Building Trusted Distributed Services Across Administrative Domains

Petros Maniatis, TJ Giuli, and Mary Baker
Stanford University
{maniatis, giuli, mgbaker}@cs.stanford.edu

Computer Science Department
Stanford University
Stanford, California 94305 USA

March 26, 2001

Abstract

In this paper we describe how to build a trusted reliable distributed service across administrative domains in a peer-to-peer network. The service we use to motivate our work is a public key time stamping service called Prokopius. The service provides a secure, verifiable but distributable stable archive that maintains time stamped snapshots of public keys over time. This in turn allows clients to verify time stamped documents or certificates that rely on formerly trusted public keys that are no longer in service or where the signer no longer exists. We find that such a service can time stamp the snapshots of public keys in a network of 148 nodes at the granularity of a couple of days, even in the worst case where an adversary causes the maximal amount of damage allowable within our fault model.

1 Introduction

Recent efforts in peer-to-peer networking [9][11][37][17][23] have demonstrated the feasibility of providing very large-scale distributed services. Different peer-to-peer networks provide various combinations of many valuable traits, such as anonymity, data permanence, efficient queries, authentication, and the ability to include participants from many different organizations and countries with no single central authority. This last trait is of particular importance to applications that offer reliability guarantees. As an example, the resistance of a document to censorship or our belief in the integrity of a document's content may be more achievable for political reasons when the network supporting the document storage and retrieval is composed of independent nodes in many countries with the nodes able to come to a majority agreement about the existence or contents of the document.

Yet without a single administrative authority or source of trust information about nodes, it remains difficult in these networks to build such services when the external information required for validating query results is no longer available. This could occur, for instance, if the validity of the document resulting from a query relies on a digital signature whose signing or verification key is no longer valid or available.

In this paper, we explore how to build long-lived reliable trusted services in a very loosely-coupled, distributed environment. In particular, we look at how to enable participating servers from diverse, independent administrative domains to implement a communal service that can be considered as a "trusted third party," even if not all participating servers are trusted by every user of this service. We also describe ways to preserve the continuity of such a service across membership or network changes over time as participants join, fail, or prove themselves untrustworthy.

The application we use to motivate this effort is a distributed secure public key time-stamping service, called Prokopius. Prokopius builds a secure, verifiable but distributable stable archive that maintains time stamped snapshots of public keys over time. This in turn allows clients to verify time stamped documents or certificates that rely on formerly trusted public keys which are no longer in service. Such functionality is essential when archiving digitally signed documents whose validity is dependent on an expired signature, not only after the signing key has changed, but even after the signer is no longer available.

While very useful, such a service would have to be very thoroughly trusted. To our knowledge, only centralized implementations of time stamping services exist today [35]. We believe that distributing not only the functionality, but also the organizational administration of such a service, can increase its ability to address the trust requirements of the diverse global digital community that the Internet has created. Although we motivate our work with Prokopius, the work is also valid for any service that requires the long-term archival of authenticated information.

We evaluate the agreement protocols used in Prokopius using a new simulator, called Narses, that simulates transport-layer flows, eliding individual packet information. This is faster than packet-level simulators [12] for large-scale networks, while still capturing the interdependencies of communications. Our results show that time stamping rounds on the order of a couple of days are entirely feasible in a network of 148 nodes, even where an adversary causes the maximum amount of damage possible within our fault model. Given the slow rate of change of public key information, this is appropriate for our application domain, although it may be too slow for other applications. While slower than a centralized service, the distributed service is more believable and more survivable, across both natural disasters and .com failures.

The overall contributions of this paper are

- The design of a secure distributed public key time stamping service that is survivable even across a complete change in service provider membership over time,
- A deployable implementation of protocols for Byzantine agreement across different administrative domains,
- The definition of ways in which these protocols can be used in the face of node membership changes and only locally managed trust relationships between nodes,
- A simulator based on transport-layer flows eliding individual packet information, and
- A performance evaluation of these protocols in the context of Prokopius.

The rest of this paper is organized as follows: In Section 2 we describe the current state of the art for time stamping services. In Section 3 we describe the design of our time stamping service and the assumptions we make regarding types of possible failures and attacks. In Section 4 we describe the component protocols we use as building blocks for our time stamping service, and the various assumptions they make regarding types of failures and the power of adversaries. In Section 5 we describe the Narses flow-based simulator. In Section 6 we evaluate Prokopius based on the time required to execute its core components, given the maximum damage an adversary may cause. In Sections 7 and 8 we describe related and future work, respectively. We conclude in Section 9.

2 Time Stamping State of the Art

A secure time stamping service (TSS) takes requests for digital time stamping from clients. A request consists of a hash of the document that the client wishes to time stamp. A TSS operates at some time granularity, called a *round*. For instance, Surety [35] provides one-second rounds, but many businesses require only weekly rounds. To complete a round, the TSS puts together the hash values for all documents it has received during that round, hashes them together, and produces a single hash value for that round. The result is used in the next round as one of the hashed values, to provide hash linking in the direction of time. That is, the round hash value for time step $t+1$ depends on documents submitted during step $t+1$ *and* the round hash value for time step t. Round hash values are available online, published on CD-ROM, aggregated in larger super-round hashes, for instance week-long super-rounds, and published in an independent secure publishing medium such as the NY Sunday Times.

Within a round, documents are linked to produce the round hash using one of various *hash linking schemes*. The first approach [18] was to link document hashes linearly. However, to verify that a document A belongs in a round when using linear hash linking, it is necessary to have document hashes for all documents following A in the round, so that one can compute the hash chain from A to the published (and verifiable) round hash. A more efficient method first proposed by Benaloh[2] is to link the document hashes from within a round into a tree structure (also known as a Merkle tree [26]). Using tree hash linking, to verify that a document A belongs in a round, one only needs the document hash and the hashes of its ancestors' siblings in the hash tree. Therefore, one only needs to remember roughly log(n) values (for n documents within a round) to prove to someone that there is a way to get from document A to the published round hash. Other optimized schemes, providing varying levels of accountability versus storage cost trade-offs have been proposed as well (anti-monotone and simply connected schemes [3]). In this paper, we assume the use of tree hash linking.

