Customizable Fault Tolerance for Wide-Area 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-2007-1 - May 2007
http://www.dsn.jhu.edu

Abstract

Constructing logical machines out of collections of physical machines is a well-known technique for improving the robustness and fault tolerance of distributed systems. We present a new, scalable replication architecture, built upon logical machines specifically designed to perform well in wide-area systems spanning multiple sites. The physical machines in each site implement a logical machine by running a local state machine replication protocol, and a wide-area replication protocol runs among the logical machines. Implementing logical machines via the state machine approach affords free substitution of the fault tolerance method used in each site and in the wide-area replication protocol, allowing one to balance performance and fault tolerance based on perceived risk.

We present a new Byzantine fault-tolerant protocol that establishes a reliable virtual communication link between logical machines. Our communication protocol is efficient (a necessity in wide-area environments), avoiding the need for redundant message sending during normal-case operation and allowing a logical machine to consume approximately the same wide-area bandwidth as a single physical machine. This dramatically improves the wide-area performance of our system compared to existing logical machine based approaches. We implemented a prototype system and compare its performance and fault tolerance to existing solutions.

1 Introduction

As network environments become increasingly hostile, even well-protected distributed information systems, constructed with security in mind, are likely to be compromised [1]. Byzantine fault-tolerant replication (e.g.,[4, 6, 23, 30]) can be used to construct survivable information systems that withstand partial compromises. Such systems are typically deployed in several local-area sites distributed across a wide-area network. Practical solutions should have two fundamental characteristics. First, they must achieve high performance in large-scale deployments, which requires the efficient use of limited wide-area inter-site bandwidth. Second, they must offer customizability, because heterogeneous sites have different risk profiles resulting from varied physical security, hardware, and performance requirements. To the best of our knowledge, no previous replication architecture simultaneously provides these two properties.

This paper presents the first scalable wide-area replication system that (1) achieves high performance through the efficient use of wide-area bandwidth and (2) allows customization of the fault tolerance approach used within and among the local-area sites. Our architecture uses the state machine (SM) approach [17, 32] to transform the physical machines in each site into a *logical machine* (LM), and the logical machines run a wide-area protocol. Using the state machine approach to build logical machines is a well-known technique for cleanly separating the protocol used

*This work was partially supported by NSF grant 0430271.

to implement the logical machine from the protocol running on top of it. Representative systems include Voltan [5], Immune [26], BASE [30], Starfish [15], and Thema [24], which are described in more detail in Section 2. The state machine approach affords free substitution of the fault tolerance method used in each site and in the wide-area replication protocol, allowing a Byzantine or benign fault-tolerant protocol to be selected depending on system requirements and perceived risks.

All previous Byzantine fault-tolerant SM-based logical machine abstractions send messages redundantly in order to guarantee reliable communication in the presence of malicious protocol participants. Typically, to prevent malicious servers from blocking the message transmission, at least $f + 1$ servers in the sending LM will each send to $f + 1$ servers in the receiving LM, where f is the number of potential faults in each LM.[1] While this strategy works well on local-area networks, where bandwidth is plentiful, it is impractical for replication systems that must send many messages over wide-area links. In our experience, it is wide-area bandwidth and not computational constraints that limits the performance of well-engineered wide-area replication systems. To address this weakness, we present BLink, the first Byzantine fault-tolerant communication protocol that guarantees efficient wide-area communication between logical machines. BLink is specifically designed for use in systems where (1) the physical machines comprising an LM are located in a LAN that provides low-latency, high-bandwidth communication, and (2) the LMs are located in different LANs, and are connected by high-latency, low-bandwidth links. BLink usually requires only one physical message to be sent over the wide-area network for each message sent by the logical machine.

