Scaling Byzantine Fault-Tolerant Replication to Wide Area Networks

Yair Amir[1], Claudiu Danilov[1], Danny Dolev[2], Jonathan Kirsch[1], John Lane[1],
Cristina Nita-Rotaru[3], Josh Olsen[3], David Zage[3]
[1] Johns Hopkins University, Baltimore, MD. {yairamir, claudiu, jak, johnlane}@cs.jhu.edu
[2] Hebrew University of Jerusalem, Jerusalem, Israel. dolev@cs.huji.ac.il
[3] Purdue University, West Lafayette, IN. {crisn, jolsen, zagedj}@cs.purdue.edu

Abstract

This paper presents the first hierarchical Byzantine fault-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 fault-tolerant approach.

1 Introduction

During the last few years, there has been considerable progress in the design of Byzantine fault-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 WANs that usually have lower bandwidth, higher latencies, and exhibit network partitions.

This paper presents Steward, the first hierarchical Byzantine fault-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 fault-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 they 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 $O(N^2)$ (N being the total number of replicas in the system) to $O(S^2)$ (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 WAN. This improves the availability of the system over WANs 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 away.
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's efficacy depends on using servers within a site which are unlikely to suffer the same vulnerabilities. Multi-version programming [1], where independently coded software implementations are run on each server, can yield the desired diversity. Newer techniques [2] can automatically and inexpensively generate variation.

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.

We implemented the Steward system and a DARPA red-team experiment has confirmed 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 f replicas in each site). According to the rules of engagement, where a red-team attack succeeded only if it stopped progress or caused consistency errors, no attacks succeeded. We have included a detailed description of the red-team experiment in [3].

The remainder of the paper is presented as follows. We provide background in Section 2. We present our assumptions and the service model in Section 3. We describe our protocol, Steward, in Section 4. We present experimental results demonstrating the improved scalability of Steward on WANs in Section 5. We discuss previous work in several related research areas in Section 6. We summarize our conclusions in Section 7.

2 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 [4, 5] 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*, coordinates the protocol. If the leader crashes or becomes unreachable, a view change occurs, allowing progress to resume in the new view under the reign of a new leader. Paxos requires at least $2f + 1$ servers to tolerate f faulty servers. Since servers are not Byzantine, a single 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, the leader assigns a sequence number to a client update and proposes this assignment to the rest of the servers. In the second round, any server receiving the proposal *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.

BFT: The BFT [6] protocol addresses the problem of replication in the Byzantine model where a number of the servers can 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 communication round 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 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. 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 broadcasting a *pre-prepare* message. In the second round, a server accepts the proposed assignment by broadcasting an acknowledgment, *prepare*. 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 by broadcasting a *commit* message. A server *commits* the corresponding update when it receives $2f + 1$ matching *commit* messages.

Threshold digital signatures: Threshold cryptography [7] distributes trust among a group of participants to protect information (e.g. threshold secret sharing [8]) or computation (e.g. threshold digital signatures [9]).

A (k, n) threshold digital signature scheme allows a set of servers to generate a digital signature as a single logical entity despite (k - 1) Byzantine faults. It divides a private key into n shares, each owned by a server, such that any set of k servers can pool their shares to generate a valid threshold signature on a message, m, while any set of fewer than k servers is unable to do so. Each server uses its key share to generate a partial signature on m and sends the partial signature to a *combiner* server, which combines the partial signatures into a threshold signature

on m. The threshold signature is verified using the public key corresponding to the divided private key.

A representative example of practical threshold digital signature schemes is the RSA Shoup [9] scheme, which allows participants to generate threshold signatures based on the standard RSA[10] digital signature. It provides verifiable secret sharing [11] (i.e. the ability to confirm that a signature share was generated using a valid private key share), which is critical in achieving robust signature generation [12] in Byzantine environments.

