and overhead to maintain a consistent state.

The *proactive security* approach in [15, 11, 10, 2, 9] uses different coding techniques to protect unreliable data in *reliable* networks; applying these methods in our context still yields linear overhead.

Fixed topology networks as in [8], will work only for non-Byzantine peers, and only allow fail-stop faults; the construction cannot handle malicious behavior of even a few malicious players.

The reliability of *hash-based peer-to-peer overlays* (or DHT's) such as Chord [17], Pastry [7], and Tapestry [18] hinges on the assumption that the IDs given to the nodes are pseudo-random, so that they can cope with a constant fraction of the nodes failing concurrently, with only logarithmic overhead. While this may seem to perfectly defeat massive attacks under these randomness assumptions, DHT's cannot handle even small-scale adaptive adversarial attacks involving the selection of adversarial IP addresses (to get close to desired IDs). One such "Sybil" attack is described in [6]. Remarkably, the attackers do not need to do anything complex such as inverting the hash function; all that is needed is to get hold of a handful (actually, logarithmic) number of IP addresses so that IDs can be obtained that allow to disconnect some target from the rest of the system. This can be accomplished by a linear number (i.e. $O(n)$) of offline trial/errors. For similar attacks, see [5].

Random or unpredictable placement of data in a logarithmic size subset of locations (as in Freenet) ensures that data is difficult to attack, but also makes it *difficult to retrieve.* Specifically, data retrieval of randomly placed data requires a linear number of queries, which is definitely unscalable.

Recently, an overlay network design for robust distributed name service has been suggested [1] that satisfies all criteria of survivability apart from work-efficiency; the work overhead can be close to linear.

2 Non-survivable overlay networks

In this section we prove that predictable overlay networks and hash-based overlay networks (i.e. networks in which the ID of a node is determined by a hash function) are not survivable. Furthermore, we show that being able to enforce a limited lifetime is crucial for the survivability of systems based on a virtual space, like hash-based systems.

2.1 Predictable overlay networks

An overlay network is *predictable* if for any fixed join/leave sequence of peers the topology will always be the same in a consistent state. Notice that all hash-based overlay networks with a fixed hash function are predictable.

We start this section by demonstrating that *no* predictable overlay network can be survivable under our definition of survivability.

Theorem 1. *Consider an arbitrary predictable overlay network of maximum (peer) degree d that can handle any sequence of N join/leave requests of peers in T time units. Then there is a join/leave sequence of $2N$ peers so that an ϵ-bounded adversary with $\epsilon \geq d/N$ can isolate an honest peer in $O(T)$ steps.*

Proof. The proof is relatively easy. First, $2N$ honest peers join, and afterwards the first N peers that joined the network leave. This takes $O(T)$ time steps. Consider now any peer in the resulting network, say v, and let $w_1, \ldots, w_d$ be its neighbors. Then, consider the join/leave sequence of honest peers that is like the sequence above but without $w_1, \ldots, w_d$. Assign the join events for $w_1, \ldots, w_d$ to the adversary. Then we arrive at the situation that v is completely surrounded by adversarial peers. This sequence *always* works because the overlay network is predictable. Hence, the theorem follows. □

2.2 Hash-based overlay networks

Hash-based overlay networks are vulnerable to adversarial attacks even if the hash function is chosen at random, and it is a one-way hash function. The mere fact that peers do not change their location over time turns them into "sitting ducks". To illustrate how an attack on hash-based approaches would look like, consider the Chord system.

Suppose that we have a system currently consisting of a set V of n nodes, each representing a peer, and further suppose we have a (pseudo-)random hash function $h : \mathsf{Names} \rightarrow [0, 1)$ that maps nodes to real values in the $[0, 1)$ ring. The real value a node v is mapped to is called its *identification number* or ID and denoted by $\mathsf{ID}(v)$. The basic structure of Chord is a doubly-linked cycle, the so-called *Chord ring*, in which all nodes are ordered according to their IDs. In addition to this, every node v has edges to nodes $f_i(v)$, called *fingers*, with $f_i(v) = \text{argmin}\{w \in V \mid \mathsf{ID}(w) \geq \mathsf{ID}(v) + 1/2^i\}$ for every $i \geq 1$ ($[0, 1)$ is treated as a ring here).