Our previous wide-area replication architecture, Steward [4], shares some similarities with our new architecture. Both systems use a hierarchical logical machine architecture and provide high performance by efficiently utilizing wide-area bandwidth. However, they use fundamentally different techniques to construct their logical machines. The servers comprising each LM in our new architecture totally order *all* events that cause a state transition in the protocol running on top of them (i.e., updates, acknowledgements, and wide-area timeouts), and execute these events in the same order. This is in striking contrast to the approach taken in Steward, where the wide-area protocol makes state transitions based on unordered events. As a result, in Steward, the protocols running within the sites and those running among the sites are interdependent and cannot be separated. Consequently, the fault tolerance approach within and among the sites cannot be customized. Since Steward runs a benign fault-tolerant wide-area protocol, it cannot survive a site compromise. We describe precisely why the Steward architecture is inflexible and inherently a poor match for diverse wide-area environments requiring customizability in Section 2.

To mitigate the high cost of the additional ordering required by the state machine approach, we use two optimizations. First, we amortize the computational costs associated with digital signatures within the LM ordering protocol using known aggregation techniques. Second, we demonstrate the first use of a Merkle tree [25] mechanism to amortize the cost of threshold signatures while producing a self-contained, threshold-signed wide-area message. Amortizing optimizations enable an LM to process and send on the order of a thousand wide-area messages per second, preventing LM throughput from limiting overall performance. State machine based LMs augmented with BLink and the Merkle tree optimization have precisely the necessary properties to build a customizable fault-tolerant replication system without sacrificing performance.

The contributions of this work are:

1. It presents a new hierarchical replication architecture for wide-area networks that combines high performance and customizability of the fault tolerance approach used within each site and among the sites. Using a Byzantine fault-tolerant protocol on the wide area protects against site compromises and offers fundamentally stronger security guarantees than our previous system.

2. It presents a new Byzantine fault-tolerant protocol, BLink, that guarantees 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. The use of BLink increases performance by over an order of magnitude in comparison to an SM-based logical machine approach that uses previous communication protocols, which require at least $2f + 1$, and typically $(f + 1)^2$, redundant sends.

[1] It may be possible to use a peer-based protocol in which each of $2f + 1$ servers sends to a unique peer. To the best of our knowledge, no existing system uses this method, except for Steward [4], which uses it sparingly to send global view change messages.

3. It shows that by using optimizations that amortize the computational cost of the logical machine ordering, the new system achieves high performance, outperforming the Steward system by 4 times.

We compare 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 4 fold better than the original Steward architecture, which was the previous state of the art. Our new architecture achieves 12 percent lower performance than a new version of Steward that we developed for comparison that uses similar amortizing optimizations. This performance difference is the cost of providing clean separation and customizability. A Byzantine over Byzantine composition, which is not possible in Steward and provides fundamentally stronger fault tolerance, performs 3 times better than the original Steward, and within 35 percent of Steward with amortizing optimizations.

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 the BLink protocol, and Section 6 describes our performance optimizations. In Section 7, we evaluate the performance of our architecture. In Section 8 we provide a proof sketch for safety and liveness. Section 9 provides discussion, and Section 10 concludes the paper.

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 [17] and Schneider [32] 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.

The state machine approach has been used in many systems to construct fault-tolerant logical machines out of collections of physical machines. Schlichting and Schneider [31] discuss 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. The Voltan system of Brasileiro, et al. [5] uses the state machine approach to construct two-processor fail-silent nodes that either work correctly or become silent if an internal failure of one of the processes is detected. The FTS system of Friedman and Hadad [12] uses active replication to construct a lightweight fault tolerance service for CORBA. The Immune system of Narasimhan et al. [26] replicates objects in CORBA applications, allowing the applications to continue operating despite Byzantine behavior. The Starfish system of Kihlstrom and Narasimhan [15] builds an intrusion-tolerant middleware service by using a hierarchical membership structure and end-to-end intrusion detection. Both Immune and Starfish use an underlying group communication system, such as SecureRing [14]. The Thema system of Merideth, et al. [24] uses state machine replication to build Byzantine fault-tolerant Web Services.

