Robust 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 suggests a method called Trust-but-Verify that, under certain simplifying assumptions, provides robust distributed name service even under massive adversarial attacks.

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 system [4]. 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 [7] that can cause severe problems

* Supported by NSF grant ANIR-0240551 and NSF grant CCR-0311795.
** Supported by NSF grant CCR-0311121 and NSF grant CCR-0311795.

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 the 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.

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. However, this does *not* prevent the adversary from registering a large

number of adaptively chosen names that are different from names of the honest peers.

Notice that the certification authority does not need to be online during the operation of the peer-to-peer system if all peers have registered names with it before. Hence, it does not introduce a single point of failure for the operation of that system and therefore does not undermine the benefits of having a decentralized peer-to-peer system.

Semantics of Join, Leave, and Lookup. Recall that a peer p is defined as an entity with a unique, certified identity, i.e. $p = s(\mathsf{Name}, \mathsf{IP})$ where s is a signature scheme, Name is the name of p, and IP is the IP address of p. 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 $\mathsf{DNS} \leftarrow \mathsf{DNS} \backslash \{(\mathsf{Name}(p), \mathsf{IP}(p))\}$
- $\mathsf{Lookup}(\mathsf{Name})$: 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.

Resources. We allow arbitrary adversaries with bounded presence, i.e. the number of adversarial identities or peers is at most an ϵ-fraction of the peers in the system at any time. Such adversaries are called *ϵ-bounded*.

We consider asynchronous systems in which every honest peer has roughly the same clock speed but 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.

To simplify the Join protocol in this extended abstract, we will assume that *all* bootstrap peers are honest.

Notice that the concept of bootstrap peers is necessary because somehow new peers have to be able to find out about peers already in the system. Assumptions like "peers contact random peers in the system" are not realistic because how should a peer find this random peer over the Internet? Furthermore, it is important to have peers that are permanently present in the overlay network because otherwise a server would have to be established that is always accessible and that provides a continuously updated list of peers currently in the system, which introduces a single point of failure.

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. Also, a message sent between honest peers cannot be deleted or altered by the adversary.

Notice that the source issue can actually be solved easily without cryptography as long as adversaries cannot hijack IP addresses or listen to communication between honest peers: if a message arrives from IP address x, then the receiver y asks x for a confirmation that contains a secret (for example, a random key). Only if y receives an acknowledgement from x containing the secret, y will accept the message. The assumption that messages cannot be deleted or altered by the adversary is realistic in our case because we assume the peers of our overlay network to sit at the edge of the Internet, and therefore peers cannot manipulate communication between other peers.

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 Join(p) (resp. Leave(p)) operation is called *completed* if any Lookup(Name(p)) operation executed afterwards by an honest peer in the system (and before another Join(p) or Leave(p) operation) returns IP(p) (resp. NULL). A peer p belongs to the system if Join(p) has been completed and Leave(p) has not been initiated yet. A Lookup(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 peers in the overlay network.

Finally, we call an overlay network *survivable* if it can guarantee the correct and (time and work) efficient execution of any overlay network operation initiated by an honest peer, with high probability, for at least $1/\text{polylog}(n)$-bounded adversaries and a join/leave rate of at least $1/\text{polylog}(n)$, i.e. at least $n/\text{polylog}(n)$ peers may join or leave the network in a time unit.

This definition implies that in a survivable network, any overlay network operation initiated by an honest peer can be executed correctly for at least $\text{poly}(n)$ time units, with high probability, where n is the minimum number of peers in the system during that time.

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 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.

An overlay network is called *weakly survivable* if it can only guarantee correctness and time-efficiency, but no work-efficiency. In this paper we propose a design called Trust-but-Verify that provides weak survivability as long as the adversarial peers are in the minority. Recently, the authors also managed to design overlay networks that are survivable in the strong sense. See [1, 2] for details.

1.4 Existing work

Classical distributed computing methods [13, 5, 14, 17] use Byzantine agreement and two-phase commit approaches with inherently *linear* redundancy and overhead to maintain a safe state.

The *proactive security* approach in [16, 12, 11, 3, 10] 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 [9], 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 [18], Pastry [8], and Tapestry [19] 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 [7]. 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 [6].

