Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks

Yair Amir, Claudiu Danilov, Danny Dolev, Jonathan Kirsch, John Lane, Cristina Nita-Rotaru, Josh Olsen, David Zage

***Abstract*— This paper presents the first hierarchical Byzantine tolerant replication architecture suitable to systems that span multiple wide area sites. The architecture confines the effects of any malicious replica to its local site, reduces message complexity of wide area communication, and allows read-only queries to be performed locally within a site for the price of additional hardware. A prototype implementation is evaluated over several network topologies and is compared with a flat Byzantine tolerant approach.**

I. Introduction

During the last few years, there has been considerable progress in the design of Byzantine tolerant replication systems. The current state of the art protocols perform very well on small-scale systems that are usually confined to local area networks. However, current solutions employ flat architectures that introduce several limitations: Message complexity limits their ability to scale, and strong connectivity requirements limit their availability on wide area networks that usually have lower bandwidth, higher latencies, and exhibit network partitions.

This paper presents Steward, the first hierarchical Byzantine tolerant replication architecture suitable for systems that span multiple wide area sites, each consisting of several server replicas. Steward assumes no trusted component in the entire system, other than a valid mechanism to pre-distribute private/public keys.

Steward uses a Byzantine tolerant protocol within each site and a lightweight, benign fault tolerant protocol among wide area sites. Each site, consisting of several potentially malicious replicas, is converted into a single logical trusted participant in the wide area fault-tolerant protocol. Servers within a site run a Byzantine agreement protocol to order operations locally, and agree upon the content of any message leaving the site for the global protocol.

Guaranteeing a consistent agreement within a site is not enough. The protocol needs to eliminate the ability of malicious replicas to misrepresent decisions that took place in their site. To that end, messages between servers at different sites carry a threshold signature attesting that enough servers at the originating site agreed with the content of the message. Using threshold signatures allows Steward to save the space and computation associated with sending and verifying multiple individual signatures. Moreover, it allows for a practical key management scheme where servers need to know only a single public key for each remote site and not the individual public keys of all remote servers.

The main benefits of our architecture are:

1) It reduces the message complexity on wide area exchanges from N^2 (N being the total number of replicas in the system) to S^2 (S being the number of wide area sites), considerably increasing the system's ability to scale.
2) It confines the effects of any malicious replica to its local site, enabling the use of a benign fault-tolerant algorithm over the wide area network. This improves the availability of the system over wide area networks that are prone to partitions, as only a majority of connected sites is needed to make progress, compared with at least $2f + 1$ servers (out of $3f + 1$) in flat Byzantine architectures.
3) It allows read-only queries to be performed locally within a site, enabling the system to continue serving read-only requests even in sites that are partitioned.
4) It enables a practical key management scheme where public keys of specific replicas need to be known only within their own site.

These benefits come with a price. If the requirement is to protect against **any** f Byzantine servers in the system, Steward requires $3f + 1$ servers in each site. However, in return, it is able to overcome up to f malicious servers in **each** site.

Steward further optimizes the above approach based on the observation that not all messages associated with the wide area fault-tolerant protocol require a complete Byzantine ordering agreement in the local site. A considerable amount of these wide area messages require a much lighter local site step, reducing the communication and computation cost on the critical path.

The paper demonstrates that the performance of Steward with $3f+1$ servers in *each site* is much better even compared with a flat Byzantine architecture with a smaller system of $3f+1$ *total* servers spread over the same wide area topology. The paper further demonstrates that Steward exhibits performance comparable (though somewhat lower) with common benign fault-tolerant protocols on wide area networks.

The Steward system is completely implemented and is currently undergoing a DARPA red-team experiment to assess

This work was supported in part by grant FA8750-04-2-0232 from the Defense Advanced Research Projects Agency.

Y. Amir, C. Danilov, J. Kirsch and J. Lane are with Johns Hopkins University, Baltimore, MD. tel: 410 516-5562, fax: 410 516-6134. {yairamir, claudiu, jak, johnlane}@cs.jhu.edu. D. Dolev is with The Hebrew University of Jerusalem, Jerusalem, Israel. tel: +972 2-658-4116, fax: +972 2-5-70-90-40. dolev@cs.huji.ac.il. C. Nita-Rotaru, J. Olsen and D. Zage are with Purdue University, West Lafayette, IN. tel: 765 496-6757, fax: 765 496-3181. {crisn, jolsen, zagedj}@cs.purdue.edu.

its practical survivability in the face of white-box attacks (where the red team has complete knowledge of system design, access to its source code, and control of up to f replicas in each site). We hope to be able to report on the insight gained from this activity in a final version of this paper.

The remainder of the paper is presented as follows. We provide a more detailed problem statement in Section II. We present our assumptions and the service model in Section III. We describe our protocol, Steward, and provide a sketch for a proof that it meets the specified safety and liveness properties, in Sections IV and V. We present experimental results demonstrating the improved scalability of Steward on wide area networks in Section VI. We discuss previous work in several related research areas in Section VII. We summarize our conclusions in Section VIII.

II. Background

Our work requires concepts from fault tolerance, Byzantine fault tolerance and threshold cryptography. To facilitate the presentation of our protocol, Steward, we first provide an overview of three representative works in these areas: Paxos, BFT and RSA Threshold Signatures.

Paxos: Paxos [1], [2] is a well-known fault-tolerant protocol that allows a set of distributed servers, exchanging messages via asynchronous communication, to totally order client requests in the benign-fault, crash-recovery model. One server, referred to as the *leader*, has the task of coordinating the protocol. If the leader crashes or becomes unreachable, a new leader is elected. Paxos requires at least $2f + 1$ servers to tolerate f faulty servers. Since servers are not Byzantine, only one reply needs to be delivered to the client.

In the common case, in which a single leader exists and can communicate with a majority of servers, Paxos uses two asynchronous communication rounds to globally order client updates. In the first round, *Proposal*, the leader assigns a sequence number to a client update, and proposes this assignment to the rest of the servers. In the second round, *Accept*, any server receiving the proposal assents to the assigned sequence number, or *accepts* the proposal, by sending an acknowledgment to the rest of the servers. When a server receives a majority of acknowledgments – indicating that a majority of servers have accepted the proposal – the server *orders* the corresponding update.

If the leader crashes or is partitioned away, the servers run a *leader election protocol* to replace the old leader, allowing progress to resume. The leader election protocol follows a similar two-round, proposal-accept pattern, where the value proposed will be a new leader. The protocol associates a unique view number with the reign of a leader (i.e. view) and defines a one-to-one mapping between the view number and the identifier of the server acting as the leader in this view. The system proceeds through a series of views, with a view change occurring each time a new leader is elected. Proposals are thus made in the context of a given view.