Paxos and BFT: Paxos [16, 18] 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 (enabling state machine replication). 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 [6] also totally orders client requests, 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 [30] 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 hierarchical state machine replication architecture for wide-area networks. 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. The servers within a site pass incoming wide-area messages directly to the upper-level wide-area protocol, *without ordering them within the site*. For most messages, this eliminates the overhead associated with Byzantine fault-tolerant agreement (Byzantine agreement is used only to assign a sequence number to client updates). The price of this optimization is the need for customized protocols specifically designed to overcome the temporary state divergence with respect to the lower-level protocols. Steward has over ten specialized protocols that run within and among the sites. Most of these protocols are associated with global view changes, during which a new leader site is elected. Since the servers comprising a Steward LM do not proceed through the same sequence of states, they must run special protocols to agree on the content of outgoing wide-area messages. For example, when a site needs to send a summary of its knowledge, it runs the CONSTRUCT-GLOBAL-CONSTRAINT protocol so that (1) the servers can agree on a common state and (2) they can invoke the THRESHOLD-SIGN protocol on the same message. Other wide-area messages require separate protocols. Note that the servers do not exhibit state divergence with respect to the global state machine replication service. Steward can withstand f out of $3f + 1$ Byzantine failures within each site but cannot survive even a single site compromise.

Other Byzantine Fault-tolerant Protocols: Yin et al. [35] 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 [23] recently introduced a two-round Byzantine consensus algorithm, which uses $5f + 1$ servers to overcome f faults.

Quorum systems obtain Byzantine fault tolerance by applying quorum replication methods [19]. Examples of such systems include Phalanx [22], and Fleet [20, 21]. The HQ protocol [9] combines the use of quorum replication with Byzantine fault-tolerant agreement, using a more lightweight quorum-based protocol during normal-case operation and BFT to resolve contention when it arises. Also related to our work are group communication systems resilient to Byzantine failures [10, 14, 27, 28]. Verissimo et al. propose a hybrid approach [8, 34], where synchronous, trusted nodes provide strong global timing guarantees.

3 System Model and Service Guarantees

Servers are organized into wide-area *sites*; each site has 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. We can use a state transfer mechanism (as in [7]) or an update reconciliation mechanism (as in [3]) to reconcile states after a remerge. The number of servers within each site varies with the desired level of fault tolerance within the site. If a benign fault-tolerant protocol 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 is deployed, then we assume there are at least $3f + 1$ servers within the site, where at most f servers may be Byzantine.

The free substitution property afforded by using SM-based logical machines allows our architecture to support a rich configuration space. Each site can employ either a Byzantine or a benign fault-tolerant SM replication protocol to implement its LM. Further, by running a Byzantine fault-tolerant wide-area protocol among LMs, our system can guarantee consistency even when the fault assumptions made within some of the sites are violated. 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 introduce updates into the system by communicating with the servers in their local site. 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. 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. Clients propose updates sequentially:

a client, c, may propose an update with sequence number $i_c + 1$ only after it receives a reply for an update with sequence number i_c. 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. 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 [33]. 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.

Our system achieves replication via the state machine approach, establishing a global, total order on client updates in the wide-area protocol. Each server executes an update with global 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.

Our replication system provides 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 tolerating even one failure can always be both safe and live [11], we provide liveness under certain synchrony conditions. We define the following terminology and then specify our liveness guarantee:

- *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.
- 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 be Byzantine) 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*.