Now, take any node v in Chord with hash value $x \in [0, 1)$. By generating a set A of adversarial nodes with hash values in $[x - \epsilon, x]$, $[x, x + \epsilon]$, and $[x + 1/2^i, x + 1/2^i + \epsilon]$ for all relevant i where $\epsilon > 0$ is sufficiently small, v will have no node pointing to it any more, and all nodes v is pointing to belong to A, with high probability. Hence, the peer p of v will effectively be isolated from the rest of the system. Notice that even a relatively modest adversary can come up with such a set A, even if the hash function is not invertible. It just has to try enough values (which is easily possible with SHA-1; the fact that the hash values may depend on IP addresses is not a limitation, because with IPv6 there will be plenty of them available – even for private users). Also, notice that an adversary just has to know the IP address of p (to compute x and) to start an attack on p.

2.3 Problems with unlimited lifetime

Also truly random IDs do not help as long as no node can be excluded from the system against its will, even if there is a secure mechanism for enforcing such an ID on *every* node that joins the system.

All hash-based systems are based on the concept of a virtual space. The basic idea underlying these systems is that nodes are given virtual locations in some space, and the overlay network is constructed based on these virtual locations. That is, depending on its virtual location, a node aims to maintain connections to other virtual locations and does this by establishing pointers to the nodes closest to these locations. See, e.g., [14] for a general framework behind this approach.

Thus, all an adversary has to do to attack such a system is to throw new nodes into the system at a maximum possible rate and to keep only those nodes that obtain IDs in regions the adversary intends to take over. Hence, unlimited lifetime can result in a fast degradation of randomness.

3 Outline of the Group Spreading Protocol

Finally, we give an outline of the Group Spreading Protocol that avoids the problems above. The details can be found in a full paper.

3.1 Basic approach

We start with some basic definitions. Recall that a *peer* is an entity with a unique *name* and a *node* is a logical unit in the system with a unique ID. A peer may have multiple nodes in the system. However, honest peers will limit their nodes to $O(\log N_t)$, where N_t denotes the number of honest nodes in the system at time t. A node is called *honest* if it belongs to an honest peer. We assume that honest nodes execute our protocols in a faithful and reliable way. Adversarial nodes may do *anything*. (Recall that we only have to worry about legal attacks because honest nodes have infinite bandwidth.) A *region* is an interval of length $1/2^r$ in $[0,1)$ for some integer $r \geq 0$ that starts at an integer multiple of $1/2^r$. The core ideas behind the Group Spreading protocol are:

1. every honest peer aims to maintain a group of $L_p = \Theta(\log N_t)$ nodes of consecutive remaining lifetimes from 1 to L_p time steps,
2. every honest node v maintains connections to all reliable nodes in all regions of size $1/2^{r_v} = \Theta(\log N_t)$ containing $\mathsf{ID}(v) \pm 1/2^i$ for some $i \geq 0$
3. the system enforces a random ID in $[0,1)$ on every node, and
4. the system enforces a lifetime of $O(\log N_t)$ on every node.

The reason for item 1 is that Group Spreading uses a simple ID generation mechanism that enforces the selection of a random ID if it terminates. But this mechanism may not terminate if adversarial nodes are involved in it. Thus, every honest peer keeps a group of $\Theta(\log N_t)$ nodes in the system so that, with high probability, sufficiently many nodes of a peer will be in regions without a close-by adversarial node, and therefore the ID generation mechanism can terminate in these regions.

Using this approach, we can prove the following theorem.

Theorem 2. Group Spreading *survives up to a $\Theta(1/\log N)$ fraction of adversarial nodes with $O(\log N)$ time and $O(\text{polylog} N)$ work per operation as long as the join/leave rate of honest nodes is $O(1/\log N)$ and the join rate of adversarial nodes is $O(1/\log^2 N)$.*

Next we sketch the proof of this theorem. We start with some notation that we will frequently use, followed by some basic assumptions. Afterwards, we sketch the protocols and their analysis.