BFT: The BFT [3] protocol addresses the problem of replication in the Byzantine model where a number of the servers can be compromised and exhibit arbitrary behavior. Similar to Paxos, BFT uses an elected leader to coordinate the protocol, and proceeds through a series of views. BFT extends Paxos into the Byzantine environment by using an additional round of communication in the common case to ensure consistency both in and across views, and by constructing strong majorities in each round of the protocol. Specifically, BFT requires end-to-end acknowledgments from $2f+1$ out of $3f+1$ servers to mask the behavior of f Byzantine servers. A client must wait for $f+1$ identical responses to be guaranteed that at least one correct server assented to the returned value.

In the common case, BFT uses three communication rounds: *Pre-Prepare*, *Prepare* and *Commit*. In the first round, the leader assigns a sequence number to a client update and proposes this assignment to the rest of the servers by multicasting a *pre-prepare* message to all servers. In the second round, a server accepts the proposed assignment by sending an acknowledgment, *prepare*, to all servers. The first two communication rounds guarantee that correct servers agree on a total order of the updates proposed within the same view. When a server receives $2f + 1$ *prepare* messages with the same view number and sequence number as the *pre-prepare*, it begins the third round, *Commit*, by multicasting a *commit* message to all servers. A server *commits* the corresponding update when it receives $2f + 1$ matching *commit* messages. The third communication round, in combination with the view change protocol, ensures the total ordering of updates across views.

Threshold digital signatures: Threshold cryptography [4] distributes trust among a group of participants to protect information (e.g. threshold secret sharing [5]) or computation (e.g. threshold digital signatures [6]). Threshold schemes define a threshold parameter, k, such that any set of at least k (out of n) participants can work together to perform a desired task (such as computing a digital signature), while any subset of fewer than k participants is unable to do so. In this way, threshold cryptography offers a tunable degree of fault-tolerance: in the benign fault model, the system can function despite (n-k) faults, and in the Byzantine fault model, an adversary must corrupt k participants to break the system. In particular, corrupting fewer than k participants yields no useful information. There is a natural connection between Byzantine fault-tolerance and threshold cryptography, since both distribute trust among participants and make assumptions about the number of honest participants required in order to guarantee correctness.

A (k, n) threshold digital signature scheme allows a set of n servers to generate a digital signature as a single logical entity despite $f = (k - 1)$ Byzantine faults. In a (k, n) threshold digital signature scheme, a private key is divided into n partial shares, each owned by a server, such that any set of k servers can pool their shares to generate a valid threshold signature, while any set of fewer than k servers is unable to do so. To sign a message m, each server uses its share to generate a partial signature on m, and sends the partial signature to a *combiner* server. The combiner combines the partial signatures into a threshold signature on m. The threshold signature is verified in the standard way, using the public key corresponding to the divided private key. Shares can

be changed proactively [7], [8] without changing the public key, allowing for increased security and fault-tolerance, since an adversary must compromise k partial shares within a certain time window to break the system.

Since the participants can be malicious, it is important to be able to verify that the partial signature provided by any participant is valid – that is, it was generated with a share from the initial key split. This property, known as verifiable secret sharing [9], guarantees the robustness [10] of the threshold signature generation.

A representative example of practical threshold digital signature schemes is the RSA Shoup [6] scheme, which allows participants to generate threshold signatures based on the standard RSA[11] digital signature. The scheme defines a (k, n) RSA threshold signature scheme, and provides verifiable secret sharing. The computational overhead of verifying that the partial signatures were generated using correct shares is significant. The resulting threshold signature can be non-interactively verified using the same technique as the standard RSA signature.

III. System Model and Service Guarantees

In our model, servers are implemented as deterministic state machines. All correct servers begin in the same initial state, and transition between the states by applying updates as they are ordered. The next state is completely determined by the current state and the next update to be applied.

We assume a Byzantine fault model. Servers are classified as either *correct* or *faulty*. Correct servers do not crash. Faulty servers may behave in an arbitrary manner, and may: exhibit two-faced behavior, fail to send messages, collude with other faulty servers, etc.

Communication is asynchronous. Messages can be delayed, lost, or duplicated. Messages that do arrive are not corrupted.

Servers are organized into wide area *sites*. Each site has a unique identifier. Each server belongs to exactly one site. The network may partition into multiple disjoint *components*, each containing one or more sites. During a partition, servers from sites in different components are unable to communicate with each other. Components may subsequently re-merge. Each site S_i has at least $3 * (f_i) + 1$ servers, where f_i is the maximum number of servers that may be faulty within S_i. For simplicity, we assume in what follows that all sites may have f faulty servers.

Clients are distinguished by unique identifiers. Clients send updates to servers within their local site and receive responses from these servers. Each update is uniquely identified by a pair consisting of the identifier of the client that generated the update and a unique, monotonically increasing logical timestamp. Clients propose updates sequentially: a client may propose an update with timestamp $i + 1$ only after it receives a reply for an update with timestamp i.

We employ digital signatures, and we make use of a cryptographic hash function to compute message digests. Client updates are properly authenticated and protected against modifications. We assume that all adversaries, including faulty servers, are computationally bounded such that they cannot subvert these cryptographic mechanisms.

We also use a (k, n) threshold digital signature scheme. Each site has a public key, and each server receives a share with the corresponding proof. The share can be used to generate a partial signature, and the proof can be used to generate a verification proof that the partial signature was computed using a valid share. A valid threshold signature representing the site is computed by using k partial signatures. We assume that the threshold scheme guarantees that threshold signatures are unforgeable without knowing k or more secret shares.

Our protocol assigns global, monotonically increasing sequence numbers to updates to establish a global, total order. Below we define the safety and liveness properties of the Steward protocol. We say that:

- *a client proposes* an update when the client sends the update to a server in the local site.
- *a server initiates* an update when, upon receiving the update from a client, the server forwards the update for global ordering.
- *a server executes* an update with sequence i when it applies the update to its state machine. A server executes update i only after having executed all updates with a lower sequence in the global total order.
- *a site executes* an update when some correct server in the site executes the update.
- *two servers within a site are connected* if they can communicate with no communication failures.
- *two sites are connected* if every correct server of each site can communicate with every correct server of the other site with no communication failures.

Definition 3.1: S1 - Safety: If two correct servers execute the ith update, then these updates are identical.

Definition 3.2: S2 - Validity: Only an update that was proposed by a client (and subsequently initiated by a server) may be executed.

Definition 3.3: GL1 - Global Progress: If there exists a set of a majority of sites, each consisting of at least $2f + 1$ correct, connected servers, and a time after which all sites in the set are connected, then if a client connected to a site in the set proposes an update, some site in the set eventually executes the update.

IV. Protocol Description

Steward leverages a hierarchical architecture to scale Byzantine replication to the high-latency, low-bandwidth links characteristic of wide area networks. It employs more costly Byzantine fault-tolerant protocols within a site, confining Byzantine behavior to a site and allowing a more lightweight, fault-tolerant protocol to be run among sites. This results in fewer messages and communication rounds on the wide area compared to a flat Byzantine solution. The price is the need to have enough hardware within a site to overcome f malicious servers.