3 System Model and Service Guarantees

Servers are implemented as deterministic state machines [13]. All correct servers begin in the same initial state and transition between 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 either *correct* or *faulty*. Correct servers do not crash. Faulty servers may behave arbitrarily. 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 having a unique identifier. Each server belongs to 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 at most 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 ($2f + 1$, $3f + 1$) threshold digital signature scheme. Each site has a public key, and each server receives a share with the corresponding proof that can be used to demonstrate the validity of the server's partial signatures. We assume that threshold signatures are unforgeable without knowing $2f + 1$ 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 correct server in the local site, and the correct server receives it.
- *a server executes* an update with sequence number 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 number in the global total order.
- *two servers are connected* if any message that is sent between them will arrive in a bounded time. This bound is not known beforehand, and it may change over time.
- *two sites are connected* if every correct server of one site is connected to every correct server of the other site.

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 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 correct server in some site in the set eventually executes the update.*

4 Protocol Description

Steward leverages a hierarchical architecture to scale Byzantine replication to the high-latency, low-bandwidth links characteristic of WANs. It uses relatively costly Byzantine fault-tolerant protocols and threshold signatures to confine malicious behavior to local sites. Each site then acts as a single logical participant in a Paxos-like, benign fault-tolerant protocol run on the wide area. The use of this lightweight protocol reduces the number of messages and communication rounds on the wide area compared to a flat Byzantine solution.

Since each entity participating in our protocol is a site consisting of a set of potentially malicious servers (instead of a single trusted participant as Paxos assumes), Steward employs several intra-site protocols to emulate the behavior of a correct Paxos participant. For example, while the leader in Paxos can unilaterally assign a

sequence number to an update, Steward instead uses a BFT-like agreement algorithm, ASSIGN-SEQUENCE, at the *leading site* to assign a sequence number. Steward then uses a second intra-site protocol, THRESHOLD-SIGN, to sign the resulting Paxos proposal message.

One server in each site, the *representative*, coordinates the local agreement and threshold signing protocols. The representative of 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 one.

Below we provide a description of Steward's common case operation, the view changes algorithms, and the timers. Complete pseudocode and a proof of correctness can be found in [3].

4.1 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 sends an update to a server in its local site. This server forwards the update to the local representative, which forwards the update to the representative of the leading site. If the client does not receive a reply in time, it broadcasts the update.
2. The representative of the leading site initiates the ASSIGN-SEQUENCE protocol to assign a global sequence number to the update; this assignment is encapsulated in a *proposal* message. The site then generates a signature on the constructed proposal using THRESHOLD-SIGN, and the representative sends the signed proposal to the representatives of all other sites for global ordering.
3. When a representative receives a signed proposal, it forwards this proposal to the servers in its site. Upon receiving a proposal, a server constructs a site acknowledgment (*accept*) and invokes THRESHOLD-SIGN on this message. The representative combines the partial signatures and then sends the resulting threshold-signed *accept* to the representatives of the other sites.
4. The representative of a site forwards the incoming *accept* messages to the local servers. A server globally orders the update when it receives $\lfloor N/2 \rfloor$ distinct *accept* messages (where N is the number of sites) and the corresponding *proposal*. The server at the client's local site that originally received the update sends a reply back to the client.

We now highlight the details of the THRESHOLD-SIGN and ASSIGN-SEQUENCE protocols.

Threshold-Sign: The THRESHOLD-SIGN intra-site protocol generates a $(2f+1, 3f+1)$ threshold signature on a given message [1]. Upon invoking the protocol, a server generates a partial signature on the message to be signed and a verification proof that other servers can use to confirm that the partial signature was created using a valid share. Both the partial signature and the verification proof are broadcast 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, which is then verified using the site's public key. If the signature verification fails, 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. Further messages from the corrupted servers are ignored.