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

In our composable architecture, the physical machines in each site implement a *logical machine* by running a local state machine replication protocol [17, 32]. We then run a state machine replication protocol on top of these logical machines, among the sites. Using SM-based logical machines is an established technique for cleanly separating the implementation of the LM from the protocol running on top of it. Our architecture leverages the flexibility afforded by this technique, allowing one to customize the protocol and type of fault tolerance desired, both within each LM and among the LMs. Further, we can use the known safety proof for the wide-area protocol (when run among single machines), together with one for the local state machine replication protocol, to trivially prove safety for the composition. The liveness proof is more complicated, but much simpler than what is necessary when the wide-area and local-area protocols are interdependent. See Section 8 for a more formal discussion of the safety and liveness

properties. In the remainder of this section, we first review how we use the state machine approach to build our logical machines, and then present several compositions of our architecture.

Implementing Logical Machines: The wide-area replication protocol running on top of our LMs runs just as it would if it were run among a group of single machines, each located in its own site. Each LM sends the same types of wide-area messages and makes the same state transitions as would a single machine running the wide-area replication protocol. To support this abstraction, the physical machines in each site use an agreement protocol to totally order all events (messages and timeouts) that cause state transitions in the wide-area protocol. The physical machines then execute the events in the agreed upon order. Thus, the LM conceptually executes a single stream of wide-area protocol events. The LMs communicate using BLink to avoid sending redundant wide-area messages.

The SM approach assumes that all events are deterministic. As a result, we must prevent the physical machines from diverging in response to non-deterministic events. For example, although the physical machines within a site may fire a local timeout asynchronously, they must not act on the timeout until its order is agreed upon. We use a technique similar to BASE [30] to handle non-deterministic events. To implement an LM timeout when a Byzantine fault-tolerant agreement protocol is used, each server in the site sets a local timer, and when this timer expires, it sends a signed message to the leader of the agreement protocol. 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).

Outgoing wide-area messages carry an RSA signature [29]. When a logical machine is implemented with a benign fault-tolerant protocol, the message carries a standard RSA signature. When running a Byzantine fault-tolerant local protocol, the physical machines within the site generate 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 (threshold) signature, saving wide-area bandwidth. Our 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.

Protocol Compositions: The free substitution property of our architecture makes it extensible, allowing one to use any of several existing state of the art replication protocols, both within each site and on the wide area. In this paper, we focus on four compositions of our architecture, using two well-known, flat replication protocols: Paxos [16, 18] as our benign fault-tolerant protocol, and BFT [6] as our Byzantine fault-tolerant replication protocol. When BFT is used within a site to implement an LM, the LM 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 LM 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. 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.

We conclude by providing an example of Paxos/BFT that traces the flow of a client update through the system during normal-case operation. First, a client sends an update to a server in its own site, which forwards the update to the leader site (i.e., the site coordinating the Paxos wide-area protocol). The leader site LM uses BFT (requiring three local communication rounds), to locally order the message event corresponding to the reception of the update by the LM. The LM generates a wide-area proposal message, binding a global sequence number to the update. The message is then threshold signed via a one-round protocol. The threshold-signed proposal is then sent (using BLink) to the other sites. Each non-leader LM orders the incoming proposal, generates an acknowledgement (accept) message for the proposal, and then sends the acknowledgement (using BLink) to the other LMs. Each LM then orders the reception of the accept message. When the proposal and a majority of accepts are collected, the LM globally orders the client update, completing the protocol. We observe that the protocol consists of many rounds, most of which are associated with ordering incoming messages; this is the price to achieve protocol separation.

5 BLink: Byzantine-resilient Communication

To achieve high performance over the low-bandwidth links characteristic of wide-area networks, our architecture requires an efficient mechanism for passing messages between logical machines. As described in Section 4, each LM is implemented by a replicated group of physical machines, some of which may be faulty. Faulty servers may

fail to send, receive, and/or disseminate wide-area messages. Existing protocols that use state machine based logical machines (e.g., [5, 24, 26]) overcome this problem by redundantly sending all messages between logical machines. For example, in a system tolerating f faults, each of $f + 1$ servers in the sending LM might send the outgoing message to $f + 1$ servers in the receiving LM. While this overhead may be acceptable in high-bandwidth LANs or systems supporting a small number of faults, the approach (or even one with $O(f)$ overhead) is poorly suited to large-scale wide-area deployments.