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.

2 The Trust-but-Verify approach

The basic approach of our Trust-but-Verify scheme is similar to previous hash-based overlay networks such as Chord [18]. All hash-based overlay networks are based on the concept of a virtual space. The basic idea underlying these systems is that peers 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 peer aims to maintain connections to other virtual locations and does this by establishing pointers to the peers closest to these locations. See, e.g. [15] for a general framework behind this approach. To illustrate this approach, we give the Chord network as an example.

2.1 The Chord overlay network

Suppose that we have a system currently consisting of a set V of n peers, and further suppose we have a (pseudo-)random hash function $h : \mathsf{Names} \to [0, 1)$ that maps peers to real values in the interval $[0, 1)$. The real value a peer p is mapped to is called its *identification number* or ID and denoted by $\mathsf{ID}(p)$. The basic structure of Chord is a doubly-linked cycle, the so-called *Chord ring*, in which all peers are ordered according to their IDs. In addition to this, every peer

p has edges to peers $f_i(p)$, called *fingers*, with $f_i(p) = \text{argmin}\{q \in V \mid \mathsf{ID}(q) \geq \mathsf{ID}(p) + 1/2^i\}$ for every $i \geq 1$.

To avoid identity theft, h is usually assumed to be a one-way hash function so that it is hard to find a name $\mathsf{Name} \neq \mathsf{Name}(p)$ for some peer p so that $h(\mathsf{Name}) = h(\mathsf{Name}(p))$.

2.2 Problems with the Chord overlay network

Using a pseudo-random hash function allows to assume that IDs of honest peers are distributed uniformly at random in $[0, 1)$, but it does *not* allow to assume that also IDs of adversarial peers are distributed uniformly at random in $[0, 1)$. The problem is that adversarial peers may pick names deliberately for the purpose of getting very close to some values in $[0, 1)$.

For example, by generating a set A of adversarial peers with IDs in $[x - \epsilon, x]$, $[x, x + \epsilon]$, and $[x + 1/2^i, x + 1/2^i + \epsilon]$ for all relevant i with $\epsilon < 1/n^2$, peer p with $\mathsf{ID}(p) = x$ will have no honest peer pointing to it any more, and all peers p is pointing to belong to A, with high probability. Hence, if the peers in A leave, p will be disconnected from the system, with high probability. 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 names.

Once p is isolated, overlay network operations cannot be executed successfully any more, because p is not able to communicate with other honest peers in the system. Thus, the Chord overlay network is not survivable in our sense.

2.3 An approach robust against isolation attacks

The Trust-but-Verify overlay network also uses a pseudo-random one-way hash function to map peers to IDs in $[0, 1)$. (Recall that the pseudo-randomness makes sure that IDs of honest peers are random, and the one-way property makes sure that if the name of a peer cannot be taken over, it is also hard to take over its ID.) However, peers are interconnected in a different way.

A *region* is an interval in $[0, 1)$ of size $1/2^r$ for some $r \in \mathbb{N}_0$ starting at an integer multiple of $1/2^r$. Imagine for a moment that every peer knew the current number of peers, n, in the system, and that every peer p aims to maintain connections to all peers in the system with IDs in the regions $R_i(p)$, $i \in \mathbb{Z}$, where $R_i(p)$ is the unique region of size closest from above to $(c \log n)/n$, for some fixed constant c, that contains $\mathsf{ID}(p) + \text{sgn}(i)/2^{|i|} (\text{mod } 1)$.

Suppose now that the ID of every honest peer is like an independent, random value in $[0, 1)$ (but hash values of adversarial peers may be any values different from the values of honest peers). Then the well-known Chernoff bounds imply the following result:

Theorem 1. *For any $(1 - \epsilon)$-bounded adversary for some constant $\epsilon > 0$ there is a constant $c = O(1/\epsilon)$ so that in every region of size closest from above to $(c \log n)/n$ there are $\Omega(\log n)$ honest peers, with high probability.*

It follows from the theorem and the way the regions are selected that *no matter how the IDs of adversarial peers are distributed in* $[0,1)$, the honest peers in the system will form a single connected component. However, just having connectivity among the honest peers does not seem to suffice to achieve the desired semantics of Join, Leave, and Lookup. The problem here is that in a Lookup(Name) execution adversarial peers may claim that a certain peer with that name is in the system, and there does not seem to be a reliable way for an honest peer to verify this claim because adversarial peers may represent the majority in the region containing $h(\mathsf{Name})$. So a different approach is needed to achieve survivability.