One possible vulnerability of the schemes described above is that a centralized digital TSS could collude with the publisher of the round hashes. A centralized digital TSS provides accountability by collecting together documents it has time stamped within a day and hashing them together. The resulting round hash is then published in a supposedly independent and secure publishing medium. Once the service has published a round hash in a secure medium, it cannot insert or remove documents from the round, since that would cause the round hash to change. If, however, the service were to collude with the publisher, then they could insert or delete the document, and then change the published round hash. This would be feasible if the publication medium is something like a web page, which is not static and survivable, and can thus be altered.

The distributed time stamping service described in this paper can help provide more accountability even for such a centralized service. In our scheme, the group of independent distributed time stampers act as the secure publishing medium. For a single time stamper to insert into or delete from its round a document, it would have to convince the rest of the group to alter its published round hash for that round, which should be much harder than trying to convince a single entity to do so, especially since that single publisher might share failure modes with the TSS (for instance, they might be owned by the same corporation or amenable to legal coercion by the same authorities).

Although this paper describes a TSS for public keys, it is trivial to extend it to a TSS for other documents. Individual nodes may time stamp documents submitted to them without consultation with other nodes. Every once in a while, these nodes all submit the round hashes of their individual time stamping efforts to the group. The hash of all these values is then included as a sibling node to the root node of the round hash for the current public key time stamping round. This means that documents time stamped at the edge of the network can later be authenticated even if the document time stamper's public key has changed or the document time stamper itself has ceased to exist. No centralized time stamping service can afford to make this guarantee.

3 Design

In this section we describe the overall Prokopius system, the assumptions we make regarding infrastructure services, the assumptions we make regarding the behavior of nodes and adversaries, and the responsibilities of individual nodes. We then give the design of the overall public key archival (time stamping) procedure. For convenience, Table 1 summarizes the various parameters mentioned in this section.

3.1 Prokopius System

Prokopius is composed of independent nodes belonging to different organizations and preferably located in many countries around the world. Nodes are likely to belong to organizations such as notary services, government bureaucracies, and legal firms that have a vested interest in the validation of contracts and documents over time. A reasonable scale for such a system might be anywhere from forty to a couple of hundred nodes. The more nodes there are in the system, the harder it is for an adversary to compromise a sufficiently large number of them to influence the correctness or viability of the system. However a larger number of nodes also takes longer to perform time stamping.

3.2 System Assumptions

Prokopius is an asynchronous distributed system of nodes with the following assumptions regarding the network and failure model (which are very similar to the assumptions made in some other practical fault-tolerant systems such as [8]). The network itself may drop messages or reorder, duplicate, or delay them. The failure model is Byzantine, meaning faulty nodes may behave in any manner (with only a few exceptions, listed below). However, the adversary is computationally bound. It cannot, for example, find collisions of a collision-free hash function, or compute discrete logs. More specific assumptions include the following:

First, as mentioned above, we assume that correct nodes are independent of each other, since this increases the difficulty for an adversary to influence the outcome of group decisions. This is the reason why nodes should be in different administrative domains, countries, or other independent organizations. Although complex failure dependencies are now being considered by the research community (e.g. [6, 25]), a satisfactory theory behind inferring arbitrary failure dependencies has yet to be defined.

Second, we assume that out of the N nodes in the system, less than a third of them will be faulty (broken or malicious) within a single time stamping round. That is, if there are F faulty nodes in the system, then $N \geq 3F+1$ for the duration of a round. This is a well-known theoretical upper limit of the maximum tolerable node misbehavior in a Byzantine setting [24]. It is common to assume this upper bound on faulty nodes, and both the safety and the liveness of the agreement protocols we use depend upon it. However, it is clear that in exceptional cases involving extremely powerful adversaries even such a well-worn assumption can be hard to maintain.

Third, we assume that a well-known mechanism for authenticated communication among the nodes is in place. This mechanism may rely on any particular coexisting infrastructure service (e.g., Verisign, Entrust, DNSSEC), or be *ad hoc*, based on a web-of-trust paradigm, or even effected through direct key exchange. This does not mean that the nodes agree on whom they trust. It simply means that correct nodes should be able to retrieve an up-to-date public key for a particular other node in the system.

Fourth, we assume that a node will eventually receive messages sent to it during a round, since messages are resent until acknowledged in an authenticated manner. If an adversary prevents all copies of a message from reaching a node during a round, then we consider the sender of that message to be a faulty node, constrained to the upper limit F mentioned above. Large-scale network outages, partitions, and so forth may be caused by the adversary, but do not persist for the duration of a protocol round. Similarly, no adversary-driven denial of service attack can last longer than a round. Our timing assumptions are similar in vein to those made by other practical Byzantine fault-tolerant systems[8], although we feel they are easier to justify in the less performance-driven context of our application.

Fifth, we assume that all correct nodes have local time sources that lie within $\frac{T}{2}$ of every other correct node's time source, where T is the duration of a time stamping round.

3.3 Node Responsibilities

Besides overall system assumptions, the nodes in Prokopius have individual local responsibilities and information they must maintain.