Steward [4] avoids sending redundant messages during normal-case operation by choosing one server (the site representative) to send outgoing messages. Steward employs a coarse-grained mechanism to monitor the performance of the representative, using a lack of global progress to signal that the representative *may* be acting faulty and should be replaced. This approach has two undesired consequences: timeouts for detecting faulty behavior can be significantly higher than they need to be, and the communication protocol is (1) not generic and (2) tightly coupled with global and local protocols, making it unusable in our customizable architecture.

In this section we present the *Byzantine Link* protocol (BLink), a new Byzantine fault-tolerant protocol that allows logical machines to efficiently communicate with each other over the wide-area network, regardless of the protocols they are running.[2] BLink consists of four sub-protocols, each tailored to the fault tolerance method employed in the sending and receiving LMs: (benign, benign), (Byzantine, benign), (benign, Byzantine), and (Byzantine, Byzantine). We first focus on the most challenging case, where each LM runs a Byzantine fault-tolerant protocol. We then describe the other sub-protocols.

BLink establishes a reliable communication link between two LMs using three 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 LM to monitor the behavior of the link and take action if it appears to be faulty. The third technique ensures fairness by preventing the adversary from starving any particular link.

5.1 Delegating Communication Responsibility

BLink constructs a set of *logical links* from each LM to its neighboring LMs. These logical links are reliable, masking faulty behavior at both the sending and receiving LMs. To support this abstraction, BLink defines a set of *virtual links*, each consisting of one server (the *forwarder*) from the sending LM and one server (the *peer*) from the receiving LM. The servers on a virtual link form a (forwarder, peer) pair. The forwarder sends outgoing wide-area messages to the peer, and the peer disseminates incoming messages to the other servers in the receiving LM.

For each outgoing logical link, the sending LM delegates communication responsibility to the forwarder of one of its virtual links. This decision is made independently for each outgoing logical link; different servers may act as forwarder on different logial links, and the same server may act as forwarder on multiple logical links. Since either the forwarder or the peer may be faulty, the other servers within the sending LM monitor the performance of the virtual link and move to the next virtual link (electing the next forwarder) if the current forwarder is not performing well enough on the given link (we define this notion more precisely below).

The virtual links are constructed as follows, for two logical machines LM_A and LM_B. Suppose LM_A has $3F_A+1$ servers, and LM_B has $3F_B + 1$ servers, with $F_A \geq F_B$. We construct $v = LCM(3F_A + 1, 3F_B + 1)$ virtual links, labeled 0 through $v - 1$. Virtual link i consists of the server in LM_A with server id $i \bmod (3F_A + 1)$ and the server in LM_B with server id $i \bmod (3F_B + 1)$. The LM moves through the virtual links sequentially, wrapping around modulo v. We use the least common multiple of $3F_A + 1$ and $3F_B + 1$ so that at each site each server is used in the same number of virtual links. This prevents the adversary from "overbenefiting" in its fault allocation by making the faulty servers participate in more virtual links than the correct servers. The BLink logical link is shown in Figure 1.

When $F_A = F_B$, it is easy to see (by an extension of the pigeonhole principle) that, out of $3F_A + 1$ virtual links, our matching guarantees the existence of at least $F_A + 1$ correct virtual links, where both the forwarder and the peer are correct (see Figure 2). When $F_A > F_B$, the following proof sketch shows that at least $1/3$ of the virtual links

[2]The term "link" refers to the logical communication link established between LMs. In particular, BLink operates over UDP.

