Customizable Fault Tolerance for Wide-Area Byzantine Replication

Yair Amir[1], Brian Coan[2], Jonathan Kirsch[1], John Lane[1]
[1] Johns Hopkins University, Baltimore, MD. {yairamir, jak, johnlane}@cs.jhu.edu
[2] Telcordia Technologies, Piscataway, NJ. coan@research.telcordia.com

Technical Report CNDS-2006-3 - December 2006
http://www.dsn.jhu.edu

Abstract

This paper presents a hierarchical replication architecture, tailored to systems that span multiple wide-area sites, that enables free substitution of the fault tolerance method used in each position of the hierarchy. This unique approach enables customization based on perceived risks, balancing performance and fault tolerance, by deploying either a Byzantine or a benign fault-tolerant protocol at each site and in each level of the hierarchy. An implementation of four different compositions is evaluated over wide-area networks.

1 Introduction

As network environments become increasingly hostile, even well-protected distributed information systems, constructed with security in mind, are likely to be compromised. Some estimate the number of new compromised zombie nodes on the Internet per day to be as high as 250,000 [1]. It seems very likely that some of these zombies serve important functions in critical systems. Thus, there is a significant need for information systems that can function correctly even when an adversary gains control over part of the system.

Our previous work, Steward [4], was the first system to efficiently scale Byzantine replication to wide-area networks that span multiple sites, each consisting of several servers. Steward uses a hierarchical architecture in which Byzantine fault-tolerant protocols are run among the servers in each site to confine malicious behavior to the local level, allowing a benign fault-tolerant protocol to be run among sites at the wide-area level. Steward is tailored to mask Byzantine behavior locally under the assumption that less than 1/3 of the servers in any site will be compromised (a common assumption in state of the art Byzantine research). Steward demonstrated that a hierarchical solution yielded considerable performance gains over a flat architecture.

After Steward's introduction, several interesting observations were made. First, potential future users noted that in some environments, an entire site can be physically compromised, in which case running a local Byzantine fault-tolerant protocol is not necessary. In such cases, sites cannot be made trusted and, therefore, running a Byzantine fault-tolerant protocol over the wide-area, between the sites, is necessary. Second, running a wide-area Byzantine protocol over a local-area Byzantine protocol can meet the same guarantees as Steward at a lower hardware cost, at the price of one additional wide-area round. Third, different sites may have different risk profiles and therefore running some local sites with Byzantine replication and other sites with benign fault-tolerant replication is useful in certain environments (e.g., where trusted hardware unlikely to be compromised is deployed).

Steward's design targeted the best possible performance, and indeed achieved performance not far from common benign fault-tolerant wide-area replication protocols. However, its performance benefit came with a significant price:

Inflexibility: Steward is designed for the case where less than 1/3 of the servers in each site will be compromised. The techniques used to mask Byzantine behavior are interdependent with the techniques used to achieve efficient wide-area global ordering. As a result, the different protocols in each level cannot be replaced to support other threat models such as complete site compromises.

Complexity: In order to save message exchange rounds and computation, the local and wide-area components of Steward are intertwined. The resultant protocol is globally optimized but its specification and correctness proof are very complex (see [3] for this complexity).

These observations convinced us that, to be practical in a diverse, wide-area environment, we must evolve our ap-

proach, even at the expense of decreased performance. A desired solution will have two key properties: *free substitution* and *clean separation*. Free substitution of the fault tolerance approach used in each position of the hierarchy allows an administrator to customize a system based on perceived risks, balancing performance and fault tolerance, by deploying either a Byzantine or a benign fault-tolerant protocol at each site and in each level of the hierarchy. Clean separation of the protocols running within each site and in each level of the hierarchy reduces the complexity of the overall protocol and its correctness proof.

This paper describes a composable, hierarchical architecture that provides these two properties. Similar to Steward, we convert the physical machines within each site into a logical entity that plays the role of a single participant in the wide-area protocol. However, our new architecture implements this conversion in a fundamentally different way. We build on the well-known technique of state machine replication [16, 29] to convert the servers in a site into a *logical machine*. The servers composing each logical machine totally order *all* events that cause a state transition in the wide-area protocol (i.e., updates, acknowledgements, and wide-area timeouts), and execute these events in the same order. Hence, the logical machine runs the wide-area protocol just as if it were implemented by a single server. The local protocol is oblivious to the state of the wide-area protocol. This separation enables free substitution.

The new architecture brings to light two new challenges that must be addressed: performance and efficient wide-area communication. Since we remove protocol-aware global optimizations, we must gain performance in some other way to make the approach practical. In addition, we must provide a mechanism that allows two logical machines, each composed of potentially malicious servers, to efficiently send messages to each other. The mechanism should not depend on the protocols running in each logical machine and should usually send a message only once.

The contributions of this work are:

1. It is the first solution that allows a system administrator to find the best way to deploy his system, meeting required guarantees by separately choosing either benign or Byzantine fault tolerance in each of the sites, as well as choosing either a benign or Byzantine fault-tolerant wide-area protocol. This, for example, allows the system administrator to have autonomy in trading off hardware costs and wide-area crossings. In particular, it is the first solution that composes a Byzantine fault-tolerant protocol between the sites over a Byzantine fault-tolerant protocol in each site, providing stronger security guarantees.

2. It presents a novel protocol that facilitates efficient wide-area communication between logical machines, each of which is constructed from several non-trusted entities, such that messages usually require one send over the wide-area network.

3. It presents optimizations that reduce the computational cost of the logical machines, improving the logical machine local throughput so that it can play a more responsive role on the wide area. As a result, it increases the overall maximum wide-area throughput by a factor of 4 compared with the previous state of the art.

We compare, both analytically and experimentally, four possible compositions of the architecture, plus the Steward architecture, over emulated wide-area networks. The experiments show that the composable architecture that runs a wide-area benign fault-tolerant protocol and Byzantine local-area protocols within each site has performance that is within 12 percent of Steward. Furthermore, the composition that runs a wide-area Byzantine fault-tolerant protocol on the wide area and Byzantine fault-tolerant protocols in each site is within 35 percent of the performance of Steward. This composition provides much stronger fault tolerance, surviving more than twice the number of total Byzantine faults compared with Steward, and up to more than a third of the total number of servers in the system in some cases.

We believe this new approach is a considerable step forward in making wide-area Byzantine replication a reality.