Assign-Sequence: The ASSIGN-SEQUENCE intra-site protocol consists of three rounds, the first two of which are similar to the corresponding rounds of BFT. At the end of the second round, any server that has received 2f *prepares* and the *pre-prepare*, for the same view and sequence number, invokes THRESHOLD-SIGN to generate a threshold signature on the representative's *proposal*.

4.2 View Changes

Several types of failure may occur during system execution, such as the corruption of a site representative or the partitioning of the leader site. Such failures require delicate handling to preserve safety and liveness.

We use two relatively independent mechanisms to handle failures. First, if a protocol coordinator [2] is faulty, the correct participants elect a new coordinator using a protocol similar to the one described in [6]. Second, we use reconciliation to constrain protocol participants such that safety is preserved across views. Note that there is a local and global component to both mechanisms, each serving a similar function at its level of the hierarchy.

Below we provide relevant details of leader election, intra-site reconciliation (via the CONSTRUCT-COLLECTIVE-STATE intra-site protocol), local view change, and global view change.

Leader Election: We refer the reader to [6] for a detailed description of the intra-site representative election protocol. To elect a leading site, each site first runs an intra-site protocol to agree upon the site it will propose; these votes are exchanged among the sites in a manner similar to the intra-site representative election protocol.

---

[1] We could use an $(f+1, 3f+1)$ threshold signature at the cost of an additional protocol round.

[2] Within a site, the protocol coordinator is the local representative, and, in the high-level protocol, the coordinator is the leading site.

Construct Collective State: The CONSTRUCT-COLLECTIVE-STATE protocol is used as a building block in both view changes. It serves two primary functions: guaranteeing sufficient intra-site reconciliation to safely make progress after a local view change and generating a message reflecting the site's level of knowledge, which is used during a global view change.

A site representative invokes the protocol by sending a sequence number, seq, to all servers within the site. A server responds with a message containing the updates it has ordered and/or acknowledged with a higher sequence number than seq. The representative computes the union of $2f+1$ responses, eliminating duplicates and using the update from the latest view if multiple updates have the same sequence number, and broadcasts it within the site. When a server receives a *union* message, it collects missing messages from the *union* and invokes THRESHOLD-SIGN on the *union*.

Local View Change: The local view change protocol is triggered in the leading site after a local representative election. The new representative invokes CONSTRUCT-COLLECTIVE-STATE with the sequence number up to which it has ordered all updates. Upon completion, correct servers (including the new representative) are reconciled such that they can constrain the updates proposed in the new view to preserve safety. The new representative then invokes ASSIGN-SEQUENCE to replay all pending updates contained in the *union* message.

Global View Change: The global view change protocol is triggered after a leading site election. The representative of the new leading site invokes CONSTRUCT-COLLECTIVE-STATE with the sequence number up to which it has ordered all updates. The resulting *union* message implicitly contains the sequence number up to which all updates have been ordered *within the site*; correct servers invoke THRESHOLD-SIGN on a message containing this number, and the representative sends the signed message to all other site representatives. Upon receiving this message, a non-leading site representative invokes CONSTRUCT-COLLECTIVE-STATE and sends the resultant *union* to the representative of the new leading site. Servers in the leading site use the *union* messages from a majority of sites to constrain the proposals they will generate in the new view.

4.3 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 protocols.

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 WAN 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 leadings 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 broadcast its last update. 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 it 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.

5 Performance Evaluation

To evaluate the performance of our hierarchical architecture, we implemented a complete prototype of our protocol including all necessary communication and cryp-

tographic 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 and assumed 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 experimental testbed consists of a cluster with twenty 3.2 GHz, 64 bit Intel Xeon computers. Each computer can compute a 1024 bit RSA signature in 1.3 ms and verify it in 0.07 ms. For n=16, k=11, 1024 bit threshold cryptography which we use for these experiments, a computer can compute a partial signature and verification proof in 3.9 ms and combine the partial signatures in 5.6 ms. The leader site was deployed on 16 machines, and the other 4 sites were emulated by one computer each. An emulating computer performed the role of a representative of a complete 16 server site. Thus, our testbed was equivalent to an 80 node system distributed across 5 sites. Upon receiving a message, the emulating computers busy-waited for the time it took a 16 server site to handle that packet and reply to it, including in-site communication and computation. We determined busy-wait times for each type of packet by benchmarking individual protocols on a fully deployed, 16 server site. We used the Spines [14] messaging system to emulate latency and throughput constraints on the wide area links.