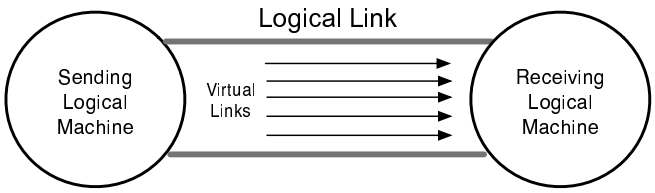

Figure 1: A logical link is constructed from LCM($3F_A+1, 3F_B+1$) virtual links. Each virtual link consists of a forwarder and a peer. At any time, one virtual link is used to send messages on the logical link. A virtual link that is diagnosed as potentially faulty is replaced.

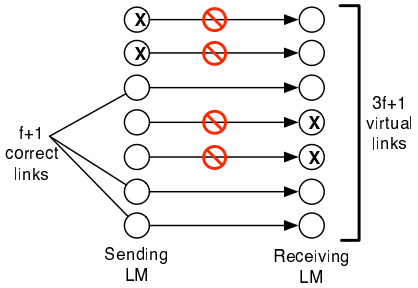

Figure 2: When $F_A = F_B$, any set of $3F_A + 1$ virtual links has at least $F_A + 1$ virtual links with both forwarder and peer correct.

are correct. Each server in LM_A is a member of $v/(3F_A + 1)$ virtual links, and each server in LM_B is a member of $v/(3F_B + 1)$ virtual links. Let b equal the number of faulty links. Then we have:

$$\begin{aligned} b &\leq F_A\tfrac{v}{3F_A+1} + F_B\tfrac{v}{3F_B+1} \\ &= (\tfrac{F_A}{3F_A+1} + \tfrac{F_B}{3F_B+1})v \\ &< (\tfrac{F_A}{3F_A} + \tfrac{F_B}{3F_B})v \\ &= \tfrac{2}{3}v \end{aligned}$$

Thus, at least $1/3$ of the virtual links are correct.

In addition to the ratio of correct virtual links, we also consider the worst-case number of virtual links through which the sending LM must cycle before reaching a correct link. We refer to this value as C_{max}. We first present the equation for C_{max} and then explain how the equation is derived:

$$\begin{aligned} G &= \lfloor \frac{F_A}{2F_B+1} \rfloor \\ L &= F_B + F_A \bmod (2F_B+1) \\ C_{max} &= G*(3F_B+1) + L + 1 \end{aligned} \tag{1}$$

Intuitively, in each group of $3F_B + 1$ links, the adversary can use F_B servers from LM_B and must consume at least $2F_B + 1$ servers from LM_A (otherwise we will have reached a correct link). Thus, LM_A may need to cycle through $G = \lfloor \frac{F_A}{2F_B+1} \rfloor$ complete groups of $3F_B + 1$ virtual links. The last group of $3F_B + 1$ (called L in the formula above) can contain F_B servers from LM_B, plus the remaining faulty servers, if any, from LM_A.

We now show the following bound on C_{max}:

$$C_{max} \leq 1.5F_A - 0.5(F_A \bmod (2F_B+1)) + F_B + 1 \tag{2}$$

We begin by removing the floor from the first term in Equation 1:

$$\begin{aligned} C_{max} &\leq \lfloor \frac{F_A}{2F_B+1} \rfloor (3F_B+1) + F_B + F_A \bmod (2F_B+1) + 1 \\ &= \left(\frac{F_A}{2F_B+1} - \frac{F_A \bmod (2F_B+1)}{2F_B+1} \right) (3F_B+1) + F_B + F_A \bmod (2F_B+1) + 1 \\ &= \frac{F_A}{2F_B+1}(3F_B+1) - \frac{F_A \bmod (2F_B+1)}{2F_B+1}(3F_B+1) + F_B + F_A \bmod (2F_B+1) + 1 \end{aligned} \tag{3}$$

We now rewrite the bound in Equation 2 in a way that will help to elucidate how the bound is derived:

$$C_{max} \leq 1.5F_A - 1.5(F_A \bmod (2F_B+1)) + F_B + F_A \bmod (2F_B+1) + 1 \tag{4}$$