The remainder of this paper is presented as follows. In Section 2, we provide background on state machine replication and the fault-tolerant protocols used by our customizable architecture. Section 3 describes our system model and service guarantees. In Section 4, we describe our system architecture. Section 5 presents solutions to two key technical challenges in achieving high performance with the composable architecture. In Section 6, we evaluate the performance of our architecture. Section 7 provides discussion, and Section 8 concludes.

2 Background and Related Work

Our work uses techniques from fault-tolerant replication, cryptography, and Byzantine fault-tolerant protocols and architectures. In this section, we describe related work most relevant to our new architecture.

State Machine Replication: Lamport [16] and Schneider [29] introduced and popularized state machine replication, where deterministic replicas execute a totally ordered stream of events that cause state transitions. Therefore, all replicas proceed through exactly the same states. This technique can be used to implement replicated information access systems, databases, and other services. Schlichting and Schneider [28] discuss the implementation and use of k-fail-stop processors, which are composed of several poten-

tially Byzantine processors. Benign fault-tolerant protocols safely run on top of these fail-stop processors even in the presence of Byzantine faults.

Paxos and BFT: Paxos [17, 15] is a fault-tolerant protocol that enables a group of distributed servers, exchanging messages via asynchronous communication, to totally order client requests in a benign fault, crash-recovery model. Paxos uses a leader to coordinate an agreement protocol. If the leader fails, the other servers elect a new leader, which coordinates sufficient reconciliation so that progress can safely continue. In the normal case, when the leader does not fail, Paxos requires two communication rounds to order a message, one of which is an all-to-all message exchange. Paxos continues to order client updates if at least $f + 1$ out of $2f + 1$ servers are connected and functioning correctly. BFT [5] also totally orders client request, similar to Paxos. However, it tolerates Byzantine faults, where compromised servers behave maliciously in an attempt to disrupt the system. BFT uses three communication rounds, two of which are all-to-all message exchanges. It can survive f Byzantine server failures out of a total of $3f + 1$. BASE [27] describes an abstraction that is built upon BFT and gives examples of how to use this abstraction to build Byzantine fault-tolerant services. We use a similar abstraction to convert the servers in one site into a logical machine.

Steward: Steward [4] is a hierachical state machine replication architecture for wide-area networks. Conceptually, it converts a group of servers in a site into a logical entity that plays the role of a single participant in a wide-area protocol. However, it does not use state machine replication to create logical machines. Instead, it uses a series of customized Byzantine fault-tolerant protocols, enabling a group of servers to emulate a participant in the Paxos protocol. Steward can withstand f out of $3f + 1$ Byzantine failures within each site but cannot survive even a single site compromise.

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 [11]. 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 $f + 2$ [9].

Replication with Benign Fault Tolerance: The two-phase commit (2PC) protocol [10] 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 [31] overcomes some of the availability problems of 2PC, paying the price of an additional communication round, and therefore, additional latency.

Replication with Byzantine Fault Tolerance: Yin et al. [33] describe a Byzantine fault-tolerant replication architecture that separates the agreement component that orders requests from the execution component that processes them. Their architecture reduces the number of storage replicas to $2f+1$ and provides a privacy firewall, which prevents a compromised server from divulging sensitive information. Martin and Alvisi [22] recently introduced a two-round Byzantine consensus algorithm, which uses $5f + 1$ servers to overcome f faults. This approach trades lower availability for increased performance. Cowling et al. [7] recently introduced a hybrid approach to achieving state machine replication that uses quorum-based methods when there is no contention and BFT to resolve contention by ordering the contending operations.

Byzantine Group Communication: Related with our work are group communication systems resilient to Byzantine failures. Two such systems are Rampart [25] and SecureRing [14]. 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 [8, 24]. 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.

Quorum Systems with Byzantine Fault-Tolerance: Quorum systems obtain Byzantine fault tolerance by applying quorum replication methods. Examples of such systems include Phalanx [21, 18] and its successor Fleet [19, 20]. 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.

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 performance. Verissimo et al. propose such a hybrid approach [6, 32], where synchronous, trusted nodes provide strong global timing guarantees.

3 System Model and Service Guarantees

We achieve replication via the state machine approach [16, 29]. 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. As described in Section 4, non-deterministic events are handled by making them appear deterministic to the wide-area protocol.

Servers are organized into wide-area *sites*, each having a unique identifier. Each server belongs to one site and has a unique identifier within that 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. The number of servers within each site varies with the desired level of fault tolerance within the site. If a benign fault-tolerant protocol (such as Paxos) is deployed within a site, then we assume there are at least $2f + 1$ servers within the site, where f is the maximum number of servers that may be faulty. If a Byzantine fault-tolerant protocol (such as BFT) is deployed, then we assume there are at least $3f + 1$ servers within the site, where at most f servers may be Byzantine.

The architecture presented in this paper supports a rich configuration space, in terms of both the number of faults and the types of faults allowed at each level of the hierarchy and within each site. Our system (unlike Steward) can still guarantee correct and live execution even when the fault assumptions made within some of the sites are violated; this is achieved by running a Byzantine fault-tolerant protocol among sites. We say that a site is Byzantine if (1) it is running a local benign fault-tolerant protocol and at least one server is Byzantine or (2) it is running a local Byzantine fault-tolerant protocol and more than f servers are Byzantine. The number of sites needed in this case is dependent on the wide-area protocol choice, but will be at least $3F + 1$, where F is the maximum number of *sites* that may be Byzantine.

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 sequence number. Clients propose updates sequentially: a client may propose an update with sequence number $i + 1$ only after it receives a reply for an update with sequence number i. Clients may be faulty; updates from faulty clients will be replicated consistently. Access control techniques can be used to restrict the impact of faulty clients.

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.

When Byzantine fault-tolerance is deployed within a site, the servers in that site use an $(f + 1, 3f + 1)$ threshold digital signature scheme [30]. 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 $f+1$ or more shares.

As in Steward, our system achieves replication by establishing a global, total order on updates in the wide-area protocol and executing the updates in this order. Below we define the safety and liveness properties provided by our replication system. 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* a wide-area protocol 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* or *a client and server are connected* if any message that is sent between them will arrive in a bounded time. The protocol participants need not know this bound beforehand.
- *two sites are connected* if every correct server in one site is connected to every correct server in the other.
- *a client is connected to a site* if it can communicate with all correct servers in that site.