Each node has two basic local responsibilities. First, each node must independently maintain a time source. It could get its time readings from an atomic clock, for instance, or a radio clock or GPS clock. (Note that a time source such as GPS is actually dependent on the U.S. military, and thus may not be considered trustworthy by everyone in the world.) The node must maintain its realtime clock such that its drift is no more than $\pm\frac{1}{4}T$ of the actual current time. (Note that this requirement should also address the previously described assumption of clock consistency among nodes.) The nodes should not synchronize their clocks with each other, since this would violate the independence assumption described above. Although we never know exactly what the real time is, we need to remain close to it, so that the entire system does not drift over time. For example, if in ten years we ask the system whether a certain public key was valid on the 26th of March in 2001, we do not want the query submitter and the system to have very different ideas about what time the 26th of March 2001 actually means. Solutions such as the Timely Computing Base proposal in [36] might help render this assumption practical in all settings.

Second, each node should maintain an independent local set of trust relationships such that there is a well-established way for the node to answer the binary predicate *trusted(node)* for any node in the system or outside it. (That is, the node must be able to answer "yes" or "no" when asked if it trusts any other particular node.) A simple approach might be for a node to trust only those other nodes that it believes belong to reputable companies with whose administrators it has securely exchanged public keys for the nodes. Other options include using a hop-count based threshold scheme in a web-of-trust or a similar scheme that considers the number of intermediate certificates in a DNSSEC certification chain.

In addition, during a time stamping round, each node must maintain the following state:

- Current membership. This is an ordered list of nodes, including itself, participating in the current round of the system. The order of the nodes is determined when the membership is set, and is identical across all nodes. Once the membership is known, each node knows the total number of nodes, N, in the system, and can derive from that the maximum number of tolerable concurrent failures, F.

- Current round real time. This is the real time, R, associated with the current round. This association is determined in the time stamping protocol described below.

- The time stamping history developed in the public key snapshot time stamping step, as described below.

Parameter	Purpose	Comment
N	Number of nodes in the system	Agreed upon with every membership agreement
F	Number of tolerable concurrent failures	Less than N/3
T	Time granularity of time stamping rounds	A parameter of the protocol
R	The "real time"agreed upon by nodes	Becomes the name of the next round
X	An individual node's view of the real time	Submitted to the agreement protocol

Table 1: The system parameters

3.4 Public Key Archival Protocol

At regular intervals, whose length is specified *a priori* for the system, a node participates in a round change. The length of the round determines the time-granularity of the time stamping performed by the system. Besides time stamping, at round changes the group of nodes participating in the system perform archive maintenance, modify current membership, and prepare the onset of the next time stamping round. Specifically, at every round change every node must execute the following steps:

1. *Synchronization.* In this step, the nodes come to an agreement that it is time to begin a new round. Each node waits until its local round timer expires. This timer is set to expire time T after the beginning of the current round according to the node's local clock. When the round timer expires, the node broadcasts a message including the identifier of the current round to all other members of the group, and listens until $N - F$ such broadcasts from individual fellow-members are received. Note that the node listens for round expired broadcasts even before its timer expires, to catch broadcasts from those nodes with faster clocks. This broadcast does not need to be a consistent broadcast, but it must be authenticated, and thus we depend on the Transport Layer Security (TLS) protocol [33] described in Section 4.

2. *Checkpointing.* Every node takes a snapshot of the public keys of the entire group. This snapshot is agreed upon as the authoritative view of the group's public keys at the current time and signed by everyone in the group. It is then added to the public key archive maintained and distributed by the group. In this way, the group lends its trustworthiness and continuity assurances to the time stamped view of inherently transient information that is contained within the public keys of the group's current nodes. This allows the time stamped information to remain verifiable, even after all of the current nodes in the group are no longer providing this service. This is a guarantee that no centralized time stamping service can afford to offer.

   This step is performed in two phases at every round:

   (a) Every member puts the current public keys of the group together in a hash-linked tree structure, ordering them based on a well known total order. Since the membership is known to everyone, correct nodes should have identical tree structures. The hash of the root of the tree represents the entire structure in the next phase.

   (b) The root hash of the resulting key tree from the previous phase, along with the root hash from the previous time stamping round, are combined, time stamped and inserted in another Merkle tree structure. This structure contains all the root hashes of all key trees that the group has time stamped over time, ordered by time stamp. This "super root hash" is signed by at least the $N - F$ correct group members, and is thereafter freely distributed in response to queries. Note that no perishable digital signature is contained within any of the two trees. The integrity of the trees is safeguarded by the collision-resistance of the used hash function. The only signed information maintained (which needs to be re-signed every time a key changes) is the root of the entire double-tree structure.

   In this step we make use of our assumption that participating nodes have consistent public key snapshots, and we make use of our variant of Merkle trees, described in Section 4.

3. *Secure Shared Coin Redistribution.* The purpose of this step is to derive a new secure shared threshold coin, for the next round. This cryptographic construct is used to provide the randomization in the particular Asynchronous Byzantine Agreement protocol we use [5] in later steps. This step makes use of the coin derived in the the previous round, as well

as the facilities for signing, authenticated transport and verifiable consistent broadcast that we describe in Section 4. The scheme requires every participant to act as the dealer of a secret, and split that secret into one share for each other participant [34]. Each participant then proceeds to broadcast a number of verification values to every other participant, and then encrypts and sends each secret share it computed to the appropriate participant. The recipient of a secret share verifies it using the previously broadcast verification values, keeping track of those secret shares that did not verify accurately.

Throughout the process, a node that realizes it has received a secret share that fails verification, broadcasts a "complaint" against the dealer who generated that share. The dealer then must prove to everyone that the share was correct, by revealing it and showing it can be verified by the public verification values, or risk being excluded from further rounds of the system, as will be seen in the next step.

4. *Membership change.* The purpose of this step is to determine the membership of the group for the next round. Verifiably faulty nodes may be rejected from the membership, and new nodes may join by being endorsed by trusted members. In this step we make use of our assumptions of a consistent view of public keys and local trust information at each node. We also make use of the Asynchronous Multivalued Byzantine Agreement protocol and Digital Signing from Section 4.