A site is made to behave as a single logical participant in the wide area fault-tolerant protocol through a combination of Byzantine agreement and threshold digital signatures. The servers within a site agree upon the content of any message leaving the site, and then construct a threshold signature on the

message to prevent a malicious server from misrepresenting the site. One server in each site, referred to as the *representative*, coordinates the internal agreement and threshold signing protocols within the site. The representative of one site, referred to as the *leading site*, coordinates the wide area agreement protocol. If the representative of a site acts maliciously, the servers of that site will elect a new representative. If the leading site is partitioned away, the servers in the other sites will elect a new leading site.

At a higher level, Steward uses a wide area Paxos-like algorithm to globally order updates. However, the entities participating in our protocol are not single trusted participants like in Paxos. Each site entity in our wide area protocol is composed of a set of potentially malicious servers. Steward employs several intra-site protocols as building blocks at each site, to emulate a correct Paxos participant in each of the wide area algorithm steps, based on need. For example, the leader participant in Paxos unilaterally assigns a unique sequence number to an update. Instead, Steward uses an intra-site protocol that employs a BFT-like mechanism to assign a global sequence number in agreement with the servers inside the *leading site*. The *leading site* will need to present to other sites a proof that the sequence indeed was assigned. Steward uses a different intra-site protocol to threshold-sign the Paxos proposal message demonstrating that $f+1$ *correct* servers in the *leading site* agreed to that global sequence number. The same threshold signature intra-site protocol is used to issue Paxos-like acknowledgments in non-leader sites.

In addition, Steward uses intra-site protocols that serve for Byzantine election of the new *representative* inside each site, as well as for proposing a new *leading site*.

The intra-site protocols used by Steward are as follows:

- THRESHOLD-SIGN: this protocol signs a message with a threshold signature composed of $2f+1$ shares, within a site. After executing this protocol, every correct process has a message that was signed with a threshold signature composed of $2f+1$ shares.
- ASSIGN-SEQUENCE: this protocol assigns a sequence number to an update received within a site, in the case when the representative is not suspected, and no internal view change takes place. It is invoked at the *leading site* to assign a unique sequence number to an update such that at least $f+1$ correct servers will agree on the sequence number.
- PROPOSE-LEADER-SITE: this protocol is used to generate an agreement inside a site regarding which wide area site should be the next *leading site* in the global ordering protocol.
- CONSTRUCT-COLLECTIVE-STATE: this protocol provides reconciliation during a view change and generates a message describing the current state of the site, as agreed by at least $f+1$ *correct* servers inside the site.

Below we provide a short description of the common case of operation of Steward, the view changes algorithms, the timers used by our protocols, and the inter-dependency between the global protocol and intra-site timeouts. A complete pseudocode of all protocols used by Steward can be found in [12].

A. The Common Case

During the common case, global progress is made and no *leading site* or site *representative* election occurs. The common case works as follows:

1) A client located at some site sends an update to a server in its local site. This server forwards the update to the local *representative*.
2) The local *representative* forwards the update to the *representative* of the *leading site*.
3) The *representative* of the *leading site* initiates a Byzantine agreement protocol within the site to assign a global sequence number to the update; this assignment is encapsulated in a *proposal* message. The site then generates a threshold digital signature on the constructed proposal, and the *representative* sends the signed proposal to all other sites for global ordering.
4) Upon receiving a signed proposal, the representative of each site initiates the process of generating a site acknowledgment (*accept*), and then sends the acknowledgment signed with a threshold signature to the representative of all other sites.
5) The representative of each site forwards the incoming accept messages to all local servers. A server globally orders the update when it receives signed accepts from a majority of sites. The server at the client's local site that originally received the update sends a reply back to the client.
6) If the client does not receive a reply to its update within a certain amount of time, it resends the update, this time broadcasting it to all servers at its site.

All site-originated messages that are sent as part of the fault-tolerant global protocol, require threshold digital signatures so that they may be trusted by other sites.

The THRESHOLD-SIGN intra-site protocol generates a (2f+1, 3f+1) threshold signature on a given message. As described in Section II, each server is assumed to have a partial share and a proof that the share was obtained from the initial secret (i.e. private key). Upon invoking the protocol on a message to be signed, the server generates a partial signature on this message. In addition, the server constructs a verification proof that can be used to confirm that the partial signature was indeed created using a valid share. Both the partial signature and the verification proof are sent to all servers within the site.

Upon receiving 2f+1 partial signatures on a message, a server combines the partial signatures into a threshold signature on that message. The constructed signature is then verified using the site's public key (RSA verification). If the signature verification fails, then one or more partial signatures used in the combination were invalid, in which case the verification proofs provided with the partial signatures are used to identify incorrect shares; the corresponding servers are classified as malicious. The invalid shares serve as proof of corruption and can be broadcast to all local servers. Further messages from the corrupted servers are ignored.

Once the *representative* of the *leading site* receives an update from a client (either local or forwarded by the *representative* of a different site), it assigns a sequence number

to this update by creating a proposal message that will then be sent to all other sites. The sequence number is assigned in agreement with other correct servers inside the site, masking the Byzantine behavior of malicious servers. The ASSIGN-SEQUENCE intra-site protocol is used for this purpose. The protocol consists of three rounds, the first two of which are similar to the corresponding rounds of the BFT protocol: the site *representative* proposes an assignment by sending a *pre-prepare* message to all servers within the site. Any server receiving the *pre-prepare* message sends to all servers a *prepare* message as acknowledgment that it accepts the representative's proposal. At the end of the second round, any server that has received 2f *prepare* messages, in addition to the *pre-prepare*, for the same view and sequence number, invokes the THRESHOLD-SIGN intra-site protocol to generate a threshold signature on the representative's proposal.

Upon completion of the ASSIGN-SEQUENCE protocol, the *representative* sends the proposal message for global ordering on the wide area to the representatives of all other sites.

Each site's representative receiving the proposal message forwards it to the other servers inside the site, and invokes the THRESHOLD-SIGN protocol to generate an acknowledgment (accept) of the proposal. The representative of the site then sends back the threshold signed *accept* message to the representatives of all other sites. Each *representative* will forward the *accept* message locally to all servers inside their site. A server within a site globally orders the update when it receives *accept* messages from a majority of sites.

B. View Changes

The above protocol describes the common-case operation of Steward. However, several types of failure may occur during system execution, such as the corruption of one or more site representatives, or the partitioning of the leader site. Such failures require delicate handling to preserve both safety and liveness.

If the representative of a site is faulty, the correct members of the site select a new representative by running a local view change protocol, after which progress can resume. The local view change algorithm preserves safety across views, even if consecutive representatives are malicious. Similarly, the *leading site* that coordinates the global ordering between the wide area sites can be perceived as faulty if no global progress is made. In this case, a global view change occurs. View changes are triggered by timeouts, as described in Section IV-C

Each server maintains a local view number and a global view number. The local view number maps to the identifier of the server's current site representative, while the global view number maps to the identifier of the wide area leader site. The local and global view change protocols update the server's corresponding view numbers.

We first introduce the CONSTRUCT-COLLECTIVE-STATE intra-site protocol, which is used as a building block in both the local and global view change protocols.