We define the following two safety conditions:

DEFINITION 3.1 S1 - SAFETY: *If two correct servers execute the i^{th} update, then these updates are identical.*

DEFINITION 3.2 S2 - VALIDITY: *Only an update that was proposed by a client may be executed.*

Since no asynchronous, fault-tolerant replication protocol can always be both safe and live [12], we provide liveness under certain synchrony conditions. We use the following terminology to encapsulate these conditions:

- A *site is stable* with respect to time T if there exists a set, S, of c servers within the site (with $c = 2f + 1$ for sites tolerant to Byzantine failures and $c = f + 1$ for sites tolerant to benign failures), where, for all times $T' > T$, the members of S are (1) correct and (2) connected. We call the members of S *stable servers*.

- Let N be the total number of sites in the system and F be the maximum number of sites that may be faulty. The *system is stable* with respect to time T if there exists a set, W, of r wide-area sites (with $r > \lfloor N/2 \rfloor$ when sites may exhibit benign failures and $r = 2F+1$ when sites may exhibit Byzantine failures) where, for all times $T' > T$, the sites in W are (1) stable with respect to T and (2) connected. We call the sites in W the *stable sites*.

We now define our liveness property:

DEFINITION 3.3 L1 - GLOBAL LIVENESS: *If the system is stable with respect to time T, then if, after time T, a stable server receives an update which it has not executed, then that update will eventually be executed.*

4 System Architecture

Our new composable architecture provides flexibility by cleanly separating the wide-area protocol and the local-area protocols run in each site. In this section we present a high-level description of the components of our architecture and explain how they lead to protocol separation.

The system uses two representative, well-known, flat replication protocols: Paxos [17, 15] as our lightweight, benign fault-tolerant protocol, and BFT [5] as our Byzantine fault-tolerant replication protocol. Each of these protocols can be selected for use within each site and on the wide area. The local-area protocol is used to totally order wide-area protocol messages that come into the site, enabling the servers within a site to execute the wide-area protocol and act as a single logical machine. The wide-area replication protocol, which runs on the logical machines, assigns a global order to updates submitted by clients.

When BFT is run within a site, a logical machine will function correctly if less than one third of the servers in the site are compromised. When BFT is run on the wide-area, the system will function correctly if less than one third of the sites are compromised. When Paxos is run within a site, the logical machine will function correctly if less than a majority of servers suffer benign faults. When Paxos is run on the wide-area, the system will function correctly if less than a majority of sites suffer benign failures. In the remainder of this paper, we refer to compositions as *wide-area protocol/local-area protocol*. For example, we refer to a composition which runs BFT on the wide-area and Paxos on the local area as BFT/Paxos. Even though our architecture uses BFT and Paxos, it is extensible and can use any of several existing state of the art flat replication protocols.

In our composable architecture, the top-level replication protocol used on the wide area runs just as it would if it were run on a series of single machines where each machine was located in its own site. It sends the same types of wide area messages and has the same communication patterns. During view changes, which occur when a leader site fails and needs to be replaced, exactly the same messages are sent as in the original flat protocol. Thus, the wide-area bandwidth requirements and message complexity in a composition with Paxos running on the wide area are the same as if a flat Paxos system were run on a single machine in each site. More significantly, we can use the known safety proof for flat Paxos, together with one for the local-area protocol, and trivially prove safety for the composition. The liveness proof is more complicated, but much simpler than what is necessary for a globally optimized architecture such as Steward where the wide-area and local-area protocols are interdependent.

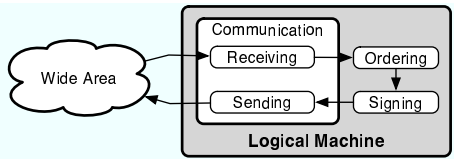

Figure 1: **Logical machine.**

Logical Machine: A **logical machine** (LM) is the group of servers within a site that, together, play the role of a single participant in the wide-area protocol. We use the state machine approach to build our logical machines [29] [16]. We say that the servers within a site implement the LM corresponding to their site. To support the logical machine abstraction, the servers implementing a given LM use the agreement protocol (Paxos or BFT) to establish a total ordering on all wide-area protocol events; the servers then handle the events according to the total order. Consequently, the logical machine handles, or *executes*, a single stream of wide-area protocol events, just as if it were implemented by a single correct server. The logical machine is composed of three modules, each providing a necessary service: communication, ordering, and signing (Figure 1).

The **Communication Module** provides efficient wide-area message transmission even in a Byzantine environment, where some of the servers that send messages from one logical machine to another try to block this communication. This module is part of our composable architecture and is described in detail in Section 5.1.

The **Ordering Module** uses Paxos or BFT to totally order the protocol events handled by the servers in an LM. These events include wide-area protocol timeouts, wide-area messages, and local timeouts used to implement the LM. The ordering module delivers those events related to the wide-area protocol to the LM, which executes them. We use a technique similar to BASE [27] to handle non-deterministic events. To implement a logical wide-area protocol timeout, each server in the site sets a local timer, and when this timer expires, it sends a signed message to the

leader of the ordering module. The leader waits for $f + 1$ signed messages proving that the timer expired at at least one correct server and then orders a logical timeout message (containing this proof). A server only transitions state based on a timeout when it receives an ordered logical timeout message. The ordering module corresponds to the local replication protocol run within a site.

The **Signing Module** generates an RSA signature [26] for wide-area messages. When running Paxos within the site (in the ordering module), this module simply generates an RSA signature on the message. When running BFT within the site, this module generates an RSA threshold signature, attesting to the fact that $f + 1$ servers agreed on the message. This prevents malicious servers within a site from forging a message. Moreover, outgoing messsages carry only a single RSA signature, saving wide-area bandwidth. Our optimized architecture amortizes the high cost of threshold cryptography over many outgoing messages. We use a technique similar to Steward to prevent malicious servers from disrupting the threshold signature protocol.

We conclude by providing an example of Paxos/BFT that traces the flow of an update through the system in the normal case, when there are no faults that require changing protocol coordinators. First, a client sends an update to a server in its own site, which forwards the update to the leader site, using the communication module. The ordering module of the leader site logical machine uses BFT, which requires three local communication rounds, to order the update. The logical machine generates a wide-area proposal message, binding a global sequence number to the update. The message proceeds to the signing module, which uses a one-round protocol to generate a threshold signature. The communication module sends the threshold-signed proposal to the other sites. Each non-leader logical machine orders the incoming proposal and generates an acknowledgement (accept) message for the proposal. The logical machine sends the acknowledgement to the other logical machines. Incoming accept messages are then ordered by the logical machine. When the proposal and a majority of accepts are collected, the logical machine globally orders the client update, completing the protocol. This example elucidates that there are many rounds, most of which are associated with the ordering module. This is the price to achieve the clean separation and free substitution properties.