We compared the performance results of the above system with those of BFT [6] on the same network setup with five sites, run on the same cluster. 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. 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. 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.

Note that, qualitatively, the results reported for BFT are not an artifact of the specific implementation we benchmarked. We obtained similar results to BFT using our BFT-like intra-site agreement protocol, ASSIGN-SEQUENCE, under the same conditions.

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 (emulating crossing the continental US).

In the first experiment, clients inject write updates. Figure 1 shows how limiting the capacity of wide area links effects update throughput. As we increase the number of clients, BFT's throughput increases at a lower slope than Steward's, mainly due to the additional wide area crossing for each update. Steward can process up to 84 updates/sec in all bandwidth cases, at which point it is limited by CPU used to compute threshold signatures. At 10, 5, and 2.5 Mbps, BFT achieves about 58, 26, and 6 updates/sec, respectively. In each of these cases, BFT's throughput is bandwidth limited. We also notice a reduction in the throughput of BFT as the number of clients increases. We attribute this to a cascading increase in message loss, caused by the lack of flow control in BFT. For the same reason, we were not able to run BFT with more than 24 clients at 5 Mbps, and 15 clients at 2.5 Mbps. We believe that adding a client queuing mechanism would stabilize the performance of BFT to its maximum achieved throughput.

Figure 2 shows that Steward's average update latency slightly increases with the addition of clients, reaching 190 ms at 15 clients in all bandwidth cases. As client updates start to be queued, latency increases linearly. BFT exhibits a similar trend at 10 Mbps, where the average update latency is 336 ms at 15 clients. As the bandwidth decreases, the update latency increases heavily, reaching 600 ms at 5 Mbps and 5 seconds at 2.5 Mbps, at 15 clients.

Adding Read-only Queries: Our hierarchical architecture enables read-only queries to be answered locally. To demonstrate this benefit, we conducted an experiment where 10 clients send random mixes of read-only queries and write updates. We compared the performance of Steward and BFT with 50 ms, 10 Mbps links, where neither was bandwidth limited. Figures 3 and 4 show the average throughput and latency, respectively, of different mixes of queries and updates. When clients send only queries, Steward achieves about 2.9 ms per query, with a throughput of over 3,400 queries/sec. Since queries are answered locally, their latency is dominated by two RSA signatures, one at the originating client and one at the servers answering the query. Depending on the mix ratio, Steward performs 2 to 30 times better than BFT.

BFT's read-only query latency is about 105 ms, and its throughput is 95 queries/sec. This is expected, as read-only queries in BFT need to be answered by at least $f+1$ servers, some of which are located across wide area links. BFT requires at least $2f+1$ servers in each site to guarantee that it can answer queries locally. Such a deployment, for 5 faults and 5 sites, would require at least 55 servers, which would dramatically increase communication for updates and reduce BFT's performance.

Wide Area Scalability: To demonstrate Steward's scalability on real networks, we conducted an experiment