3.2 Notation

- r_v: *range* of a node v, selected upon creation of v by a node w so that w's *view* (i.e. the nodes it knows) of the region of size $1/2^{r_v}$ containing $\mathsf{ID}(w)$ is as close as possible to $\rho(r_v - \log r_v)$ for some fixed constant ρ
- L_v: maximum lifetime of a node v, computed as $L_v = \lambda \cdot r_v$ for some fixed constant λ
- L_p: $(3/4)\min_{v \in p} L_v$
- $\hat{L} = \max_v L_v$ and $\ell = \min_v L_v$ over all nodes in the system
- $p = (\mathsf{Name}(p), \mathsf{IP}(p))$: represents a peer
- $v = (p(v), \mathsf{ID}(v), r_v)$: represents a node, where $p(v)$ is the peer owning v
- $\Gamma_{v,t}$: all nodes v is connected to at time t
- $R_i(v)$: the unique region of size $1/2^{r_v}$ containing $\mathsf{ID}(v) + \text{sgn}(i)/2^{|i|}$
- $m = (\mathsf{Source}(m), \mathsf{Dest}(m))$: represents a message
- B: set of bootstrap peers

- A_t: nodes that are part of a join operation of a peer at time t
- D_t: nodes that are part of a leave operation of a peer at time t
- C_t: nodes whose creation is started at time t
- M_t: nodes that are mature at time t (i.e. their creation is completed)
- V_t: nodes that are *legal members* at time t (i.e. they have a connection to an honest node in the system)

Given a set of nodes S, S^h denotes the set of honest nodes in S and S^a denotes the set of adversarial nodes in S. So $N_t = |V_t^h|$. Furthermore, given a set of nodes S and a region $R \subseteq [0,1)$, $S(R)$ denotes the set of nodes in S with IDs in region R. Given a set S_t and a time interval I, $S_I = \bigcup_{t \in I} S_t$.

3.3 Prerequisites

There is a sufficiently small constant $\epsilon > 0$ so that for all $t \geq 0$,

P1 $B \subseteq V_t^h$,
P2 $|A_t^h \cup D_t^h| \leq \epsilon \cdot N_t / \log N_t$,
P3 $|A_t^a| \leq \epsilon \cdot N_t / \log^2 N_t$, and
P4 $|V_t^a| \leq \epsilon \cdot N_t / \log N_t$.

Suppose that the adversary has bounded resources (concerning computational cycles and bandwidth). Then, in practice, conditions P3 and P4 could be enforced by presenting computational challenges or Turing tests to new nodes that are created via bootstrap peers and by continuously checking connections to other nodes in the system. If a peer does not respond in time, its request for creating a node is ignored by the bootstrap peer, resp. the connection to the corresponding node is removed.

3.4 Creating a new node

Suppose that a node u wants to create a new node v. Then u calls the Create operation, which does the following. (t denotes the current time step.)

1. ID generation stage:
   (a) u sends an ID generation request to all nodes in $G = \Gamma_{u,t}(R_0(u))$, then waits for $L_u/3$ steps, and afterwards asks the nodes in G to compute the ID and send it to u. If u has not received the same ID x from $\geq |G|/5$ nodes within $O(1)$ steps, it aborts the protocol. Otherwise, u continues with the authorization stage.
   (b) Each node $w \in G$ receiving an ID request, generates a random x_w and sends $h(x_w)$ (for some bit commitment scheme h) to all nodes in $R_0(u)$. Afterwards, it waits for $L_u/3 + O(1)$ steps during which it accepts commitments from other nodes till u sends an ID computation request to w. If w has not heard back from u by then, it aborts the protocol.

   (c) Each node $w \in G$ receiving an ID computation request from u waits until $L_u/3$ steps are over since it received the ID generation request and then sends x_w to all nodes in G. If w receives all $x_{w'}$ for all $h(x_{w'})$ it received before within $O(1)$ steps, it computes $x = \bigoplus_{w'} x_{w'}$ (including x_w) and sends x to u. Otherwise, it aborts the protocol.