5 Technical Challenges and Solutions

In this section, we present solutions to two technical challenges in achieving efficient, high-performance replication in a composable architecture.

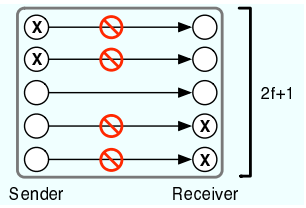

Figure 2: **In any set of 2f+1 (forwarder, peer) link pairs, at least one pair must have both servers correct. In this example, f=2.**

5.1 Byzantine-resilient Communication

We first address the issue of how logical machines can efficiently communicate over wide-area links, which have relatively low bandwidth compared to local-area links. Since each logical machine is implemented by a replicated group of machines, some of which may be faulty, care must be taken to ensure that messages are successfully sent and received despite faulty behavior, without consuming too much extra wide-area bandwidth. For example, the naive solution, in which each of $f + 1$ replicas in the sending LM sends to $f + 1$ replicas in the receiving LM, results in an impractical O(f^2) bandwidth increase. Ideally, each logical machine will use roughly the same wide-area bandwidth as a single physical machine.

We now present a new **link protocol** that allows logical machines to efficiently communicate with each other over the wide-area network, regardless of the protocols they are running. The protocol masks faulty behavior at its endpoints and provides a reliable transport mechanism between the sending logical machine and the receiving logical machine. The link protocol is implemented by the sending and receiving parts of the communication module.

The link protocol is built upon three simple but powerful techniques. The first technique provides a novel way of delegating the responsibility for wide-area communication such that (1) messages are normally sent only once and (2) the adversary is unable to repeatedly block communication between two logical machines. The second technique leverages the power of threshold cryptography and state machine replication to allow the servers in the sending logical machine to monitor the behavior of the link and take action if it appears to be faulty; it also provides reliability. The third technique is a fairness mechanism that prevents the adversary from starving any particular client or link.

Delegating Communication Responsibility: Servers within the sending logical machine elect a *forwarder* for each outgoing link. This election is done independently for each link; different servers may act as forwarder on different links, or the same server may act as forwarder on multiple links. Each potential forwarder is statically assigned one unique *peer* server, in the receiving logical machine, to which it sends outgoing messages and which dis-

seminates incoming messages. This assignment is made using the servers' unique identifiers: the two servers with the same identifier in the sending and receiving logical machines form a (forwarder, peer) pair. Given two logical machines, each with $3f + 1$ servers, the pigeonhole principle guarantees that, out of $2f + 1$ (forwarder, peer) pairs, there exists at least one pair in which both participants are correct (see Figure 2) Since either the forwarder or the peer may be faulty, the other servers within the sending logical machine monitor the performance of the link and elect a different forwarder if the current one is not performing well enough (we define this notion more precisely below). The successor is the next server in a static order, with wrap-around, placed on all servers in the sending logical machine. Thus, after rotating through at most $2f + 1$ pairs, the link is guaranteed to reach a pair with two correct endpoints, allowing messages to flow from the sending machine to the receiving machine.

When the Byzantine fault-tolerant link protocol is run between two logical machines consisting of different numbers of servers, the forwarder and peer can be chosen as follows. Let $L1$ and $L2$ be two logical machines, and let F be the number of faults that $L1$ can survive and f be the number of faults that $L2$ can survive. Let $F \geq f$. $L1$ has a total of $3F+1$ servers, and $L2$ has a total of $3f+1$ servers. We assign $3F + 1$ distinct pairs of peers, where one of the peers is in $L1$ and the other is in $L2$. Each of the $3F + 1$ servers in $L1$ is a member of exactly one pair. We assign pairs round robin, based on unique server identifiers. Each server in $L2$ is paired with the servers in $L1$ that have equal identifiers modulo $3f + 1$.

We now present a proof sketch showing that when the pairs are assigned as described, there will be at least one peer pair where both servers are correct. A single server in $L1$ is a member of exactly one pair. It follows that each of the F malicious servers in $L1$ can corrupt one pair. A single server in $L2$ can be a member of no more than $c_2 = \lceil \frac{3F+1}{3f+1} \rceil$ pairs. Since there are at most f malicious servers in $L2$, these servers can corrupt at most $f * c_2$ pairs. Therefore, together, the malicious servers in $L1$ and $L2$ can corrupt at most $c = f\lceil \frac{3F+1}{3f+1} \rceil + F$ pairs. Since there are a total of $3F + 1$ links, we need to show that $c < 3F + 1$. We can rewrite this relationship as $c \leq f\left(\frac{F}{f} + 1\right) + F < 3F + 1$. Simplifying yields $c \leq 2F + f < 3F + 1$. This relationship is true because $F > f$. This completes our proof sketch.

Reliability and Monitoring: The link protocol uses threshold-signed, cumulative acknowledgements to ensure reliability. Each message sent on an outgoing logical link is assigned a link-specific sequence number. Assigning these sequence numbers consistently is simple, since outgoing messages are generated in response to totally-ordered wide-area protocol events and can be sequenced using this total order. Each logical machine periodically generates a threshold-signed acknowledgement message, which contains, for each link, the sequence number through which the logical machine has received all previous messages. The generation of the acknowledgement is triggered by executing a logical machine timeout [1]. The peer server for each incoming link sends the acknowledgement to its corresponding forwarder, which presents the acknowledgement to the servers in the sending logical machine.

The acknowledgement serves two purposes. First, it is used to determine which messages need to be retransmitted over the link in order to achieve reliability. This reliability is guaranteed even if the current forwarder is replaced, since the next forwarder knows exactly which messages remain unacknowledged and should be resent. Second, the servers in the sending logical machine use the acknowledgement to evaluate the performance of the current forwarder. Each server in the sending logical machine maintains a queue of the unacknowledged messages on each link, placing a logical machine timeout on the acknowledgement of the first message in the queue. If, before the timeout expires, the forwarder presents an acknowledgement indicating the message was successfully received by the receiving logical machine, the timeout is canceled and a new timeout is set on the next message in the queue [2]. However, if the timeout expires before such an acknowledgement is received, the servers in the sending logical machine suspect that the link is faulty and elect the next forwarder.