The CONSTRUCT-COLLECTIVE-STATE protocol generates a message describing the current state of a site, as agreed by at least $f+1$ *correct* servers within the site. The constructed message is referred to as a *union* message. The *representative* of a site invokes the protocol by sending a sequence number to all servers inside the site. Upon receiving the invocation message, all servers send to the *representative* a message containing updates they have ordered and/or acknowledged with a higher sequence number than the *representative*'s number. The *representative* computes a union on the contents of $2f+1$ of these messages, eliminating duplicates and using the latest update for a given sequence number if conflicts exist. The *representative* packs the contents of the union into a message and sends the message to all servers in the site. Upon receiving such a union message, each server updates its own state with missing updates as needed, generates a partial signature on the message, and sends the signed message to all servers within the site. A server then combines $2f+1$ such partial signatures into a single message that represents the updates that the site ordered or acknowledged above the original sequence number.

Local view change: The local view change protocol is similar to the one described in [3]. It elects a new site *representative* and guarantees that correct servers cannot be made to violate previous safety constraints.

The protocol is invoked when a server at some site observes that global progress has not been made within a timeout period, and is used at both the *leading site* and non-leader sites. A server that suspects the representative is faulty increases its local view number and sends to all local servers a *new_representative* message, which contains the proposed view number. Individual servers increase their proposed local view in a way similar to [3]. Upon receiving a set of $2f+1$ *new_representative* messages proposing the same view number (and, implicitly, a new representative), the new representative computes the sequence number of the highest update ordered, such that all updates with lower sequence numbers were ordered. We call this sequence number "ARU" (All Received Up-to). The new representative then invokes the CONSTRUCT-COLLECTIVE-STATE protocol based on its ARU. Finally, the new representative invokes the ASSIGN-SEQUENCE protocol to replay all pending updates that it learned from the signed union message.

Global view change: In the global view change protocol, wide area sites exchange messages to elect a new *leading site* if the current one is suspected to be faulty (partitioned away or with fewer than $2f+1$ correct servers). Each site runs an intra-site protocol, PROPOSE-LEADER-SITE, to generate a threshold-signed message containing the global view number that the site has agreed to propose.

The PROPOSE-LEADER-SITE protocol is invoked in a distributed fashion. Upon suspecting that the *leading site* is faulty, a server within a site increases its global view number and generates a partial signature on a message that proposes the new view. Upon receiving $2f+1$ partial signatures for the same global view number, the local *representative* combines the shares to construct the site's proposal. To ensure liveness, a server already suspects the *leading site* , and that receives $f+1$ partial signatures referring to global view numbers higher than its own, updates its global view number to the smallest value of the $f+1$ view numbers, and sends a corresponding

partial signature to the other servers in the site.

If enough servers in a site invoke the PROPOSE-LEADER-SITE protocol, the representative of that site will issue the resultant threshold-signed *new_leading_site* message that contains the identifier of that site and the proposed global view number. When the *representative*of the new *leading site* receives a majority of such messages proposing the same global view, it starts a local reconciliation protocol by invoking the CONSTRUCT-COLLECTIVE-STATE protocol on its own ARU. We call the highest sequence of an ordered update in the resulting union message, below which all lower sequence numbers are ordered, "Site ARU". The *representative* of the new *leading site* invokes the THRESHOLD-SIGN protocol on a message containing the Site ARU, and sends the resulting threshold-signed message to the representatives of all other sites. Based on the Site ARU received, the representatives of the non-leader sites invoke the CONSTRUCT-COLLECTIVE-STATE protocol and send the resultant union message back to the representative of the new *leading site*. A set of union messages from a majority of sites is used by servers in the *leading site* to constrain the messages they will generate in the new view so that safety is preserved.

C. Timeouts

Steward relies on timeouts to detect problems with the representatives in different sites or with the leading site. Our protocols do not assume synchronized clocks; however, we do assume that the rate of the clocks at different servers is reasonably close. We believe that this assumption is valid considering today's technology. Below we provide details about the timeouts in our protocol.

Local representative (T1): This timeout expires at a server of a non-leading site to replace the representative once no (global) progress takes place for that period of time. Once the timeout expires at $f+1$ servers, the local view change protocol takes place. T1 should be higher than 3 times the wide area network round-trip to allow a potential global view change protocol to complete without changing the local representative.

Leading site representative (T2): This timeout expires at a server at the leading site to replace the representative once no (global) progress takes place for that period of time. T2 should be large enough to allow the representative to communicate with a majority of the sites. Specifically, since not all sites may be lined up with correct representatives at the same time, T2 should be chosen such that each site can replace its representatives until a correct one will communicate with the *leading site*; the site needs to have a chance to replace $f+1$ representatives within the T2 time period. Thus, we need that $T2 >$(f+2)$*maxT1$, where $maxT1$ is an estimate of the largest T1 at any site. The $(f+2)$ covers the possibility that when the leader site elects a representative, the T1 timer is already running at other sites.

Leading site (T3): This timeout expires at a site to replace the leading site once no (global) progress takes place for that period of time. Since we choose T2 to ensure a single communication round with every site, and since the leading site needs at least 3 rounds to prove progress, in the worse case, the leading site must have a chance to elect 3 correct representatives to show progress, before being replaced. Thus, we need $T3 = (f+3)T2$.

Client timer (T0): This timeout expires at the client, triggering it to inquire the status of its last update by interacting with various servers at the site. T0 can have an arbitrary value.

Timeouts management: Servers send their timers estimation (T1, T2) on global view change messages. The site representative disseminates the $f+1st$ highest value (the value for which f higher or equal values exist) to prevent the faulty servers from injecting wrong estimates. Potentially, timers can be exchanged as part of local view change messages as well. The leading site representative chooses the maximum timer of all sites with which communicates to determine T2 (which in turn determines T3). Servers estimate the network round-trip according to various interactions they have had. They can reduce the value if communication seems to improve.

V. Correctness

In this section we provide a sketch of the proof that our algorithm provides safety and liveness; the complete proof can be found in [12].

Safety: The safety property requires that if two correct servers execute the ith update, then these updates are identical. Intuitively, this property is preserved by the ASSIGN-SEQUENCE protocol and by the reconciliation mechanism implemented by the CONSTRUCT-COLLECTIVE-STATE protocol invoked after a view change (either local or global). By requiring $2f+1$ servers to agree on any message within a site, the ASSIGN-SEQUENCE protocol guarantees that at least $f+1$ correct servers assented to the proposed sequence number. By using $2f+1$ shares to compute the threshold signature for any message that leaves a site, it ensures that any future set of $2f+1$ servers will contain at least one correct server that assented to the proposal. This guarantees that the site will not "forget" that it acknowledged the proposal should a view change occur, preventing a conflict to happen. Finally, by requiring a majority of site acknowledgments to globally order an update, the protocol guarantees that any intersection will contain a site from the previous majority.