2.4 Outline of Trust-but-Verify approach

The main idea of our Trust-but-Verify approach is that honest peers will organize in regions $R \subseteq [0,1)$ they consider to be safe (i.e. the honest peers are in the majority) and not just some regions of size $(c \log n)/n$. If an honest peer feels that one of its regions is no longer safe, it will change it to a larger region. On the other hand, if an honest peer feels that a subregion within one of its regions is now safe, it will move to the subregion. Each honest peer will continuously probe peers it knows in its regions. If some peer does not respond, the honest peer will drop its connection to it. Since we assume honest peers to be reliable, this will not happen to honest peers. Messages are routed along safe regions to make sure that adversarial peers cannot alter, delay, or delete them. Next we give a more detailed description of how to maintain regions, how to interconnect the peers, and how to join, leave, and search in the overlay network.

3 The Trust-but-Verify scheme

3.1 Safe regions

Recall that a region is an interval in $[0,1)$ of size $1/2^r$ for some $r \in \mathbb{N}_0$ starting at an integer multiple of $1/2^r$. Suppose that every peer p knows the current number n of peers in the system. Given a peer p and a region R, let V_R^p denote p's *view* of R, i.e. the set of peers p knows in R. p considers R to be *safe* if, for some fixed constants $c > 1$ and $\epsilon \leq 1/12$,

1. $|R| \geq (c \log n)/n$ and
2. $|V_R^p| \leq (1+\epsilon)|R| \cdot n$.

Safe regions have the following nice property when given an ϵ-bounded adversary:

Theorem 2. *Let $0 < \epsilon \leq 1/12$ and $c > 0$ be constants. If for some peer p and region R, p considers R to be safe, V_R^p contains all honest peers in R, and c is sufficiently large compared to ϵ, then at least $3/4$ of the peers in V_R^p are honest, with high probability.*

The theorem directly follows from the well-known Chernoff bounds because if $|R| \geq (c \log n)/n$ for a sufficiently large c, then the number of honest peers in R is at least $(1-2\epsilon)|R| \cdot n$, with high probability. In this case, the fraction of adversarial peers in V_R^p can be at most $3\epsilon \leq 1/4$ if $\epsilon \leq 1/12$.

3.2 Maintaining safe regions

Since the membership of regions can change rapidly, each honest peer p will keep *safe snapshots* $(S_i(p), r_i^s(p))$ and *current versions* $(C_i(p), r_i^c(p))$ of regions containing $\mathsf{ID}(p) + \mathrm{sgn}(i)/2^{|i|}$, where $S_i(p)$ is the view and $1/2^{r_i^s(p)}$ is the size of the snapshot region $R_i^s(p)$ and $C_i(p)$ is the view and $1/2^{r_i^c(p)}$ is the size of the current region $R_i^c(p)$ of p. These regions are updated as follows.

1. A subregion $R \subseteq R_i^c(p)$ containing $\mathsf{ID}(p) + \mathrm{sgn}(i)/2^{|i|}$ is now safe: then p executes $(S_i(p), r_i^s(p)) \leftarrow (V_R^p, -\log|R|)$ and $(C_i(p), r_i^c(p)) \leftarrow (S_i(p), r_i^s(p))$, i.e. p updates its snapshot and current region to R.
2. $R_i^c(p)$ is unsafe: then p executes $C_i(p) \leftarrow C_i(p) \cup \mathsf{View}(R)$ and $r_i^c(p) \leftarrow r_i^c(p) - 1$, i.e. p doubles the size of its current region and extends its view by requesting a view of the unknown half, R, of the new region (the View command is specified below).
3. A new peer q joins $R_i^c(p)$: then p executes $C_i(p) \leftarrow C_i(p) \cup \{q\}$.
4. An old peer q leaves $R_i^c(p)$: then p executes $C_i(p) \leftarrow C_i(p) \setminus \{q\}$.
5. An old peer q leaves $R_i^s(p)$: then p executes $S_i(p) \leftarrow S_i(p) \setminus \{q\}$.

p maintains connections to all peers it knows of in every $R_i^c(p)$ and $R_i^s(p)$ (and some limited number of older versions of $R_i^s(p)$ to preserve the correctness of overlay network operations) and all peers q it knows of with p in $R_i^c(q)$ or $R_i^s(q)$.