Fairness: The third technique addresses the dependency between the evaluation of the link forwarder and the performance of the ordering module at the receiving logical machine. Intuitively, if the ordering module could selectively refuse to order certain messages or could delay them too long, then a correct forwarder might not be able to collect an acknowledgement in time to convince the other servers that it sent the messages correctly. We would like to settle on a correct (forwarder, peer) pair to the extent possible, and thus we augment the ordering module with a fairness mechanism, tailored to meet the needs of the link protocol.

Conceptually, there are two types of input events to the ordering module: link-based events and client-based events. When a peer in a receiving logical machine receives an incoming message, it disseminates the message within the site; all servers then expect the ordering module to order the message such that it can be executed by the logical machine. Similarly, client updates must be ordered by the leader site logical machine so that they are processed in the wide-area protocol. To ensure fairness, servers must place a timeout on the leader of the ordering module to prevent the selective starvation of a particular link or client.

Servers within the logical machine maintain a queue for

[1] Servers could also piggy-back acknowledgements on regular outgoing messages for more timely, fine-grained feedback.

[2] This mechanism can be augmented to enforce a higher throughput of acknowledged messages by placing a timeout on a batch of messages.

each incoming link and a single queue for all clients. When the leader receives a message to be ordered, it places the message on the appropriate queue (i.e., on a link queue or on the client queue). The leader then attempts to order messages off of the queues in round-robin fashion. Since incoming link messages have link-based sequence numbers, all servers know exactly which message should be the next one ordered for each link. Thus, upon receiving the next message on a link, a server places a timeout on the message and attempts to replace the leader if the message is not ordered in time. Since there does not exist a total order on all client updates, servers allow the leader to order *any* client update from the client queue. However, the servers suspect the leader if it tries to order two updates from the same client without ordering one from another client for which an update is in the queue.

5.2 Performance Optimizations

Our composable architecture has significant computational overhead, because each logical machine must order all events that cause state transitions in the wide-area protocol. This Byzantine fault-tolerant ordering (which in our architecture uses digital signatures) is computationally costly. In addition, each logical machine threshold signs all outgoing messages, which imposes an even greater computational cost. Consequently, we use Merkle hash trees to amortize the cost of threshold signing, and we improve the performance of logical machine event processing via aggregation. These optimizations are applied *only* to the local protocols. Thus, there is a one-to-one correspondence between wide-area messages in an optimized, composable protocol and its unoptimized equivalent.

Merkle Tree Based Signatures: We first consider how to amortize the cost of generating threshold signatures, which are applied to outgoing messages. Instead of threshold signing every outgoing message, we generate a single threshold signature that can be used to authenticate several messages. Each outgoing message is completely self-contained, including everything necessary to validate the message (except the public key). We accomplish this by creating a signature based on a Merkle hash tree [23].

The leaf nodes in a Merkle hash tree contain the hashes of the messages that need to be sent. Each of the internal nodes contains a hash of the concatenation of the two hashes in its children nodes. The signature is generated over the hash contained in the root. When a message is sent, we include the series of hashes that can be used to generate the root hash. The number of included hashes is $log(N)$, where N is the number of messages that were signed with the single signature.

Logical Machine Event Processing: We now consider how to increase the throughput of local event processing by the logical machine. We use aggregation to order several logical machine events at once. This improvement allows a logical machine to order thousands of events per second over local-area networks while providing Byzantine fault tolerance. With this performance, it is likely that the incoming wide-area bandwidth will limit throughput. Events flowing into the ordering module are buffered. When the leader of the ordering module is ready to begin the Byzantine fault-tolerant ordering protocol (by sending a BFT Pre-Prepare message to the other servers), it takes all of the events in the buffer (up to approximately 50) and creates a Pre-Prepare message that can order all of them using one Byzantine agreement invocation. The Pre-Prepare includes digests corresponding to each event that was in the buffer. Note that this does not limit the size of incoming messages being ordered.

Protocol Rounds			
Protocol	Wide Area	Local Area	Total
Steward	2	4	6
Paxos/Paxos	2	6	8
BFT/Paxos	3	8	11
Paxos/BFT	2	11	13
BFT/BFT	3	15	18

Table 1: Number of Protocol Rounds.

6 Performance Evaluation

To evaluate the performance of our composable architecture, we implemented our protocols, including all necessary communication and cryptographic functionality.

Testbed and Network Setup: We used a network topology consisting of 5 wide-area sites, each containing 16 physical machines, to quantify the performance of our system. In order to facilitate comparisons with Steward, we chose to use the same topology and numbers of machines used in [4]. If BFT is run within a site, then the site can tolerate up to 5 Byzantine servers. If Paxos is run within a site, then the site can tolerate 7 benign server failures. If BFT is run on the wide-area, then the system can tolerate one Byzantine site compromise. If Paxos is run on the wide area, then the system remains available if no more than two sites are disconnected from the others.

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=6, 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 3.4 ms. The leader site was fully deployed on 16 machines, and the other 4 sites were emulated by one computer each.

Protocol Computational Costs		
Protocol	Threshold RSA Sign	RSA Sign
Steward	1	3
Paxos/Paxos	0	$2 + (S - 1)$
BFT/Paxos	0	$3 + 2(S - 1)$
Paxos/BFT	1	$3 + 2(S - 1)$
BFT/BFT	2	$4 + 4(S - 1)$

Table 2: Number of expensive cryptographic operations that each server at the leader site does for one update.

Each emulating computer performed the role of a representative of a complete 16 server site. Therefore, our testbed is 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 intra-site communication and computation. We also modeled the aggregation used by the optimized protocols. We determined busy-wait times for each type of packet by benchmarking the different types of ordering protocols on a fully deployed, 16 server site. The Spines [2] messaging system was used to emulate latency and throughput constraints on the wide-area links. We limited the capacity of wide-area links to 10 Mbps in all tests.

We compared the performance results of five protocols with and without optimizations. Four of these are based on our composable architecture: Paxos/Paxos, BFT/Paxos, Paxos/BFT, BFT/BFT. The fifth is an augmented implementation of Steward, which includes the option of using the same optimization techniques used in our new architecture. All the write updates in our experiments carried a payload of 200 bytes, representative of an SQL statement.

We exclusively use RSA signatures for authentication, both for consistency with our previous work and to provide non-repudiation, which is valuable when identifying malicious servers. The benign fault-tolerant protocols use RSA signatures to protect against external attackers. While it is possible to use more efficient cryptography in the compositions based on Paxos, these changes improve the unoptimized protocols but have a smaller effect on the optimized versions. We also note that BFT can use MACs, which improves its latency and results in much better unoptimized performance. However, this change has a smaller effect on our optimized protocols, because the total update latency is dominated by the wide-area latency.