Liveness: To guarantee liveness, Steward must ensure that servers move to a new local or global view when no progress takes place. By requiring a set of $2f+1$ servers to initiate a local view change, the local view change protocol guarantees that faulty replicas cannot create instability and prevent progress in the system by forcing frequent local view changes. The representative can cause view changes by malicious actions, but then a different representative will replace it. Since there are at most f faulty replicas in a site, after at most $f+1$ local view changes it is guaranteed that the site will have a correct representative.

In addition, by requiring a set of $2f+1$ servers to initiate a proposal for a global view change, the global view change algorithm prevents a malicious server from causing frequent global view changes.

A critical component that guarantees global liveness is the selection and management of timeout values, as described in

Section IV-C. The timeout values guarantee that the correct representative at the leading site has enough time to communicate with a correct representative at each site before it is replaced. Note that this requires that the correct representative at the leading site will need to stay valid long enough to allow up to $f + 1$ complete view changes to occur in non-leading sites.

VI. Performance Evaluation

To evaluate the performance of our hierarchical Byzantine replication architecture, we implemented a complete prototype of our protocol including all the necessary communication and cryptographic functionality. In this paper we focus only on the networking and cryptographic aspects of our protocols, and do not consider disk writes.

Testbed and Network Setup: We selected a network topology consisting of 5 wide area sites, assuming that there can be at most 5 Byzantine faults in each site, in order to quantify the performance of our system in a realistic scenario. This requires 16 replicated servers in each site.

Our architecture uses RSA threshold signatures [6] to represent an entire site within a single trusted message sent on the wide area network, thus trading computational power for wide area bandwidth and latency, in the number of wide area crossings. We believe this tradeoff is realistic considering the current technology trend: end-to-end wide area bandwidth is slow to improve, while latency reduction of wide area links is limited by the speed of light.

Our experimental testbed consists of a cluster with twenty 3.2GHz Intel Xeon computers, all of them having a 64-bit architecture. On these computers, a 1024 bit RSA signature can be computed in 1.3 msec and can be verified in 0.07 msec. The leader site was deployed on 16 of the machines, and the other 4 sites were emulated by one computer each[1]. The emulating computers were seen from the other sites as if they were the representatives of complete 16 server sites, for a system consisting of a total of 80 nodes spread over 5 sites. Upon receiving a packet at a non-leader site, the emulating computers were busy-waiting for the amount of time it took a 16 server site to handle that packet and reply to it, including both in-site communication and the necessary computation. The busy-waiting times for each type of packet were determined in advance by benchmarking individual protocols on a fully deployed, 16 server site. We used the Spines [13] messaging system to emulate latency and throughput constraints on the wide area links.

We compared the performance results of the above system with those obtained by BFT [3] on the same network setup with five sites, run on the same cluster, only that instead of using 16 servers in each site, for BFT we used a **total** of 16 servers across the entire network. This allows for up to 5 Byzantine failures in the entire network for BFT, instead of up to 5 Byzantine failures in each site for Steward; however, since BFT is a flat solution where there is no correlation between faults and the sites where they can occur, we believe this comparison is fair and conservative. We distributed the BFT servers such that four sites contain 3 servers each, and one site contains 4 servers. All the write updates and read-only queries in our experiments carried a payload of 200 bytes, representing a common SQL statement.

Bandwidth Limitation: We first investigate the benefits of the hierarchical architecture in a symmetric configuration with 5 sites, where all sites are connected to each other with 50 milliseconds latency links. A 50 millisecond delay emulates the wide area crossing of the continental US.

In the first experiment, clients inject write updates. Figure 1 shows the update throughput when increasing the number of clients, limiting the capacity of wide area links between the sites to 10, 5 and 2.5Mbps, both for Steward and BFT. The graph shows that up to 2.5Mbps, Steward is not limited by bandwidth. The system is able to process a total of about 84 updates/sec, being limited only by CPU, used for computing threshold signatures at the sites.

As we increase the number of clients, the BFT throughput increases at a lower slope than Steward, mainly due to the one additional wide area crossing for each update. At 10 Mbps, BFT achieves about 58 updates/sec, being limited by the available bandwidth. Similarly, at 5 Mbps it can sustain a maximum of 26 updates/sec, and at 2.5 Mbps a maximum of about 6 updates/sec. We also notice a reduction in the throughput of BFT as the number of clients increases. We believe this is due to a cascading increase of message loss, generated by the lack of a wide area flow control in the original implementation of BFT. Such a flow control was not needed as BFT was designed to work in LANs. For the same reason, we were not able to run BFT with more than 24 clients at 5 Mbps, and 15 clients at 2.5Mbps. We believe that adding a client queuing mechanism would stabilize the performance of BFT to its maximum achieved throughput, regardless of the number of clients.

The average update latency, as depicted in Figure 2, shows Steward achieving almost constant latency. The latency slightly increases with the addition of clients, reaching 190 ms when 15 clients send updates into the system. At this point, as client updates start to be queued, their latency increases linearly with the number of clients in the system. BFT exhibits a similar behavior at 10 Mbps, only that its update latency is affected by the additional number of messages sent and the additional wide area crossing, such that for 15 clients the average update latency is 336 ms. As the bandwidth decreases, the update latency increases heavily, reaching up to 600 ms at 5 Mbps and 5 seconds at 2.5 Mbps, for 15 clients.

Adding Read-only Queries: One of the benefits of our hierarchical architecture is that read-only queries can be answered locally, at each site. To demonstrate these benefits we conducted an experiment where 10 clients send mixes of read-only queries and write updates, chosen randomly at each client, with different ratios. We compared the performance of Steward and BFT when both systems are not limited by bandwidth constraints. We used links of 50 ms, 10 Mbps between the

[1]Our implementation was tested on a complete deployment where each site is composed on multiple computers using the complete set of protocols and is currently undergoing a 5-sites DARPA red-team exercise. In order to evaluate Steward's scalability on large networks supporting many faults at each site, we used emulating computers for non-leader sites to limit the deployment to our cluster of 20 machines.