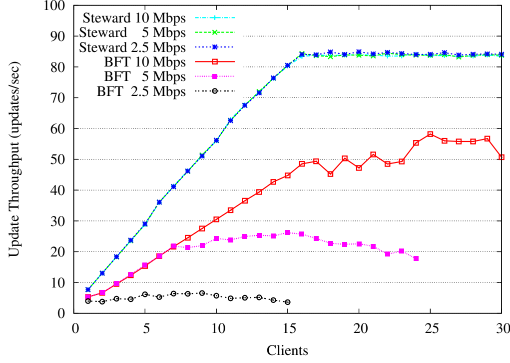

Figure 1: Write Update Throughput

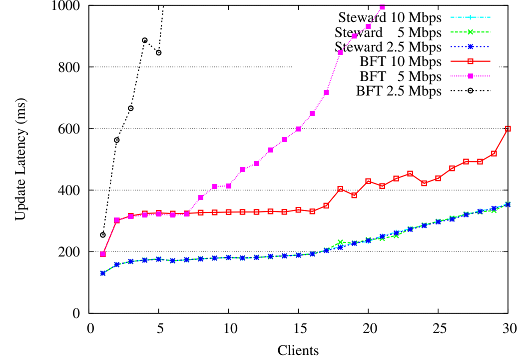

Figure 2: Write Update Latency

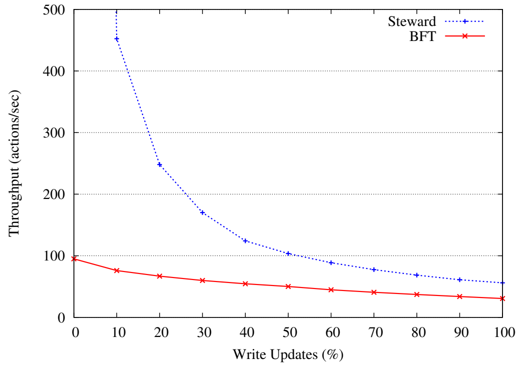

Figure 3: Update Mix Throughput - 10 Clients

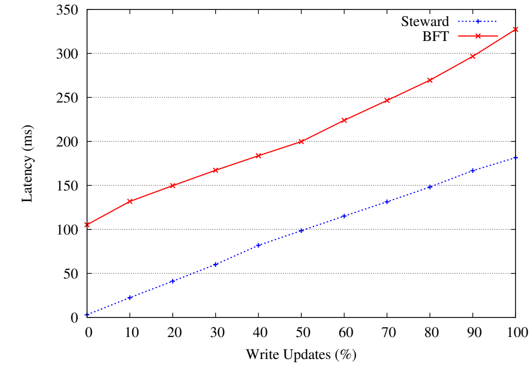

Figure 4: Update Mix Latency - 10 Clients

that emulated a wide area network spanning several continents. We selected five sites on the Planetlab network [15], measured the latency and available bandwidth between all sites, and emulated the network topology on our cluster. We ran the experiment on our cluster because Planetlab machines lack sufficient computational power. The five sites 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 1.3 Mbps (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. Steward is able to achieve its maximum throughput of 84 updates/sec with 27 clients. The latency increases from about 200 ms for 1 client to about 360 ms for 30 clients. BFT is bandwidth limited to about 9 updates/sec. The update latency is 631 ms for one client and increases to several seconds with more than 6 clients.

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 benign fault-tolerant replication protocols. We compare the performance of our hierarchical Byzantine architecture with that of two-phase commit protocols. In [16] we evaluated the performance of two-phase commit protocols [17] using a WAN setup across the US, called CAIRN [18]. 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 38 ms and 1.86 Mbps.

Figures 7 and 8 show the average throughput and latency of write updates, respectively, of Steward and BFT on the CAIRN network topology. Steward achieved about 51 updates/sec in our tests, 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 [16] was able to achieve 76 updates/sec. 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. BFT achieved a maximum throughput of 2.7 updates/sec and an update latency of over a second, except when there was a single client.