Protocol Rounds and Cryptographic Costs: Table 1 shows the number of protocol rounds in Steward, and in each of the four combinations of our composable architecture. The protocol rounds are classified as wide-area when the message is sent between sites, and local-area when it is sent between two physical machines within a site. Table 1 shows that Steward has the least rounds of any of the protocols, including Paxos/Paxos. The difference in total rounds ranges from 6 (Steward) to 18 (BFT/BFT). However, it is important to observe that all of the protocols listed have either two or three wide-area rounds.

Table 2 shows the computationally expensive cryptographic operations required for each update at the leader site in the unoptimized protocols. The costs are a function of the number of sites, denoted by S. The table shows the number of threshold signatures to which each server in the leader must contribute and the number of RSA signatures that each server in the leader site must compute. In the tests presented in this paper, the unoptimized versions of our algorithm are always limited by computational resources. Consequently, these costs are inversely proportional to the maximum throughput. Our throughput tests match what one would expect based on the costs presented in Table 2.

Performance of Unoptimized Protocols: We first compare the performance of the five unoptimized protocols. Each time an external message is ordered or a threshold signed message is produced, the protocols do all cryptographic operations listed in Table 2. The following two tests were performed on a symmetric configuration where all sites are connected to each other with the same latency and bandwidth constraints. We used a 50 ms (emulating crossing the continental US) and 100 ms (to demonstrate scalability) network diameter. Clients inject updates into the system by sending them to a server in their site. A client waits until it receives proof that the update was ordered and then immediately injects the next update.

Figure 3 shows update throughput as a function of the number of clients. In all of the graphs, throughput initially increases as the number of clients increases. When the load on the CPU increases to 100%, throughput plateaus. This graph shows the performance benefit of Steward's optimized architecture. In Steward, external wide-area accept messages do not need to be ordered before the replicas can process them. Steward achieves over twice the performance of Paxos/BFT, its equivalent composition. This is the price of clean separation. Steward even outperforms Paxos/Paxos, which has more ordering and RSA signature generation, but does not use threshold signatures. The initial slope of these curves is most dependent on the number of wide-area protocol rounds shown in Table 1. The peak performance of each of the protocols is a function of the number of cryptographic operations shown in Table 2. The Paxos/BFT composition has about twice the throughput of the BFT/BFT composition, and it has approximately half of the cryptographic costs. A similar relationship exists between Paxos/Paxos and BFT/Paxos.

Figure 4 shows average update latency measured at the clients as a function of the number of clients. In each of the curves, the update latency remains approximately constant until the CPU is 100% utilized, at which point, latency climbs as the number of clients increases. In our system, we queue client updates if the system is overburdened and in-

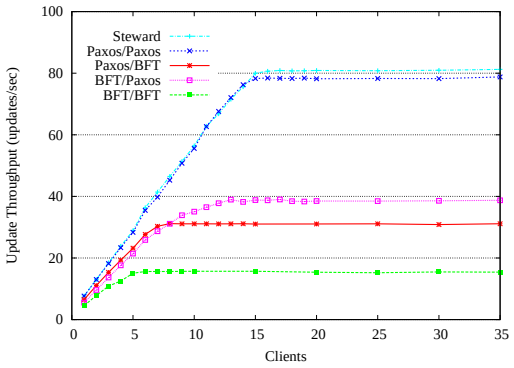

Figure 3: Throughput of Unoptimized Protocols, 50 ms Diameter

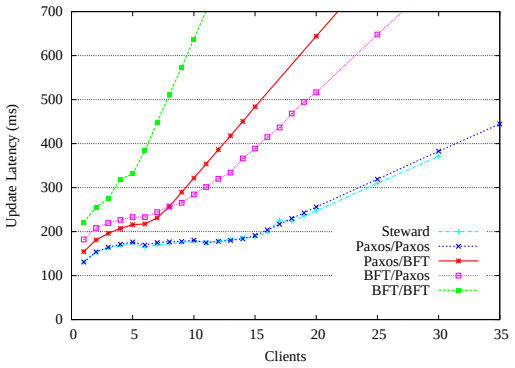

Figure 4: Latency of Unoptimized Protocols, 50 ms Diameter

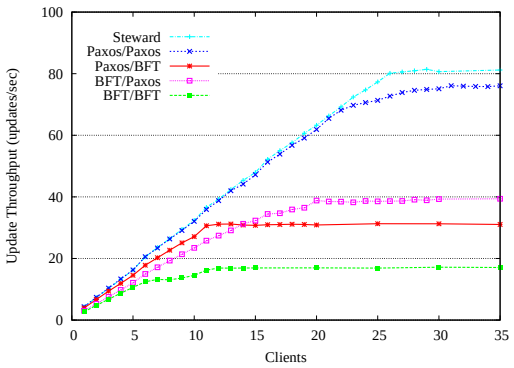

Figure 5: Throughput of Unoptimized Protocols, 100 ms Diameter

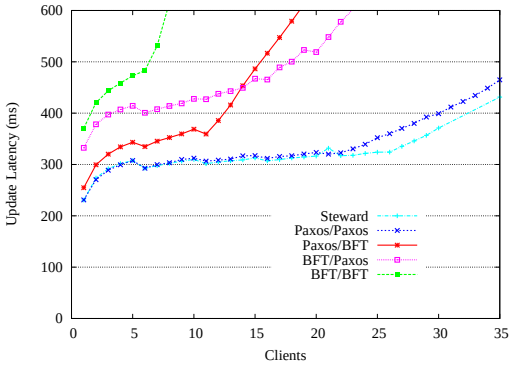

Figure 6: Latency of Unoptimized Protocols, 100 ms Diameter

ject these updates in the order in which they were received.

Figures 5 and 6 show the results for the same tests as above with 100 ms network diameter. We observe the same maximum bandwidth and latency trends. Additional latency on the wide-area links reduces the slope of the lines in Figure 5 (update throughput), but has no effect on the maximum throughput that is achieved.

Performance of Optimized Protocols: We now present the performance of the five protocols with the optimizations described in Section 5.2. In these protocols, the cost of the cryptographic operations listed in Table 2 are amortized over several updates when CPU load is high. In contrast to the unoptimized protocols, none of our optimized protocols were CPU limited in the following tests. Maximum throughput was always limited by wide-area bandwidth constraints. In all cases, the optimized protocols increased throughput by at least a factor of 4 compared to their unoptimized versions.