   During membership change, every node declares its trust or lack thereof in the "correctness" of each of its fellow members, taking into account its local trust information, which includes information it has gathered in the course of executing the protocols concerning the node's ability to meet expectations concerning maximum time skew, minimum bandwidth and computation capabilities, and so forth. In this consideration, nodes also take into account information garnered during the redistribution of the new secure shared coin keys. That is, if some node received more than F complaints from other members of the group, because it sends node coin key shares that cannot be verified against the public verification values, then other nodes may decide to remove the sender from the next membership.

   Nodes do this by broadcasting a signed bit vector that contains a 1 in the bit position of every fellow member they no longer trust, and a 0 in the bit position of every fellow member that they have nothing against. Along with this bit string, nodes send an unsorted list of new nodes wishing to join the group, whose membership the sender endorses (i.e., the sender trusts the nodes it suggests letting in, again based on local information). We call this bit string and list of new nodes the *membership preference* of each node.

   Every node waits to receive N-F such signed preference messages. Once it has collected N-F preferences from distinct fellow members (including its own), it processes them to produce a valid new membership proposal that would be acceptable to all those whose preferences it holds. N-F represents the least number of answers that a node can expect to receive with certainty from honest members of the group within the round.

   To produce a verifiably valid proposal, each node must be supported in its opinion about any other node by N-2F similar opinions. That is, for node removals, a node places a 1 in the position of the bitvector in which at least N-2F preferences contained 1's. Similarly, a node addition must be supported by at least N-2F group members.

   Another restriction imposed on valid membership change proposals is a maximum number of F changes between any consecutive group memberships. The objective of this restriction is to make sure that any consecutive group memberships contain enough correct nodes in common to yield a majority that would supersede up to F malicious nodes acting incorrectly in concert. This is a common requirement in Byzantine-fault masking systems and is similar to the Byzantine-fault masking quorum systems of [25].

   Once it has constructed a valid proposal, a node participates in the Multivalued Byzantine Agreement protocol. When the agreement protocol terminates, the node accepts the agreed upon group membership as the makeup of the group for the next round. Honest nodes who receive an agreement that does not include them are expected to go away, and are ignored by others in deliberations associated with the next round. Nodes who just joined may request a current copy of the state (the history trees) from any participant, since the state is verifiable given the current signed history root hash. Note that since we have required that no more than F changes occur between successive group memberships, any newcomer can trust the collective signature on the previous history root hash, since it has at least N-F signatures in common with its current group.

Once the new group membership has been established, a member that belongs to the new group can put together its new secret coin scheme: it is the (modular) sum of all secret shares sent to it by all current members of the group.

5. *Derivation of new round name.* In this step we determine a real time association for the next round. We make use of our assumption of a local clock at each node, as well as the Multivalued Byzantine Agreement and Digital Signing protocols described in Section 4.

   (a) Every node broadcasts its signed opinion on what the real time association should be. Correct nodes pick their real time as that opinion. Given our clock correctness assumptions, no two correct nodes' opinions on the real time are farther apart than T/2.

   (b) When a node receives N-F properly signed opinions (that fall within an interval T/2 centered at the node's opinion of the correct time) from distinct fellow members, it determines the mean of all opinions, and proposes this value, X, to the validated multivalued asynchronous Byzantine agreement protocol with tag (round#, X). The validation of the proposal is the concatenation of all opinions used to derive that proposal by the node, along with the associated signatures.

   (c) The node then waits until agreement on a proposal, validated by a collective signature, is reached. This proposal, R, is the real time, or name, associated with the next new round.

To bootstrap this protocol, for the initial round, we must agree on a group membership through some external means. This might be a political agreement such as allowing each interested government to include a node in the system.

4 Component Protocols and Data Structures

In this section we describe the individual components we use as building blocks for our time stamping service. We list the assumptions made by these protocols and data structures and explain their general purpose.

4.1 Authenticated Transport Layer

We base communications on an authenticated transport layer, such as that provided by Transport Layer Security (TLS), which is a minor modification of the Secure Socket Layer (SSL) [33]. We use this protocol to provide authenticated communication between two nodes. As long as a node can verify the public key certificate sent to it by its counterpart (using local certification authority certificates), two nodes can communicate in an authenticated manner. Generally, we use only the Message Authentication Codes (MACs) provided by this protocol, and not signing or encryption. The exceptions are session key establishment and encryption of new coin generation keys which must be distributed when node membership changes.

The assumption made in our use of this protocol is the existence of a mechanism for nodes to acquire the public keys of other nodes. Our goal is to provide permanent authenticated archival of these keys, but we do not ourselves provide that public key infrastructure.

4.2 Verifiable Consistent Broadcast Facility

We require a verifiable consistent broadcast facility, in this case, the echo-broadcast facility used in [32]as reformulated in [4]. The protocol is verifiable in that it allows a sender to broadcast a payload to a group, and compose a *proof* that a sufficient number of group members has actually received the payload. It is consistent in that any group members receiving the payload will all receive the same payload. Note that the protocol is not considered reliable, in that not all group members are guaranteed to receive the payload, but merely a sufficient number. This sufficient number is derived from the overall number of faulty nodes we can tolerate. Specifically, if we have N total nodes and can tolerate no more than F failures, then a sufficient number is $\left\lceil \frac{(N+F+1)}{2} \right\rceil$.

This component uses digital signatures on a digest of the broadcast payload. It is not sufficient to rely on the underlying TLS mechanism for authenticity, because we wish to be able to prove subsequently that a broadcast payload was seen by the group, even after the sessions used to transport the payload back and forth are long gone.

4.3 Digital Signing Facility

We use standard threshold signature schemes, whereby to be considered *signed*, a message must be individually signed by a number of group members greater than or equal to a set threshold (which is generally the total number of nodes minus the maximum number of failures we can tolerate).