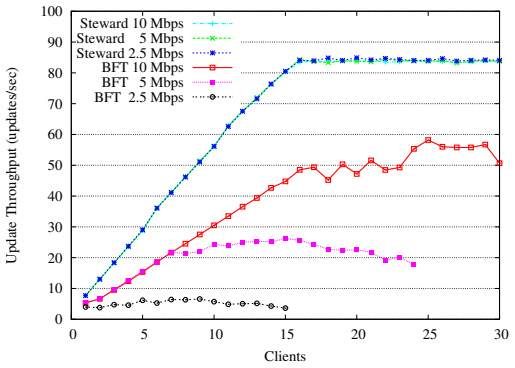

Fig. 1. Write Update Throughput

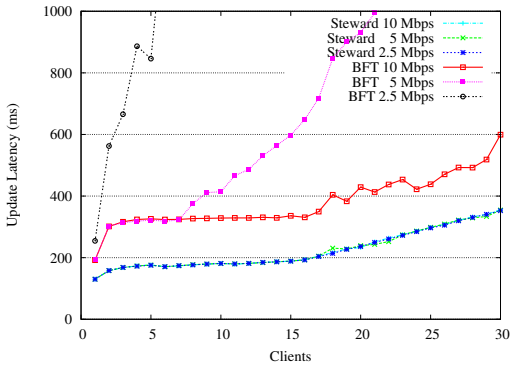

Fig. 2. Write Update Latency

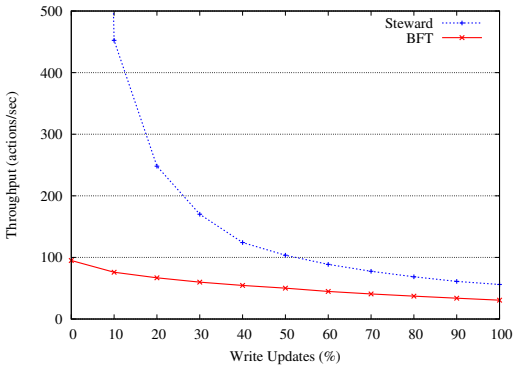

Fig. 3. Update Mix Throughput - 10 Clients

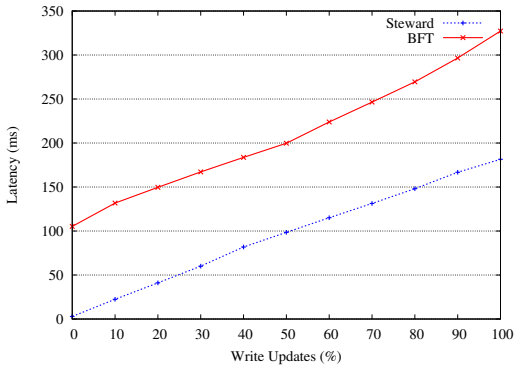

Fig. 4. Update Mix Latency - 10 Clients

sites. Figures 3 and 4 show the average throughput and latency, respectively, of different mixes of queries and updates sent using Steward and BFT. When clients send only read queries, Steward achieves about 2.9 ms per query, with a throughput of over 3,400 queries per second. This is because all the queries are answered locally, their latency being dominated by two RSA signature operations: one at the originating client, and one at the servers answering the query.

For BFT, the latency of read-only queries is about 105 ms, and the total throughput achieved is 95 queries per second. This is expected, as read-only queries in BFT need to be answered by at least $f + 1$ servers, some of which being located across wide area links. BFT could have achieved queries locally in a site if we deployed it such that there are at least $2f + 1$ servers in each site (in order to guarantee liveness it needs $f + 1$ *correct* servers to answer queries in each site). Such a deployment, for $f = 5$ faults and 5 sites, would need at least 55 servers total, which will dramatically increase communication for updates, and further reduce BFT's performance.

As the percentage of write updates in the query mix increases, the average latency for both Steward and BFT increases linearly, with Steward latency being about 100 ms lower than BFT at all times. This is a substantial improvement considering the absolute value of the update latency, the ratio between the latency achieved by the two systems ranging from a factor of two, when only write updates are served, to a factor of 30, when only read queries are served. The throughput drops with the increase of update latency, such that at 100% write updates there is only about a factor of two between the throughput achieved by Steward and BFT.

Wide Area Scalability: To demonstrate the scalability of the hierarchical architecture we conducted an experiment that emulated a wide area network that covers several continents. We selected five sites on the Planetlab network [14], measured the latency and available bandwidth characteristics between every pair of sites, and emulated the network topology on our cluster in order to run Steward and BFT. We ran the experiment on our cluster, and not directly on Planetlab because Planetlab machines are not of 64-bit architecture. Moreover, Planetlab computers provide a shared environment where multiple researchers run experiments at the same time, bringing the load on almost all the machines to more than 100% at all times. Such an environment lacks the computational power required for the two systems tested, and would artificially influence our experimental results.

The five sites we emulated in our tests are located in the US (University of Washington), Brazil (Rio Grande do Sul), Sweden (Swedish Institute of Computer Science), Korea (KAIST) and Australia (Monash University). The network latency varied between 59 ms (US - Korea) and 289 ms (Brazil - Korea). Available bandwidth varied between 405 Kbps(Brazil - Korea) and 1.3Mbps (US - Australia).

Figure 5 shows the average write update throughput as we increased the number of clients in the system, while Figure 6 shows the average update latency. As seen in Figures 5 and 6,

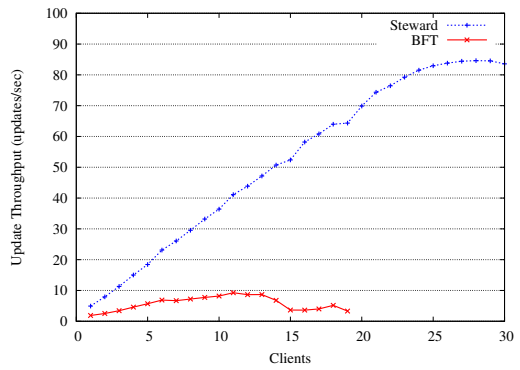

Fig. 5. Wide Area Network Emulation - Write Update Throughput

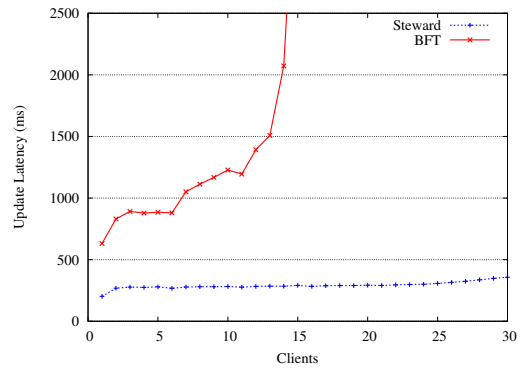

Fig. 6. Wide Area Network Emulation - Write Update Latency

Steward is able to achieve its maximum limit of about 84 updates/second when 27 clients inject updates into the system. The latency increases from about 200 ms for 1 client, to about 360 ms for 30 clients.

BFT is limited by the available bandwidth to a maximum of about 9 updates/sec, while the update latency starts at 631 ms for one client, and jumps to the order of seconds when more than 6 clients are introduced.

Comparison with Non-Byzantine Protocols: Since Steward deploys a lightweight fault-tolerant protocol between the wide area sites, we expect it to achieve performance comparable to existing non-Byzantine fault-tolerant protocols commonly used in database replication systems, but with Byzantine guarantees (while paying more hardware).

In the following experiment we compare the performance of our hierarchical Byzantine architecture with that of two-phase commit protocols. In [15] we evaluated the performance of two-phase commit protocols [16] using a wide area network setup across the US, called CAIRN [17]. We emulated the topology of the CAIRN network using the Spines messaging system, and ran Steward and BFT on top of it. The main characteristic of the CAIRN topology is that East and West Coast sites were connected through a single link of 38ms and 1.86Mbps.