2. Authorization stage:
   (a) u computes r_v and sends an authorization request with (x, r_v) to all nodes in G.
   (b) Each node $w \in G$ that sent u the same x $O(1)$ steps before, routes its view of $R_0(u)$ to the region R_x of size $1/2^{r_v}$ containing x, and each node w' in R_x routes its view of R_x back to the nodes in $R_0(u)$, giving a set $S_w \subseteq V_I(R_x)$ for w with $I = [t - 2\log N_t, t]$ if this process took at most $2\log N_t$ steps (otherwise, w aborts the protocol). If so, w sends an authorization to the nodes in S_w and forwards S_w to u.
   (c) Once u receives sets S_w from $\geq 3|G|/20$ nodes in G, it computes the set $G' = \{w' \in \bigcup_w S_w : \ |\{w \mid w' \in S_w\}| \geq |G|/10\} \subseteq V_I(R_x)$.
3. Integration stage:
   (a) v sends an integration request to all nodes in G'. Then it waits for $O(1)$ steps to make sure that all nodes relevant for v added v to their connection table. Afterwards, v sends an integration request to all nodes relevant for it.
   (b) Each node w in R_x that was authorized by sufficiently many nodes in $R_0(u)$ at most $\log N_t + O(1)$ steps ago, notifies v about all nodes relevant for v and authorizes all nodes relevant for v to integrate v in their connection table.
   (c) Each node w' relevant for v that receives sufficiently many authorizations from nodes in R_x and an integration request from u within $O(1)$ steps, adds u to its connection table.

Apart from the bootstrap nodes, every node is only allowed to initiate the Create protocol 3 times during its life.

3.5 Insert and Lookup operations

The Insert and Lookup operations use the binary search method of Chord to forward requests, with the only difference that they are region-based, i.e. messages are forwarded along a sequence of regions rather than nodes. An honest node v accepts a message m only if $v \in V_t(R_i(m))$ for some $i > 0$ and m was sent to v by at least $1/5$ of the nodes in $\Gamma_{t,v}(R_{i-1}(m))$, where $R_i(m)$ is the ith region on the path of m.

Each peer p in the system calls the Insert operation every $L_p/3$ steps to store $(\mathsf{Name}(p), \mathsf{IP}(p))$ in the unique region R_p of size $1/2^{r_v}$ for some node v of p that contains $h(\mathsf{Name}(p))$ for some one-way hash function $h : \mathsf{Names} \rightarrow [0, 1)$. This makes sure that sufficiently many honest nodes in R_p know p at any time. Thus, when executing a Lookup operation for $\mathsf{Name}(p)$, it will return $\mathsf{IP}(p)$ as long as p is in the system.

3.6 Join and Leave operations

When a peer p wants to join the system, it contacts some bootstrap peer q. q will then initiate 3 Create operations via one of its nodes in each step until p has L_p nodes with remaining lifetimes from 1 to L_p. Once this is done, p is mature. Afterwards, p will initiate 3 Create operations via one of its nodes in each step (using each node only once) to keep L_p nodes in the system.

Leaving is easy. The peer p simply does not create any new nodes and waits until all of its old nodes left the system.

3.7 Safety

The correctness of the operations crucially depends on whether the system is safe. That is, we require for all regions R with $|R| = (\gamma \log N_t)/N_t$ for a sufficiently large γ that

S1 $|V_t^h(R)| \in [(2/3)\gamma \log N_t, (4/3)\gamma \log N_t]$,
S2 $|M_{t-\ell/2}^h(R) \cap M_t^h(R)| \geq (1/3)\gamma \log N_t$,
S3 $|V_I^a(R)| \leq (1/20)\gamma \log N_t$ for $I = [t - \ell/2, t]$, and
S4 for all $t' \in [t - \hat{L}, t]$, $|V_{t'}^h(R)| \in [(1-\delta)|V_t^h(R)|, (1+\delta)|V_t^h(R)|]$ for a small constant $\delta > 0$.