Comparing Equation 3 and Equation 4, we see that the last three terms appear in both equations. Thus, we can ignore them for the time being and focus on proving the following intermediate result:

$$\frac{F_A}{2F_B+1}(3F_B+1) - \frac{F_A \bmod (2F_B+1)}{2F_B+1}(3F_B+1) \quad \leq \quad 1.5F_A - 1.5(F_A \bmod (2F_B+1)) \tag{5}$$

Rewriting the left side of Equation 5, we wish to show:

$$(3F_B+1)\frac{F_A - F_A \bmod (2F_B+1)}{2F_B+1} \leq 1.5F_A - 1.5(F_A \bmod (2F_B+1)) \tag{6}$$

We now consider the three possible cases for the relationship between F_A and $2F_B+1$. First, if $F_A < 2F_B+1$, then modular arithmetic implies that $F_A \bmod (2F_B+1) = F_A$. Thus, in this case, Equation 6 is an equality. Substituting, we obtain:

$$\begin{aligned}(3F_B+1)\frac{F_A - F_A \bmod (2F_B+1)}{2F_B+1} &= (3F_B+1)\frac{F_A-F_A}{2F_B+1} \\ &= 0 \\ &= 1.5F_A - 1.5F_A \\ &= 1.5F_A - 1.5(F_A \bmod (2F_B+1))\end{aligned}$$

We now consider the second case. If F_A is a multiple of $2F_B+1$, then $F_A \bmod (2F_B+1) = 0$. In this case, substitution reveals that the left side of Equation 6 is strictly less than the right side:

$$\begin{aligned}(3F_B+1)\frac{F_A - F_A \bmod (2F_B+1)}{2F_B+1} &= (3F_B+1)\frac{F_A-0}{2F_B+1} \\ &= \frac{3F_B+1}{2F_B+1}F_A \\ &< 1.5F_A \\ &= 1.5F_A - 1.5*0 \\ &= 1.5F_A - 1.5(F_A \bmod (2F_B+1))\end{aligned}$$

The third step holds because $\frac{3F_B+1}{2F_B+1}$ is bounded above by 1.5.

Finally, we consider the third case. If $F_A > 2F_B+1$ but not a multiple of $2F_B+1$, then $F_A \bmod (2F_B+1)$ produces a value in the open interval $(0, 2F_B+1)$, which must be less than F_A. In this case, the left side of Equation 6 is strictly less than the right side:

$$\begin{aligned}(3F_B+1)\frac{F_A - F_A \bmod (2F_B+1)}{2F_B+1} &= \frac{3F_B+1}{2F_B+1}(F_A - F_A \bmod (2F_B+1)) \\ &< 1.5(F_A - F_A \bmod (2F_B+1))\end{aligned}$$

The last step holds because $\frac{3F_B+1}{2F_B+1} < 1.5$ and $(F_A - F_A \bmod (2F_B+1))$ is positive.

Thus, we have shown that Equation 6 (and, therefore, Equation 5) holds in all cases. Since all other terms are identical in Equations 3 and 4, we have shown the following:

$$\begin{aligned}C_{max} &\leq \frac{F_A}{2F_B+1}(3F_B+1) - \frac{F_A \bmod (2F_B+1)}{2F_B+1}(3F_B+1) + F_B + F_A \bmod (2F_B+1) + 1 \\ &\leq 1.5F_A - 1.5(F_A \bmod (2F_B+1)) + F_B + F_A \bmod (2F_B+1) + 1\end{aligned}$$

Combining terms, we obtain the following, which matches Equation 2 and completes the proof:

$$C_{max} \quad \leq \quad 1.5F_A - 0.5(F_A \bmod (2F_B+1)) + F_B + 1$$

We now provide the reader with an intuitive feel for the relationship between C_{max} (Equation 1) and its bound (Equation 2) by summarizing actual values for C_{max} over a wide range of possible configurations. For configurations in which F_A and F_B can vary from 1 to 1000, with $F_A \geq F_B$, the minimum, maximum, and average values for C_{max} are $(1.334F_A+1)$, $(2.0F_A+1)$, and $(1.704F_A+1)$, respectively.