Figures 7 and 8 show the average throughput and latency of write updates, respectively, of Steward and BFT on the CAIRN network topology. Steward was able to achieve about 51 updates/sec in our tests, being limited mainly by the bandwidth of the link between the East and West Coasts in CAIRN. In comparison, an upper bound of two-phase commit protocols presented in [15] was able to achieve 76 updates/sec. As our architecture uses a non-Byzantine fault-tolerant protocol between the sites, it was expected to achieve comparable results with two phase commit protocols. We believe that the difference in performance is caused by the presence of additional digital signatures in the message headers of Steward, adding 128 bytes to the 200 byte payload of each update.

The high bandwidth requirement of BFT causes it to achieve a very low throughput and high latency on the CAIRN network. The maximum throughput achieved by BFT was 2.7 updates/sec and the update latency was over a second, except when a single client injected updates in the entire system.

Summary: The performance results we presented show that our hierarchical Byzantine architecture achieves performance comparable (though somewhat lower) to non-Byzantine protocols when run on wide area networks with multiple sites, and is able to scale to networks that span across several continents. In addition, our experiments show that the ability of our architecture to answer queries locally inside a site gives substantial performance improvements beyond the qualitative benefit of allowing read-only queries in the presence of partitions. In contrast, flat Byzantine protocols, while performing very well on local area networks, do not scale well to multiple sites across a wide area network. They have high bandwidth requirements, and use additional rounds of communication that increase individual update latency and reduce their total achievable throughput.

VII. Related Work

Agreement and Consensus: At the core of many replication protocols is a more general problem, known as the agreement or consensus problem. There are several models that researchers considered when solving consensus, the strongest one being the Byzantine model in which a participant can behave in an arbitrary manner. Other than the behavior of a participant (malicious or not), other relevant considerations are whether communication is asynchronous or synchronous, whether authentication is available or not, and whether the participants communicate over a flat network or not. A good overview of significant results is presented in [18]. Optimal results state that under the assumption that communication is not authenticated and nodes are directly connected, in order to tolerate f Byzantine failures, $3f+1$ participants and $f+1$ communication rounds are required. If authentication is available, then $f+1$ rounds are still required, but the number of participants just has to be greater than $f+1$ [19]. An important factor that must be taken into consideration is whether participants are directly connected or not. In [20], Dolev shows that in an arbitrary connected network, if f Byzantine faults must be tolerated and the network is $f+1$ ($2f+1$ if no signature exists) connected, then agreement can be achieved in $2f+1$ rounds.

Byzantine Group Communication: Related with our work are group communication systems resilient to Byzantine failures. The most well-known such systems are Rampart [21]

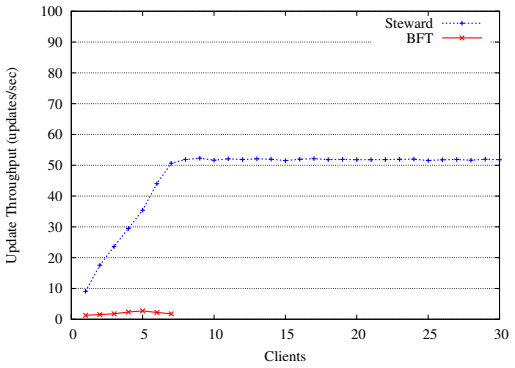

Fig. 7. CAIRN Network Emulation - Write Update Throughput

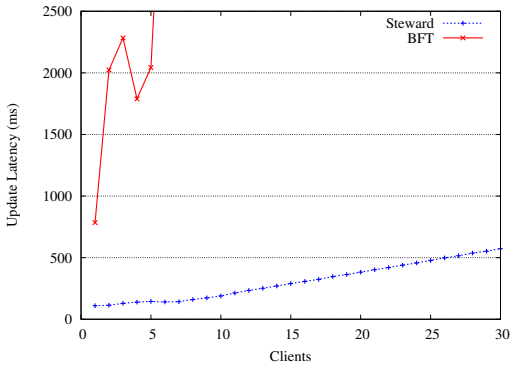

Fig. 8. CAIRN Network Emulation - Write Update Latency

and SecureRing [22]. Although these systems are extremely robust, they have a severe performance cost and require a large number of un-attacked nodes to maintain their guarantees. Both systems rely on failure detectors to determine which replicas are faulty. An attacker can exploit this to slow correct replicas or the communication between them until enough are excluded from the group.

Another intrusion-tolerant group communication system is ITUA [23], [24], [25], [26]. The ITUA system, developed by BBN and UIUC, focuses on providing intrusion tolerant group services. The approach taken considers all participants as equal and is able to tolerate up to less than a third of malicious participants.

Replication with Benign Faults: The two-phase commit (2PC) protocol [16] provides serializability in a distributed database system when transactions may span several sites. It is commonly used to synchronize transactions in a replicated database. Three-phase commit [Ske82] overcomes some of the availability problems of 2PC, paying the price of an additional communication round, and therefore, additional latency. Paxos [1] is a very robust algorithm for benign fault-tolerant replication. Paxos uses two rounds of messages in the common case to assign a total order to updates and requires $2f + 1$ replicas in order to tolerate f faults.

Quorum Systems with Byzantine Fault-Tolerance: Quorum systems obtain Byzantine fault-tolerance by applying quorum replication methods. Examples of such systems include Phalanx [27], [28] and its successor Fleet [29], [30]. Fleet provides a distributed repository for Java objects. It relies on an object replication mechanism that tolerates Byzantine failures of servers, while supporting benign clients. Although the approach is relatively scalable with the number of replica servers, it suffers from the drawbacks of flat non-hierarchical Byzantine replication solutions.

Replication with Byzantine Fault-Tolerance: The first practical work to solve replication while withstanding Byzantine failures is the work of Castro and Liskov [3]. Their algorithm requires $3f + 1$ replicas in order to tolerate f faults. In addition, the client has to wait for $f + 1$ identical answers (which, for liveness guarantees may require waiting for up to $2f + 1$ answers) in order to make sure that a correct answer is received. The algorithm obtains very good performance on local area networks. Yin et al. [31] propose an improvement for the Castro and Liskov approach by separating the agreement component that orders requests from the execution component that processes requests. The separation allows utilization of the same agreement component for many different replication tasks. It also reduce the number of processing storage replicas to $2f + 1$. Martin and Alvisi [32] recently introduced an algorithm that is able to achieve Byzantine consensus in only two rounds, while using $5f + 1$ servers in order to overcome f faults. This approach trades lower availability ($4f + 1$ out of $5f + 1$ connected replicas are required, instead of $2f + 1$ out of $3f + 1$ in BFT), for increased performance. The solution seems very appealing for local area networks that provide high connectivity between the replicas. We considered using it within the sites in our architecture to reduce the number of intra-site communication rounds. However, as we make use of threshold signatures inside a site, the overhead of combining larger signatures of $4f + 1$ shares would greatly overcome the benefits of using one less communication round within the site.

Alternate architectures: An alternate hierarchical approach to scale Byzantine replication to wide area networks can be based on having a few trusted nodes that are assumed to be working under a weaker adversary model. For example, these trusted nodes may exhibit crashes and recoveries but not penetrations. A Byzantine replication algorithm in such an environment can use this knowledge in order to optimize the performance and bring it closer to the performance of a fault-tolerant, non-Byzantine solution.