Suppose that the system has been safe so far. Then the following claims hold:

C1 For all $R = R_i(v)$ for some $v \in V_t^h$, $|M_{t-\ell/2}^h(R) \cap M_t^h(R)| \geq |\Gamma_{v,t}(R)|/5$.
C2 For every routed message accepted by some $v \in V_t^h(R_i(m))$ there is a $t' < t$ s.t. m was sent to v by some $u \in V_t^h(R_{i-1}(m))$ if $i > 0$, and otherwise m was sent to v by $\mathsf{Source}(m) \in \Gamma_{v,t}(R_0(m))$.
C3 For every routed message m generated at time t with $\mathsf{Source}(m) \in M_t^h$ and all $i \geq 0$, m is accepted by all $v \in M_t^h(R_i(m)) \cap M_{t'}^h(R_i(m))$ at some time $t' \in [t, t+i]$.

Using these claims, one can show the following lemma.

Lemma 1. *As long as the system is safe, any* Insert *or* Lookup *operation or* Create *operation with a successful ID generation stage initiated by a mature honest node needs $O(\log N_t)$ time and $O(\text{polylog} N_t)$ work to be completed.*

3.8 Invariants

For the safeness of the system as well as the correct execution of Join and Leave operations, we need the following invariants ($\alpha > \beta > 0$ are constants).

I1 For all $v \in V_t$ it holds for all $w \in V_t^h$ with $v \in \Gamma_t(w)$ that w's views on v match. Thus, $\mathsf{ID}(v)$ and r_v are well-defined.
I2 For all $v \in V_t$, $\mathsf{ID}(v)$ is a random value in $[0, 1)$.
I3 For all $v \in V_t$ there is a $t' \in [t - \alpha \log N_t, t - \beta \log N_t]$: $v \in A_{t'} \cup C_{t'}$.
I4 For all $v \in M_t^h$ it holds that $M_t^h(R_i(v)) \subseteq \Gamma_{v,t}(R_i(v))$ for all i.

The safeness and the invariants are shown to be true by induction:

Lemma 2. *As long as the system is safe, the invariants are fulfilled.*

Lemma 3. *As long as the invariants are fulfilled, the system is safe, with high probability.*

Lemma 4. *As long as the system is safe and the invariants hold, any* Join *or* Leave *operation executed by an honest peer needs* $O(\log N_t)$ *time and* $O(\mathrm{polylog} N_t)$ *work to be completed, and every mature honest peer can keep* $\Theta(\log N_t)$ *nodes in the system, with high probability.*

This completes the proof of Theorem 2. The full paper will be made available at www.cs.jhu.edu/∼scheideler.

References

1. B. Awerbuch and C. Scheideler. Robust distributed name service. In *Proc. of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.
2. R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: Long-term protection against break-ins. *RSA CryptoBytes*, 3(1):1–8, 1997.
3. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. of the 5th Usenix Symp. on Operating Systems Design and Implementation (OSDI)*, 2002.
4. M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of the 2nd Usenix Symp. on Operating Systems Design and Implementation (OSDI)*, 1999.
5. S. Crosby and D. Wallach. Denial of service via algorithmic complexity attacks. In *Usenix Security*, 2003.
6. J. R. Douceur. The sybil attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
7. P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.
8. A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proc. of the 13th ACM Symp. on Discrete Algorithms (SODA)*, 2002.
9. Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *Proc. of the 38th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 384–393, 1997.
10. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 100–110, 1997.
11. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO '95*, pages 339–352, 1995.
12. L. Lamport. The weak Byzantine generals problem. *Journal of the ACM*, 30(3):669–676, 1983.
13. L. Lamport and N. Lynch. Distributed computing. Chapter of Handbook on Theoretical Computer Science. Also, to be published as Technical Memo MIT/LCS/TM-384, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1989.
14. M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. of the 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2003.
15. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. of the 10th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 51–59, 1991.
16. R. De Prisco, B. W. Lampson, and N. Lynch. Revisiting the Paxos algorithm. In *Workshop on Distributed Algorithms*, pages 111–125, 1997.
17. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of the ACM SIGCOMM '01*, 2001.
18. B.Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of California at Berkeley, Computer Science Department, 2001.