We use this facility for all of our broadcast and agreement protocols for simplicity, even though a more efficient implementation might employ a distributed shared signing facility, as seen in the protocol formulations in [4]. In addition, although distributed shared signing is efficient to use, it generally assumes that group membership does not change, and we cannot make that assumption. If membership changes, we must negotiate new signing keys (the secret share-signing key, the share verification keys, the signature verification keys, and so forth), which can become very expensive. Ways to perform such renegotiation without resorting to a trusted "key dealer" for different public key cryptosystems have been described [16, 10], but we have not yet explored their feasibility for large groups of nodes.

4.4 Asynchronous Multivalued Byzantine Agreement

The objective of this component is to allow a group of participants to come to agreement on a particular decision, even in the face of some malicious participants. It has been theoretically proven that agreement is possible only when the malicious participants are fewer than a third of all participants [24].

We use the validated Byzantine agreement protocol of [4], which is based on an asynchronous binary Byzantine agreement protocol [5] derived from the seminal randomized agreement algorithm by [1]. The randomization is essential in addressing the attack of the "aloof malicious participant". In this attack, malicious parties participate in the protocol but over time influence membership changes to include more malicious participants gradually. When the number of malicious participants becomes large enough, the assumptions behind the Byzantine agreement protocol become false, and the correctness of the protocol fails.

The solution to this problem in practical systems that use deterministic protocols [8] is proactive recovery: every once in a while, we go around and "flush" out all nodes, reloading their state and software from a trusted source. This approach indeed helps us approximate the assumption that no more than a third of nodes are malicious participants, but this only works against external intrusion or corruption, and only in a setting where all nodes belong to the same organization or at least to cooperating administrative organizations. When participating nodes belong to independent organizations, some of which might wish to be malicious, such proactive recovery or software rejuvenation measures do not work.

The randomized algorithm circumvents this problem by using a coin that is guaranteed to be independent of what the adversary wants whenever nodes need to break a stalemate while coming to an agreement. This property is based essentially on the random number generator described next.

4.5 Distributed Shared Random Number Generator

A *secure shared coin flipping* scheme forces even a malicious node to commit to a particular random number without yet knowing what that random number is going to be. It is this feature that makes it hard for a malicious node to influence the outcome of the agreement protocols we use.

Our coin flipping scheme draws its security from the assumed intractability of calculating discrete logs in polynomial time, and it is based on an idea first explored in [27], further fitted to our communications setting in [5]. To reinitialize after a group membership change, we use a distributed secret key generation mechanism for discrete-log based signing systems, originally proposed by Pedersen [28], but most recently amended by Gennaro [16]. Gennaro's amendment strengthens the original scheme to prevent an adversary from biasing the negotiated secret key. To our knowledge, this is the first time this scheme has been used to redistribute coin share-signing keys in such a coin flipping system.

4.6 A Variation of Merkle Trees

The objective of this round is to lend the trustworthiness of the current system membership to a time stamp on a snapshot of the group's nodes' public keys. To do this, we leverage the assurances that existing public key mechanisms offer for the here and now, to produce secure archives of timed snapshots of their contents.

In our time stamping protocol, we use a variation of hash trees derived from Merkle trees [26]. Merkle trees are very convenient in that they allow the computation of a "proof" of the existence or not of a particular datum in them, and that proof is easy to compute (just a sequence of hash operations), fairly compact (it grows roughly logarithmically with the number of data items in the tree, and cannot be falsified (as long as the hash function is collision-resistant). If the datum exists inside the tree, this proof is the hash of every internal tree node that is a sibling of an ancestor of the sought datum in the tree, along with the root of the tree, whose integrity the time stampers are charged with maintaining. If the datum does not exist in the tree, the proof is similar to the existence proof above, except it concerns data items in the tree that would precede and follow the sought datum in the well-known total ordering, showing that there is nothing in between.

Another very convenient property of Merkle trees is that they ensure the integrity of the ordering and contents of the tree structure, without requiring that requesters communicate with a trusted party for anything but the root hash. In fact, totally untrusted storage servers can serve this information to clients without even bothering the time stamping system described above, as long as the current system members all sign the current historic root hash and distribute it through the untrusted storage servers, along with the rest of the two trees. As long as a client can verify the signatures on the root hash, it can verify the validity or not of any existence or non-existence proof of a datum in a Merkle tree.

We make a number of optimizations of the original Merkle scheme to reduce the storage complexity of the archive. We combine the original Merkle tree structure with balanced trees to ensure that tree depth remains close to the logarithm of nodes within it - which means that proofs of data existence or non-existence remain logarithmic in size. Furthermore, we employ version trees, which allow copy-on-modify behavior for tree nodes: if the Merkle tree rebalancing operation causes an internal node's hash value to be recalculated, then a new copy of that node is stored. Otherwise, the already stored node is pointed to by a new ancestor.

5 The Narses Simulator

The goal behind the Narses simulator is to simulate a large number of nodes over very long periods of time (on the scale of months) and with highly variable flow sizes. We simulate communication at the flow level, since the computational complexity of packet-level simulators (such as ns [12]) is too high for our simulations to run in reasonable time. We wanted a scheduler that could approximate the accuracy of packet-based simulators without the processing time required to consider events on a packet level. To this end, we have attempted to build a flow-based simulator that elides individual packet information but still reflects the impact of traffic interdepenencies. The result is a compromise between speed and accuracy, but one that consistently errs on the side of pessimism (the simulation results report a time equal to or greater than what would be consumed in practice.)

Because we make the assumption that all nodes participating in the time stamping service are well-connected, we can reduce the computational complexity of the simulations by assuming that only source and destination links limit the bandwidth of a flow. In other words, there are no bottleneck links between source and destination. This allows us to simulate our network topology as a star where every node in the network is connected with a central node that has a direct connection to every node and infinite bandwidth on each link.