Such a hybrid approach was proposed in [33], [34] by Verissimo et al, where trusted nodes were also assumed to perform synchronously, providing strong global timing guarantees. The hybrid failure model of [33] inspired the Survivable Spread [35] work, where a few trusted nodes (at least one per site) are assumed impenetrable, but are not synchronous, may crash and recover, and may experience network partitions and merges. These trusted nodes were implemented by Boeing Secure Network Server (SNS) boxes, which are limited computers designed specifically not to be penetrable.

In our opinion, both the hybrid approach proposed in [34], and the approach proposed in this paper seem viable to practically scale Byzantine replication to wide area networks. The hybrid approach makes stronger assumptions while our approach pays more hardware and computational costs. Fur-

ther developing both approaches and contrasting them can be a fertile ground for future research.

VIII. Conclusions

This paper presented a hierarchical architecture that enables efficient scaling of Byzantine replication to systems that span multiple wide area sites, each consisting of several potentially malicious replicas. The architecture reduces the message complexity on wide area updates, increasing the system's ability to scale. By confining the effect of any malicious replica to its local site, the architecture enables the use of a benign fault-tolerant algorithm over the wide area network, increasing system availability. Further increase in availability and performance is achieved by the ability to process read-only queries within a site.

We implemented Steward, a fully functional prototype that realizes our architecture, and evaluated its performance over several network topologies. The experimental results show considerable improvement over flat Byzantine replication algorithms, bringing the performance of Byzantine replication closer to existing benign fault-tolerant replication techniques over wide area networks.

References

[1] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, pp. 133–169, May 1998.

[2] Lamport, "Paxos made simple," *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, vol. 32, 2001.

[3] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002.

[4] Y. G. Desmedt and Y. Frankel, "Threshold cryptosystems," in *CRYPTO '89: Proceedings on Advances in cryptology*, (New York, NY, USA), pp. 307–315, Springer-Verlag New York, Inc., 1989.

[5] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[6] V. Shoup, "Practical threshold signatures," *Lecture Notes in Computer Science*, vol. 1807, pp. 207–223, 2000.

[7] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, (London, UK), pp. 339–352, Springer-Verlag, 1995.

[8] L. Zhou, F. Schneider, and R. van Renesse, "APSS: Proactive Secret Sharing in Asynchronous Systems."

[9] P. Feldman, "A Practical Scheme for Non-Interactive Verifiable Secret Sharing," in *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, (Los Angeles, CA, USA), pp. 427–437, IEEE Computer Society, IEEE, October 1987.

[10] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold dss signatures," *Inf. Comput.*, vol. 164, no. 1, pp. 54–84, 2001.

[11] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, Feb. 1978.

[12] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage, "Steward: Scaling byzantine fault-tolerant systems to wide area networks," Tech. Rep. CNDS-2005-3, Johns Hopkins University and Purdue University, December 2005.

[13] "The spines project, http://www.spines.org/."

[14] "Planetlab." http://www.planet-lab.org/.

[15] Y. Amir, C. Danilov, M. Miskin-Amir, J. Stanton, and C. Tutu, "On the performance of consistent wide-area database replication," Tech. Rep. CNDS-2003-3, December 2003.

[16] K. Eswaran, J. Gray, R. Lorie, and I. Taiger, "The notions of consistency and predicate locks in a database system," *Communication of the ACM*, vol. 19, no. 11, pp. 624–633, 1976.

[17] "The CAIRN Network." http://www.isi.edu/div7/CAIRN/.

[18] M. J. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," in *Fundamentals of Computation Theory*, pp. 127–140, 1983.

[19] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM Journal of Computing*, vol. 12, no. 4, pp. 656–666, 1983.

[20] D. Dolev, "The byzantine generals strike again," *Journal of Algorithms*, vol. 3, no. 1, pp. 14–30, 1982.

[21] M. K. Reiter, "The Rampart Toolkit for building high-integrity services," in *Selected Papers from the International Workshop on Theory and Practice in Distributed Systems*, (London, UK), pp. 99–110, Springer-Verlag, 1995.

[22] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "The SecureRing protocols for securing group communication," in *Proceedings of the IEEE 31st Hawaii International Conference on System Sciences*, vol. 3, (Kona, Hawaii), pp. 317–326, January 1998.

[23] M. Cukier, T. Courtney, J. Lyons, H. V. Ramasamy, W. H. Sanders, M. Seri, M. Atighetchi, P. Rubel, C. Jones, F. Webber, P. Pal, R. Watro, and J. Gossett, "Providing intrusion tolerance with itua," in *Supplement of the 2002 International Conference on Dependable Systems and Networks*, June 2002.

[24] H. V. Ramasamy, P. Pandey, J. Lyons, M. Cukier, and W. H. Sanders, "Quantifying the cost of providing intrusion tolerance in group communication systems," in *The 2002 International Conference on Dependable Systems and Networks (DSN-2002)*, June 2002.

[25] P. Pandey, "Reliable delivery and ordering mechanisms for an intrusion-tolerant group communication system." Masters Thesis, University of Illinois at Urbana-Champaign, 2001.

[26] H. V. Ramasamy, "A group membership protocol for an intrusion-tolerant group communication system." Masters Thesis, University of Illinois at Urbana-Champaign. 2002.

[27] D. Malkhi and M. K. Reiter, "Secure and scalable replication in phalanx," in *SRDS '98: Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems*, (Washington, DC, USA), p. 51, IEEE Computer Society, 1998.

[28] D. Malkhi and M. Reiter, "Byzantine quorum systems," *Journal of Distributed Computing*, vol. 11, no. 4, pp. 203–213, 1998.

[29] D. Malkhi and M. Reiter, "An architecture for survivable coordination in large distributed systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 2, pp. 187–202, 2000.

[30] D. Malkhi, M. Reiter, D. Tulone, and E. Ziskind, "Persistent objects in the fleet system," in *The 2nd DARPA Information Survivability Conference and Exposition (DISCEX II). (2001)*, June 2001.

[31] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement from execution for byzantine fault-tolerant services," in *SOSP*, 2003.

[32] J.-P. Martin and L. Alvisi, "Fast byzantine consensus.," in *DSN*, pp. 402–411, 2005.

[33] M. Correia, L. C. Lung, N. F. Neves, and P. Veríssimo, "Efficient byzantine-resilient reliable multicast on a hybrid failure model," in *Proc. of the 21st Symposium on Reliable Distributed Systems*, (Suita, Japan), Oct. 2002.

[34] P. Verissimo, "Uncertainty and predictability: Can they be reconciled," in *Future Directions in Distributed Computing*, no. 2584 in LNCS, Springer-Verlag, 2003.

[35] "Survivable spread: Algorithms and assurance argument," Tech. Rep. Technical Information Report Number D950-10757-1, The Boeing Company, July 2003.