Notice that there can be different values i_1 and i_2 with $R_{i_1}^c(p) \subseteq R_{i_2}^c(p)$, so regions may sometimes be contained in other regions, but for any two active regions R_1 and R_2 of a peer p it must hold that either $R_1 \subseteq R_2$ (resp. $R_2 \subseteq R_1$) or $R_1 \cap R_2 = \emptyset$. Those regions R_1 with $R_1 \subseteq R_2$ only have to be maintained implicitly, so that p only has to explicitly store $O(\log n)$ regions at any point in time.

Also, notice that $R_i^s(p) \subseteq R_i^c(p)$ at any time. New peers are only added to $C_i(p)$ because safe snapshots have to preserve the safeness condition at any time, and therefore new peers should not be added there. However, peers may still leave safe snapshots. Hence, they have to be updated quickly enough to make sure that honest peers always represent the majority. Fortunately, if the departure rate of honest peers is limited to $\epsilon/\log^2 n$ for some small constant $\epsilon > 0$, then the honest peers will be able to find new safe snapshots quickly enough. More precisely, the following result can be shown.

Theorem 3. *If the adversary is ϵ-bounded for some sufficiently small constant $\epsilon > 0$ and the arrival and departure rate of (honest and adversarial) peers is at most $\epsilon'/\log^2 n$ for some sufficiently small constant $\epsilon' > 0$, then our region update rules ensure that for every honest peer p and every i, at least $2/3$ of the peers in $R_i^s(p)$ are honest at any time, with high probability.*

To be able to verify this result, we describe below how to execute the $\mathsf{View}(R)$ operation. Notice that $R_i^s(p) \subseteq R_i^c(p)$ at any time. It is not difficult to check that this operation is well-defined and completes in $O(\log n)$ time.

```
View(R):
    // p: peer initiating the view request
    if ∃i ∈ ℤ : R ⊆ R_i^s(p) then return V_R^p (w.r.t. C_i(p))
    else
        if ID(p) ∈ R then
            // R_1, R_2: two halves of R, i.e. R_1 ∪ R_2 = R
            for each i ∈ {1, 2}:
                if ∃j ∈ ℤ : R_i ⊆ R_j^s(p) then V_i = V_{R_i}^p (w.r.t. C_j(p))
                else
                    fix any R_j^s(p) ⊆ R_i
                    send to all peers in S_j(p): (View, R_i)
                    wait until (View, R_i, V_i(q)) from ≥ 2/3 of peers q ∈ S_j(p)
                    V_i ← {q' : |{V_i(q) | q' ∈ V_i(q)}| ≥ |S_j(p)|/3}
            return V_1 ∪ V_2
        else
            send to all peers in the S_j(p) closest to R: (View, R)
            wait until receiving (View, R, V(q)) from ≥ 2/3 of peers q ∈ S_j(p)
            V ← {q' : |{V(q) | q' ∈ V(q)}| ≥ |S_j(p)|/3}
            return V

Upon receiving (View, R) from some peer p':
    // q: peer receiving the region request
    if ∃i ∈ ℤ : R ⊆ R_i^s(q) then send to p': (View, R, V_R^q)
    else
        if ID(q) ∈ R then
            // use same strategy as above to compute V_1 and V_2
            send to p': (View, R, V_1 ∪ V_2)
        else
            // use same strategy as above to compute V
            send to p': (View, R, V)
```

It remains to show how to execute Join, Leave, and Lookup operations and how to estimate n.