The simulator is modeled after the traditional layered network stack, but we lump the physical, link, network, and transport layers into one layer. The overall behavior of all of these layers acting in unison is modeled using an algorithm we call *fair flow scheduling*, where we define a flow to be all of the bytes of a message passed from a higher layer to the transport layer.

Fair flow scheduling allocates bandwidth to flows only up to their *fair share*, even if the flow could use more than its fair share. The fair share bandwidth is the bandwidth of a node divided by the number of flows sent or received by that node. Where a flow moves between two nodes of different bandwidth, the bandwidth of the flow is determined by taking the *minimum* of the fair share bandwidth of the flow's source and the flow's destination. For example, if a 10Mbps node with four current flows initiates a flow to a 100Mbps node with three current flows, the allocated bandwidth is Min(100/(3 + 1) , 10/(4+1)) = 2Mbps. Even though the 100Mbps node could devote 25Mbps to this flow, the flow only takes up 2Mbps. In practice this means that some of the other flows on the 100Mbps node could utilize more than 1/4 of the bandwidth, but in our simulation, we pessimistically leave them with their fair share (bandwidth divided by number of flows).

Although we built the Narses simulator for evaluating the protocols that comprise our time stamping service, we believe the simulator may be useful for evaluating other large-scale long-running network activities. In separate concurrent work we are evaluating the accuracy and speed of Narses when compared with packet-level simulation. The simulations reported in Section 6 all together took a total time of about 45 minutes running on a dual Pentium 3 Xeon at 1 GHz per processor and one Gigabyte of main memory.

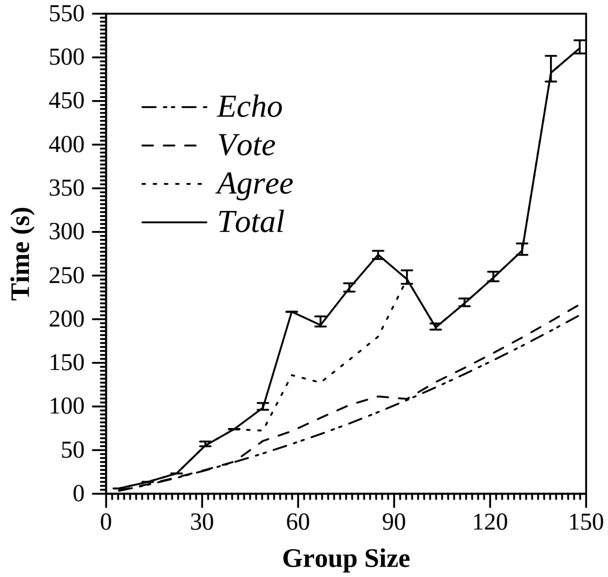

Figure 1: This graph shows the increase in the completion time of the different phases of a single multivalued Byzantine agreement protocol instance. The lowest curve shows the proposal broadcast (echo) phase, The second lowest curve shows the additional time for the vote broadcast phase. The third curve shows the additional time to complete a single binary agreement. The top curve shows the total time taken by the multivalued agreement.

6 Evaluation

In this paper we seek to test whether a distributed trusted service using randomized Byzantine agreement can be realistic for our application domain. The slow rate-of-change of information inherent in public key time stamping allows us to use methods that might otherwise be inappropriate.

The basic building block used during every round change is the Asynchronous Multivalued Byzantine Agreement component, which is invoked by members three times to agree on the new group membership for the following round (and by extension, on the exact secure shared coin to be used therein), and to agree on the real-time association of a round, (the time that will be included in every time stamp issued during the next round). The agreement has several phases, namely the *echo* phase, where nodes consistently broadcast their validated proposal, the *vote* phase, where nodes broadcast their yes/no vote on a particular proposal by a specific node, and the *agreement* phase, which is the phase during which nodes attempt to reach consensus on whether or not to use a particular validated proposal or not.

Figure 1 presents the results of our simulations. This graph shows the increase in the completion time of the different phases of a single Multivalued Byzantine Agreement protocol instance as the group membership size increases. The maximum group membership size simulated is 148 nodes. The lowest curve shows the echo phase, which happens exactly once per agreement. The second lowest curve shows the additional time taken for the vote broadcast phase, which is much shorter than the proposal (echo) broadcast phase, because it does not require a consistent broadcast. The third curve shows the additional time taken to complete a single binary agreement. The top curve shows the total time taken by the multivalued agreement. The discrepancy between the total time and the time to complete the binary agreement is due to the fact that the vote and binary agreement phases may happen more than once, depending on the proposal chosen to vote on. Since the adversary knows in advance in which order the proposals are voted on, it can schedule the corruption of the first F proposers, causing the eventual agreement to delay due to repetition of the voting and binary agreement phases.

However, the adversary cannot cause more than $2F$ such repetitions [4]. Given 148 nodes, this means (2 * 49) total possible repetitions of the voting and agreement phases for a single multivalue agreement. From the graph, we see that these phases together account for about 510 seconds for a single repetition. Adding the 204 seconds required for the echo phase, a single multivalue agreement may take about 14 hours. Since three such multivalue agreements are required for one round, this means the adversary can cause a time stamping round to take about 42 hours, or almost two days.

This is indeed a worst-case scenario, for two reasons. First, although we have not done so, the membership agreement and real-time association agreement steps of the time stamping protocol may in fact be combined, requiring only two multivalue agreements per round. Second, Cachin reports a protocol with a constant number of expected rounds [4] for multivalued agreement which also relies on randomization. However, we have not yet implemented or simulated this protocol.

Thus it is reasonable to state that a single round of the time stamping protocol for 148 nodes with an adversary causing maximal damage can still be performed in a matter of a couple of days.