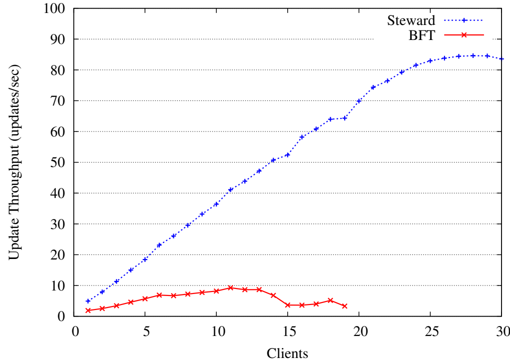

Figure 5: WAN Emulation - Write Update Throughput

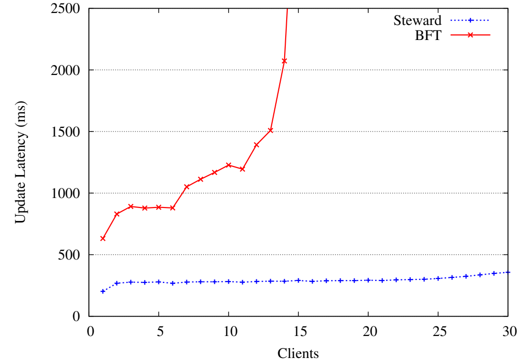

Figure 6: WAN Emulation - Write Update Latency

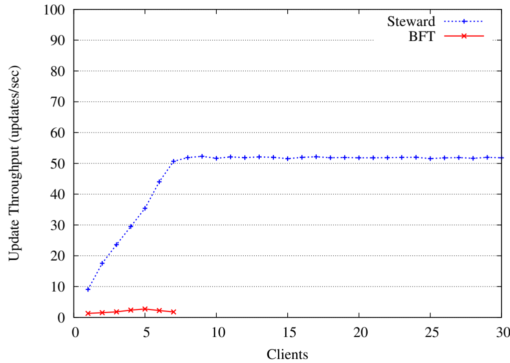

Figure 7: CAIRN Emulation - Write Update Throughput

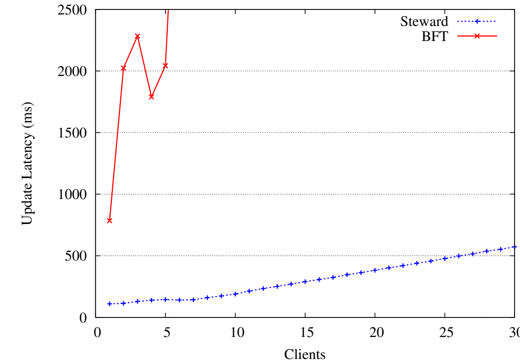

Figure 8: CAIRN Emulation - Write Update Latency

6 Related Work

Agreement and Consensus: At the core of many replication protocols is a more general problem, known as the agreement or consensus problem. A good overview of significant results is presented in [19]. The strongest fault model that researchers consider is the Byzantine model, where some participants behave in an arbitrary manner. If communication is not authenticated and nodes are directly connected, $3f+1$ participants and $f+1$ communication rounds are required to tolerate f Byzantine faults. If authentication is available, the number of participants can be reduced to $f+2$ [20].

Fail Stop Processors: Previous work [21] discusses the implementation and use of k-fail-stop processors, which are composed of several potentially Byzantine processors. Benign fault-tolerant protocols safely run on top of these fail-stop processors even in the presence of Byzantine faults. Steward uses a similar strategy to mask Byzantine faults. However, each trusted entity in Steward continues to function correctly unless $f+1$ or more computers in a site are faulty, at which point safety is no longer guaranteed.

Byzantine Group Communication: Related with our work are group communication systems resilient to Byzantine failures. Two such systems are Rampart [22] and SecureRing [23]. Although these systems are extremely robust, they have a severe performance cost and require a large number of uncompromised 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 [24, 25]. 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 [17] 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 [4] is a very robust algorithm for benign fault-tolerant replication and is described in Section 2.