Figure 7 shows the update throughput as a function of the number of clients. The relative maximum throughput and slopes of the curves are very different from the unoptimized versions. For example, Paxos/Paxos, Steward, and Paxos/BFT have almost the same maximum throughput. This attests to the effectiveness of the optimizations in greatly reducing the performance overhead associated with clean separation. The optimization improves the performance of the compositions more than it improves Steward because the composable architecture uses many more local rounds. In a wide-area environment, local rounds are relatively inexpensive *if* they do not consume too much computational resources. The optimizations eliminate the computational bottleneck of the unoptimized protocols. Thus, performance of the optimized version is predominantly dependent on the number of wide-area protocol rounds.

The local-area protocol has a smaller, but significant, effect on performance. The slopes of the curves are different because of the difference in latency contributed by the local-area protocols. BFT and threshold signing contribute the greatest latency. As a result, Steward has a steeper slope than its equivalent composition, Paxos/BFT. Here also, we can see the benefit of Steward, but the performance difference is considerably smaller than in the unoptimized protocols. Paxos contributes very little latency and therefore, Paxos/Paxos's performance slightly exceeds Steward's. Note that Paxos/Paxos benefits slightly more than Steward from the optimizations. This is due to the fact that Paxos/Paxos locally orders more messages than Steward (which orders the update locally only once).

Figure 8 shows the average update latency in the same experiment, measured by the client. Although aggregation is commonly associated with an increase in latency, the optimized protocols have similar or lower latency compared

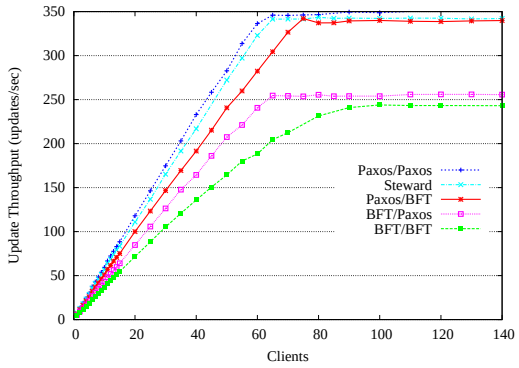

Figure 7: Throughput of Optimized Protocols, 50 ms Diameter

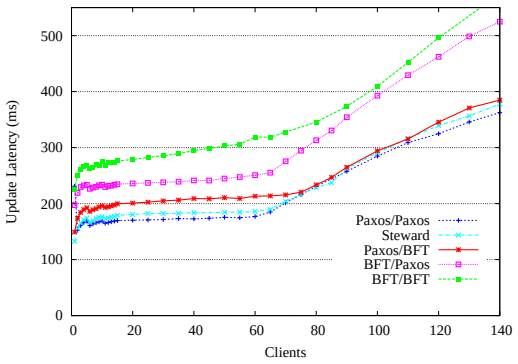

Figure 8: Latency of Optimized Protocols, 50 ms Diameter

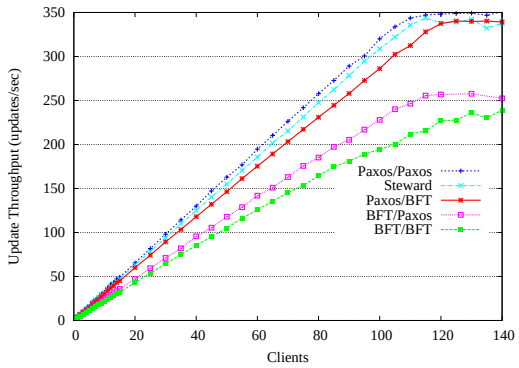

Figure 9: Throughput of Optimized Protocols, 100 ms Diameter

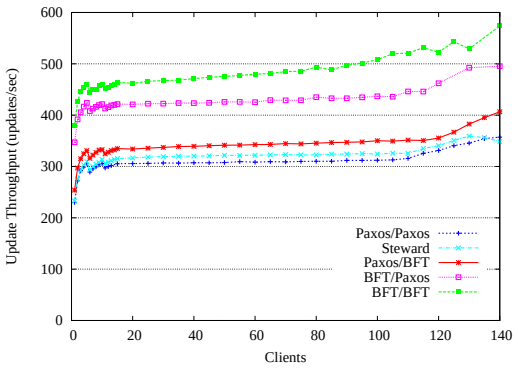

Figure 10: Latency of Optimized Protocols, 100 ms Diameter

to the unoptimized variants. A logical machine must locally order at least two external messages to execute a client's update. Therefore, even with a single client in the system, if the external accept messages arrive at about the same time, the latency can be lower with aggregation. When there are many clients, the average latency of the optimized protocols is considerably less than the latency of the unoptimized protocols, because the optimized protocols have much higher maximum throughput. Figures 9 and 10 validate the same trends on a 100 ms diameter network.

Discussion: The optimized composable architecture achieves practical performance, with throughputs of hundreds of updates per second, even while offering the strong security guarantees of BFT/BFT. This presents a factor of 4 improvement compared with the previous state of the art for wide-area Byzantine replication (i.e., unoptomized Steward) . The performance of the unoptimized protocols is computationally limited and reflects the cost associated with achieving composability and flexibility.

While the composable architecture natively suffers from a performance reduction compared with an architecture such as Steward that makes protocol-aware global optimizations, the optimized version achieves roughly 85 percent of Steward's throughput. One can build a protocol-aware globally optimized solution for each of the other three compositions to obtain faster performance, trading protocol complexity and implementation effort. While this performance improvement may be necessary in some deployments, we believe the performance achieved by the optimized composable architecture is likely to be sufficient in most cases.

7 Discussion

Using Other Replication Protocols: Our composable architecture can be extended by adding new replication protocols for use on the wide area or local area besides Paxos and BFT. Several existing protocols provide desirable properties and are promising candidates for use within our system. As demonstrated in Section 6, wide-area protocol rounds are very costly due both to increased latency and increased message complexity. Therefore, if a system requires Byzantine fault tolerance and high performance and can tolerate reduced availability, Martin and Alvisi's two-round Byzantine fault-tolerant replication protocol [22] is well-suited for use as our wide-area protocol. We believe that a composition that used this protocol would approach the performance of a composition that used Paxos on the wide area, because both are two-round protocols. The work of Yin et al. on privacy firewalls [33] can also be used effectively within a site, as part of our local-area protocol. Verrisimo's work on hybrid architectures [6, 32] is another excellent candidate for use within our architecture. Special

trusted hardware that provides stronger guarantees within a site can be used to strengthen the fault tolerance of our logical machines.