7 Related Work

Although time stamping seems to be essential to conduct secure transactions with lasting effects in the digital world of the Internet, the number of researchers exploring this topic is surprisingly small. In addition to the time stamping work described in Section 2, most notably Just [20] describes the intricacies of relying on the trustworthiness of the time stamper to ensure the immutability of the document time line, and ways in which earlier work fails to make time stampers accountable. This work, although not directly applicable to the core time stamping described in this paper (where relative order of public keys within a round is insignificant), is very important for the operation of the potential document time stamping services run by the "edges" of our network, by the group member nodes individually. A very detailed specification of a time stamping service has been produced in the TIMESEC project [30].

Practical systems that tolerate Byzantine faults have been proposed and implemented in the past (e.g., [7, 32]). However, only deterministic approaches have been used, due to the fact that randomization was traditionally considered expensive to use. To circumvent the impossibility result of [14], various avenues are followed, including the use of failure detectors that are very hard to build and prove correct in the case of systems where group membership must change, or the imposition of a static membership that is preconfigured by the system's administrators. Unfortunately, we do not have the luxury of knowing well in advance what the projected membership of our system's group will be far into the future, nor can we assume that a single administrative organization can be charged with maintaining this scheme; in fact, we most certainly do not want a single organization to perform this task all by itself. It seems that in such a setting, where we can use neither proactive recovery [8], nor a static group membership, randomization is the only way to go. Fortunately, the timing requirements of our application are lax enough that we can employ the more expensive randomized agreement techniques described here without significant service quality reduction. Recently, Cachin [6] described an architecture for secure and fault-tolerant service replication on the Internet using the basic components we use in this work, although the work relies on the group membership being static.

Other researchers have explored techniques for extracting out detail and hence reducing computation complexity for simulations of large-scale networks. Flowsim [15] abstracts away some of the detail of packet-level simulators by grouping packets with closely spaced send times together into a single event called a packet train. This is motivated by the observation that links on real networks are frequently occupied solely by packets from the same flow.

In addition, [19] develops two abstractions for simulating large scale mutlicast networks. The first abstraction is to eliminate maintenance messages that maintain the multicast tree. The second abstraction, and the one most related to our work, is that they do away with hop-by-hop routing and instead schedule packet delivery directly from source to destination. The key difference is that they do not take queueing delay into account, which means that cross traffic does not affect the transmission of packets. We take into account queuing delay by dividing up bandwidth between concurrent flows in or out of the same node.

8 Future Work

There are several avenues of future work for us. Some obvious areas proof of correctness of the protocols, further validation of the Narses simulator, and evaluation of a reasonably-sized deployment of Prokopius.

While Prokopius is merely a TSS for public key information, as described in Section 2 it is trivial to extend it to an arbitrary document time stamping service. In this sense, Prokopius becomes the secure publication medium for locally provided time stamping services that can lie at the edge of the network. We plan to extend Prokopius in this way, since we require a document TSS for the following project.

Prokopius is but one component of a larger project focused on providing *histories of online names.* An online name might be an email address, instant messaging identifier, or perhaps even a telephone number. It is likely to be an identifier that works over several applications or online media. Such names are useful for several purposes, including contacting people or assigning identities to online contracts. While there are several projects and companies that aim to provide unique online identifiers for people ([31, 21] or `www.onename.com`), we recognize that people's online names are likely to change due to changes in employment or geographical address or because the companies hosting the online names go out of business or themselves change names. Knowing an old online name for a person is thus no help in reaching that person if there is no way to find the new name from the old one.

Assuming anyone wishes to do so, our naming history service allows people to provide an authenticated history of their online names, which is a list of signed mappings between each name and the next most recent name. When given an old name (and approximate time frame during which that name was valid), the history service returns the newest online identifier for that person. The history service is an example of authenticated archived information. Mappings between an old name and a newer name in the service may be authenticated using a public key that is no longer valid. The time stamping service is thus an essential component that allows us to validate the authentication of each link in a person's naming history, and we would like to finish building the history service and evaluate Prokopius in that context.

9 Conclusion

In this paper we show that it is possible to build secure reliable and survivable services in a peer-to-peer network using Byzantine fault-tolerant protocols. Prokopius is able to offer public key time stamping rounds on the order of a couple of days for a 148-node network even in the case where an adversary causes the maximum damage allowed within our fault model. Rounds on the order of a couple of days, or even on the order of a week, are sufficient to support applications, such as archival of public key snapshots, where the information itself changes slowly.

In contrast to a centralized time stamping service, the distributed service can survive changes in public keys and even a complete change in service provider membership over time. The service can validate documents signed in the past with keys no longer in service and by entities that have ceased to exist.

This public key time stamping service can also form the core of a time stamping service for other kinds of documents, also built in a peer-to-peer network. In this case, individual nodes in the network may time stamp the documents submitted to them without consultation with other nodes, and thus with only local overhead for each time stamp. Every once in a while, these nodes all submit the round hashes of their individual time stamping efforts to the group for agreement.

10 Acknowledgments

This work has been generously supported by a gift from NTT Mobile Communications Network, Inc. (NTT DoCoMo), a USENIX Scholars Fellowship for Petros Maniatis, the Stanford Networking Research Center, and DARPA contract N66001-00-C-8015.

References

[1] Ben-Or, M. "Another advantage of free choice: Completely asynchronous agreement protocols." In Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing (PODC), 1983.

[2] Benaloh, J. and deMare, M. "Efficient Broadcast Time-Stamping." Clarkson University Department of Mathematics and Computer Science Technical Report number TR-MCS-91-1. April 1991.