3.3 Join and Leave

If a new peer p wants to join via some bootstrap peer q, q starts with requesting views for regions of p of lowest possible size, i.e. $(c \log n)/n$. If this does not provide a safe region for some $S_i(p)$, q moves to a larger region for that i until it obtains views satisfying the requirements of a safe region for every relevant i (here, $i \in \{-\log n, \ldots, \log n\}$ is sufficient). Afterwards, q passes these views to p. p uses these views to integrate itself into the system.

The Leave(p) operation is straightforward: p just cuts all connections it has in the overlay network. Join takes $O(\log^2 n)$ time and Leave takes $O(1)$ time.

3.4 Lookup

Next we specify the Lookup operation. It is not difficult to check that it completes in $O(\log n)$ time.

```
Lookup(Name):
    // p: peer initiating the lookup request
    if ∃i : h(Name) ∈ R_i^c(p) then
        if ∃q ∈ C_i(p) : Name(q) = Name then return q
        else return NULL
    else
        send to all peers in the S_i(p) closest to h(Name): (Lookup, Name)
        wait until receiving (Lookup, Name, a) from > 1/2 of peers in S_i(p)
        return a

Upon receiving (Lookup, Name) from some peer p':
    // q: peer receiving the lookup request
    if ∃i : h(Name) ∈ R_i^c(q) then
        if ∃q' ∈ C_i(q) : Name(q') = Name then
            send (Lookup, Name, q') to p'
        else send to p': (Lookup, Name,NULL)
    else
        send to all peers in the S_i(q) closest to h(Name): (Lookup, Name)
        wait until receiving (Lookup, Name, a) from > 1/2 of peers in S_i(q)
        send to p': (Lookup, Name, a)
```

3.5 Estimating n

```
Size(r):
    // p: peer executing the operation
    R ← unique region of size r containing ID(p)
    (R_1, R_2) ← two halfs of R
    for each i ∈ {1, 2}:
        if ∃j ∈ ℤ : R_i ⊆ R_j^s(p) then v_i = |V_{R_i}^p| (w.r.t. C_j(p))
        else
            fix any R_j^s(p) ⊆ R_i
            wait until receiving (Size, v_q, r + 1) from ≥ 2/3 of peers q ∈ S_j(p)
            v_i ← median({v_q : q ∈ S_j(p) and q sent message})
    if r = 0 then return v_1 + v_2
    else
        R' ← unique region of size r − 1 containing ID(p)
        send to all q ∈ V_{R'}^p: (Size, v_1 + v_2, r)
```

Finally, we show how to estimate n. This is done recursively, starting with each peer p executing $\mathsf{Size}(r_0)$ for $r_0 = \lfloor \log(\frac{n}{c \log n}) \rfloor$. $\mathsf{Size}(r_0)$ never has to wait for incoming messages and therefore terminates for all honest peers. This will then allow $\mathsf{Size}(r_0 - 1)$-calls to terminate, which allow $\mathsf{Size}(r_0 - 2)$-calls to terminate, and so on until an estimate of the number of peers in the system is returned by $\mathsf{Size}(0)$. To make sure that this estimates n sufficiently well, only estimates from honest peers are accepted by using the median rule in each stage.

It is not difficult to show that these operations make the Trust-but-Verify approach weakly survivable.

References

1. B. Awerbuch and C. Scheideler. Enforced Spreading: An improved protocol for provably secure distributed name service. Technical report, Johns Hopkins University, February 2004.
2. B. Awerbuch and C. Scheideler. Group Spreading: A protocol for provably secure distributed name service. In *Proc. of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, 2004.
3. R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: Long-term protection against break-ins. *RSA CryptoBytes*, 3(1):1–8, 1997.
4. 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.
5. M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of the 2nd Usenix Symp. on Operating Systems Design and Implementation (OSDI)*, 1999.
6. S. Crosby and D. Wallach. Denial of service via algorithmic complexity attacks. In *Usenix Security*, 2003.
7. J. R. Douceur. The sybil attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. L. Lamport. The weak Byzantine generals problem. *Journal of the ACM*, 30(3):669–676, 1983.
14. 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.
15. 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.
16. 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.
17. R. De Prisco, B. W. Lampson, and N. Lynch. Revisiting the Paxos algorithm. In *Workshop on Distributed Algorithms*, pages 111–125, 1997.
18. 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.
19. 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.