Finally, we note that the consistency guarantees of our wide-area protocol can be relaxed for use with systems that do not require state machine replication semantics. Thus, a composition could use a state machine replication protocol as the local-area protocol and an anti-entropy protocol [13] on the wide area.

Safety and Liveness Proof Sketch: Our composable architecture offers flexibility by separating the wide-area protocol, run among the logical machines, from the local-area protocol, run within the logical machine. As a direct consequence, it is possible to use a variety of replication protocols at each level of the hierarchy. Our architecture uses Paxos and BFT, both of which guarantee safety and liveness under certain synchrony assumptions. Therefore, we can directly use the known properties of these protocols when proving safety and liveness of our hierarchical architecture.

Paxos and BFT do not rely on synchrony assumptions for safety. As a result, the safety of a protocol composition follows directly from the safety of these two protocols. The local state machine replication protocol used in the ordering component ensures that all replicas in a logical machine transition through the same states and invoke the signing component on identical outgoing messages. When running BFT in the logical machine, we use threshold cryptography so that malicious servers cannot generate messages that are signed by the logical machine. Thus the logical machine will not exhibit two-faced behavior assuming that it contains at most f malicious servers. In a system where BFT is run on the wide area, the wide area protocol guarantees safety if at most F sites are compromised.

We now show that protocol compositions using Paxos and BFT are live if the system is stable, as described in Section 3. Paxos and BFT guarantee liveness when the message delay does not grow faster than the timeout used to detect a failed leader. In a flat system, the message delay is dominated by wide-area network latency. In our system, message delay is a function of both network latency and the delay associated with local ordering of wide-area protocol events. The message delay can be expressed as: $\Delta_{mess} = (2f + 2)(L + (f + 2)T_{local})$, where f denotes the number of faults tolerated by a site, L denotes latency due to network round trip and acknowledgment timer granularity, and T_{local} denotes the timeout used to detect a failed local leader. The $(2f + 2)$ term reflects the potential need to rotate through $2f$ link peer pairs before reaching a correct pair. The remaining term is the link protocol timeout, which is dependent on L and T_{local}. T_{local} is multiplied by $(f+2)$ because there must be enough time to rotate through f malicious local leaders at the receiving site.

When the system is stable, all of the terms in the equation are constant except for T_{local}. Thus, if T_{local} does not grow faster than the timeout used to detect a failed leader site in the wide-area protocol, then the composable system is live. In a stable system, either progress occurs or wide-area protocol leaders are elected continuously and the wide-area timeout continues to increase. The timeouts are calculated according to the following equations: $T_{local} = \gamma^{\alpha V_{local}}$ where α is any positive constant, V_{local} is the local-area protocol view number, and γ is any constant greater than 1; $T_{global} = \gamma^{\beta V_{global}}$ where T_{global} is the wide-area protocol timeout, β is any positive constant, and V_{global} is the wide-area protocol view number. If $\alpha < \beta$, then the timeouts will grow as required for system liveness.

8 Conclusions

This paper presented a composable, hierarchical replication architecture, tailored to systems that span multiple wide-area sites, that enables free substitution of the fault tolerance method used in each position of the hierarchy. The approach enables an administrator to customize his system, deploying either a Byzantine or a benign fault-tolerant protocol at each site and in each level of the hierarchy. The paper also presented a new link protocol that provided efficient communication between logical machines, facilitating clean separation between the local and wide-area protocols. The paper presented two optimizations that resulted in maximum wide-area Byzantine replication throughput at least four times higher than the previous state of the art.

References

[1] http://www.ciphertrust.com/resources/statistics/zombie.php.

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

[3] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling byzantine fault-tolerant replication to wide area networks. Technical Report CNDS-2006-2, Johns Hopkins University, www.dsn.jhu.edu, November 2006.

[4] Y. Amir, C. Danilov, J. Kirsch, J. Lane, D. Dolev, C. Nita-Rotaru, J. Olsen, and D. Zage. Scaling byzantine fault-tolerant replication to wide area networks. In *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pages 105–114, Washington, DC, USA, 2006. IEEE Computer Society.

[5] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[6] 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.

[7] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. Hq replication: A hybrid quorum protocol for

byzantine fault tolerance. In *Proceedings of the Seventh Symposium on Operating Systems, Washington*, Nov. 2006.

[8] 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.

[9] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal of Computing*, 12(4):656–666, 1983.

[10] K. Eswaran, J. Gray, R. Lorie, and I. Taiger. The notions of consistency and predicate locks in a database system. *Communication of the ACM*, 19(11):624–633, 1976.

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

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[13] R. A. Golding and K. Taylor. Group membership in the epidemic style. Technical Report UCSC-CRL-92-13, University of California, Santa Cruz, CA, Mar. 1992. (paper copy $4.00) (available electronically as ucsc-crl-92-13.ps.Z).

[14] 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*, volume 3, pages 317–326, Kona, Hawaii, January 1998.

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

[16] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[17] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[18] D. Malkhi and M. Reiter. Byzantine quorum systems. *Journal of Distributed Computing*, 11(4):203–213, 1998.

[19] D. Malkhi and M. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):187–202, 2000.

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

[21] 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*, page 51, Washington, DC, USA, 1998. IEEE Computer Society.

[22] J.-P. Martin and L. Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Secur. Comput.*, 3(3):202–215, 2006.

[23] R. C. Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, 1979.

[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] 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*, pages 99–110, London, UK, 1995. Springer-Verlag.

[26] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, Feb. 1978.

[27] R. Rodrigues, M. Castro, and B. Liskov. Base: using abstraction to improve fault tolerance. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 15–28, New York, NY, USA, 2001. ACM Press.

[28] R. D. Schlichting and F. B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *Computer Systems*, 1(3):222–238, 1983.

[29] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[30] V. Shoup. Practical threshold signatures. *Lecture Notes in Computer Science*, 1807:207–223, 2000.

[31] D. Skeen. A quorum-based commit protocol. In *6th Berkeley Workshop on Distributed Data Management and Computer Networks*, pages 69–80, 1982.

[32] P. Verissimo. Uncertainty and predictability: Can they be reconciled. In *Future Directions in Distributed Computing*, number 2584 in LNCS. Springer-Verlag, 2003.

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