[3] A. Buldas, P. Laud, H. Lipmaa, J. Villemson. "Time-Stamping with Binary Linking Schemes." Advances in Cryptology CRYPTO '98, ed. H. Krawczyk, vol. 1462, LNCS, pp. 486-501, Springer-Verlag, 1998.

[4] Cachin, C., Kursawe, K., Petzold, F., Shoup, V. "Secure and Efficient Asynchronous Broadcast Protocols." Research Report RZ 3317, IBM Research, March 2001.

[5] Cachin, C., Kursawe, K., and Shoup, V. "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography." In Proc. 19th ACM Symposium on Principles of Distributed Computing, Portland, OR, pp 123-132, July 2000.

[6] Cachin, C. "Distributing trust on the Internet." In Proc. Intl. conference on dependable systems and

networks (DSN-2001), Gothenborg, Sweden, June 2001. To appear.

[7] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance." In Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, New Orleans, LA, February 1999.

[8] M. Castro and B. Liskov, "Proactive Recovery in a Byzantine-Fault-Tolerant System." In Proceedings of the 4th Symposium on Operating Systems Design and Implementation, San Diego, CA, October 23-25, 2000, pp. 273-287.

[9] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System." In Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, ed. by H. Federrath. Springer: New York (2001).

[10] Damgård, I., Koprowski, M. "Practical Threshold RSA Signatures without a Trusted Dealder." To appear in Proc. of EuroCrypt 2001.

[11] R. Dingledine, M. Freedman, D. Molnar. "The Free Haven Project: Distributed Anonymous Storage Service." In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, July 2000.

[12] K. Fall and K. Varadhan, editors, "ns notes and documentation." The VINT Project, U.C. Berkeley, LBL, USC/ISI, and Xerox PARC, July 1999. Available from http://www-mash.cs.berkeley.edu/ns/.

[13] P. Feldman. "A practical scheme for non-interactive verifiable secret sharing." In proceedings of the 28th Foundations of Computer Science, 1987, pages 427-437.

[14] Fischer, M., Lynch, N., Paterson, M. "Impossibility of Distributed COnsensus With One Faulty Process". Journal of the ACM, 32(2), 1985.

[15] J.-S. Ahn, P. Danzig, D. Estrin, and B. Timmerman. "Hybrid Technique for Simulating High Bandwidth Delay Computer Networks," In Proceedings of ACM SIGMETRICS, ACM Press, New York, 1993, pp. 260-261.

[16] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin, "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems." Proceedings of EUROCRYPT'99, Springer-Verlag, LNCS 1592, pp.295-310.

[17] Gnutella: http://www.gnutellanews.com.

[18] Haber, S. and Stornetta, W. "How to Time-Stamp a Digital Document." Journal of Cryptology, Vol. 3, No. 2, pp. 99-111, 1991.

[19] P. Huang, D. Estrin, and J. Heidemann. "Enabling Large-scale simulations: selective abstraction approach to the study of multicast protocols," In Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 241-248. Montreal, Canada, IEEE. July, 1998.

[20] Just, M. "Some Timestamping Protocol Failures." In Proceedings of the Internet Society Symposium on Network and Distributed Security (NDSS '98).

[21] Maniatis, P., Baker, M. "IdentiScape: Tackling the Personal Online Identity Crisis." Technical Report CSL-TR-00-804. Computer Systems Laboratory. Stanford University, June 2000.

[22] G. Kesidis and J. Walrand, "Quick Simulation of ATM Buffers with On-Off Multiples Markov Fluid Sources," ACM Transactions on Modeling and Computer Simulations, July 1993, pp. 269-276.

[23] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao "OceanStore: An Architecture for Global-Scale Persistent Storage." Proceeedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000).

[24] Lamport, L., Shostak. R., Pease, M. "The Byzantine Generals Problem." ACM Transaction on Programming Languages and Systems, Vol. 4, No. 3, July 1982, pp. 382-401.

[25] Malkhi, D. and Reiter, M. "Byzantine Quorum Systems." Journal of Distributed Computing, 11(4), 1998.

[26] Merkle, R. "Protocols for Public Key Cryptosystems". IEEE Symposium on Security and Privacy, 1980, pages 122-134.

[27] Naor, M., Pinkas, B., and Reingold, O. "Distributed Pseudo-Random Functions and KDCs." In J. Stern, editor, Advances in Cryptology: EUROCRYPT '99, volume 1592 of LNCS. Springer, 1999.

[28] T. Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing." In Advances in Cryptology - Crypto '91, pages 129-140. LNCS No. 576.

[29] T. Pedersen. "A threshold cryptosystem without a trusted party." In Advances in Cryptology - Eurocrypt 1991, pages 522-526. LNCS No. 547.

[30] Quisquater, J. J., Massias, H., Avila, J. S., Preneel, B. Van Rompay, B. "Specification and implementation of a timestamping system." Technical Report of the TIMESEC project. Université Catholique de Louvain, Belgium, 1999.

[31] Raman, B., Katz, R., and Joseph, A. "Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network." Workshop on Mobile Computing Systems and Applications (WMSCA'00), Dec 2000.

[32] Reiter, M. "Secure agreement protocols: Reliable and atomic group multicast in Rampart." in Proceedings of the 2nd ACM Conference on Computer and Communications Security, November 1994.

[33] RFC 2246. The TLS Protocol, Version 1.0. January 1999.

[34] A. Shamir, "How to Share a Secret." In Communications of the ACM, Volume 22, Number 11, November 1979.

[35] Surety, "Secure Time/Date Stamping in a Public Key Infrastructure." Available at http://www.surety.com.

[36] Verissimo, P., Casimiro, A. "The Timely Computing Base." Technical Report TR-99-2. Departamento de Informática, Universidade de Lisboa, Portugal. May 1999.

[37] M. Waldman, A. Rubin, and L. Cranor "Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System." 9th USENIX Security Symposium, August 2000.