Quorum Systems with Byzantine Fault-Tolerance: Quorum systems obtain Byzantine fault tolerance by ap-

plying quorum replication methods. Examples of such systems include Phalanx [26, 27] and its successor Fleet [28, 29]. 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 servers, it suffers from the drawbacks of flat 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 [6], which is described in Section 2. Yin et al. [30] propose separating the agreement component that orders requests from the execution component that processes requests, which allows utilization of the same agreement component for many different replication tasks and reduces the number of processing storage replicas to $2f + 1$. Martin and Alvisi [31] recently introduced a two-round Byzantine consensus algorithm, which uses $5f + 1$ servers in order to overcome f faults. This approach trades lower availability for increased performance. The solution is appealing for local area networks with high connectivity. While we considered using it within the sites in our architecture, the overhead of combining larger threshold signatures of $4f + 1$ shares would greatly overcome the benefits of using one less intra-site communication round.

Alternate architectures: An alternate hierarchical approach to scale Byzantine replication to WANs 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 performance.

Verissimo et al propose such a hybrid approach [32, 33], where synchronous, trusted nodes provide strong global timing guarantees. This inspired the Survivable Spread [34] 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, limited computers designed to be impenetrable.

Both the hybrid approach and the approach proposed in this paper can scale Byzantine replication to WANs. The hybrid approach makes stronger assumptions, while our approach pays more hardware and computational costs.

7 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 scalability. 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 WAN, 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 WANs.

Acknowledgments: Yair Amir thanks his friend Dan Schnackenberg for introducing him to this problem area and for conversations on this type of solution. He will be greatly missed.

This work was partially funded by DARPA grant FA8750-04-2-0232, and by NSF grants 0430271 and 0430276.

References

[1] A. Avizeinis, "The n-version approach to fault-tolerant software," *IEEE Transactions of Software Engineering*, vol. SE-11, pp. 1491–1501, December 1985.

[2] "Genesis: A framework for achieving component diversity, http://www.cs.virginia.edu/genesis/."

[3] 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 CSD TR 05-029, Purdue University, www.dsn.jhu.edu, December 2005.

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

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

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

[7] 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.
[8] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[9] V. Shoup, “Practical threshold signatures,” *Lecture Notes in Computer Science*, vol. 1807, pp. 207–223, 2000.
[10] 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.
[11] 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.
[12] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, “Robust threshold dss signatures,” *Inf. Comput.*, vol. 164, no. 1, pp. 54–84, 2001.
[13] F. B. Schneider, “Implementing fault-tolerant services using the state machine approach: A tutorial,” *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, 1990.
[14] “The spines project, http://www.spines.org/.”
[15] “Planetlab.” http://www.planet-lab.org/.
[16] 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.
[17] 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.
[18] “The CAIRN Network.” http://www.isi.edu/div7/CAIRN/.
[19] M. J. Fischer, “The consensus problem in unreliable distributed systems (a brief survey),” in *Fundamentals of Computation Theory*, pp. 127–140, 1983.
[20] D. Dolev and H. R. Strong, “Authenticated algorithms for byzantine agreement,” *SIAM Journal of Computing*, vol. 12, no. 4, pp. 656–666, 1983.
[21] R. D. Schlichting and F. B. Schneider, “Fail-stop processors: An approach to designing fault-tolerant computing systems,” *Computer Systems*, vol. 1, no. 3, pp. 222–238, 1983.
[22] 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.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] D. Malkhi and M. Reiter, “Byzantine quorum systems,” *Journal of Distributed Computing*, vol. 11, no. 4, pp. 203–213, 1998.
[28] 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.
[29] 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.
[30] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, “Separating agreement from execution for byzantine fault-tolerant services,” in *SOSP*, 2003.
[31] J.-P. Martin and L. Alvisi, “Fast byzantine consensus.,” in *DSN*, pp. 402–411, 2005.
[32] 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.
[33] P. Verissimo, “Uncertainty and predictability: Can they be reconciled,” in *Future Directions in Distributed Computing*, no. 2584 in LNCS, Springer-Verlag, 2003.
[34] “Survivable spread: Algorithms and assurance argument,” Tech. Rep. Technical Information Report Number D950-10757-1, The Boeing Company, July 2003.