5.2 Reliability and Monitoring

BLink 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 events totally-ordered by the LM and can be sequenced using this total order. Each LM periodically generates a threshold-signed acknowledgement message, which contains, for each logical link, the sequence number through which the LM has received all previous messages. The generation of the acknowledgement is triggered by executing an LM timeout, as described in Section 4. Servers could also piggy-back acknowledgements on regular outgoing messages for more timely, fine-grained feedback. The peer server for each incoming logical link sends the acknowledgement to its corresponding forwarder, which presents the acknowledgement to the servers in the sending LM.

The acknowledgement serves two purposes. First, it is used to determine which messages need to be retransmitted over the link 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 LM use the acknowledgement to evaluate the performance of the current forwarder. Each server in the sending LM maintains a queue of the unacknowledged messages on each logical link, placing an LM 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 LM, the timeout is canceled and a new timeout is set on the next message in the queue. However, if the timeout expires before such an acknowledgement is received, the servers in the sending LM suspect that the virtual link is faulty and elect the next forwarder. Note that this mechanism can be augmented to enforce a higher throughput of acknowledged messages by placing a timeout on a batch of messages. Of course, BLink does not guarantee delivery when a site at one or both ends of the logical link is Byzantine.

5.3 Fairness

The third technique used by the BLink protocol addresses the dependency between the evaluation of the virtual link forwarder and the performance of the leader of the agreement protocol in the receiving LM. Intuitively, if the leader in the receiving LM could selectively refuse to order certain messages or could delay them too long, then a correct forwarder (in the sending LM) 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 virtual link to the extent possible, and thus we augment the agreement protocol with a fairness mechanism.

When a peer in a receiving LM receives an incoming message, it disseminates the message within the site; all servers then expect the leader of the agreement protocol to initiate the message for ordering such that it can be executed by the LM. To ensure fairness, servers must place a timeout on the leader of the agreement protocol to prevent the selective starvation of a particular incoming logical link. Servers within the LM maintain a queue for each incoming logical link. When the leader receives a message to be ordered, it places the message on the appropriate 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.

5.4 Other BLink Sub-protocols

We now consider the problem of inter-LM communication when one or both of the LMs is implemented using a benign fault-tolerant state machine replication protocol. Given that we have a solution for the (Byzantine, Byzantine) case, the simplest approach would be to modify the mapping of virtual links to fit the other three cases: (benign, benign), (benign, Byzantine), and (Byzantine, benign). The number of virtual links in the (benign, benign) case is set to LCM($2F_A + 1$, $2F_B + 1$). In the (benign, Byzantine) case, the number of virtual links is LCM($2F_A + 1$, $3F_B + 1$). In the (Byzantine, benign) case, the number of virtual links is LCM($3F_A + 1$, $2F_B + 1$).

We can use an argument similar to the one found in Section 5.1 to prove the existence of a correct virtual link in each of the cases. In the (benign, benign) case, the mapping yields at least one correct virtual link. In the (benign, Byzantine) and (Byzantine, benign) cases, the analysis shows that at least $1/6$ of the virtual links are correct.

When the sending logical machine runs a benign fault-tolerant protocol, it is possible to use a different approach to reduce the number of virtual links through which the LM must cycle before reaching a correct link. The approach assumes that the correct servers in the sending LM can communicate equally well with the correct servers in the receiving LM. This assumption implies that there is no need for the sending LM to replace a correct forwarder. The sending LM thus allows its forwarder to try different peers until it establishes a correct virtual link. The forwarder will need to cycle through at most $F_B + 1$ such peers before finding a correct one. The servers in the sending LM can use a standard ping/Hello protocol to monitor the status of the current forwarder. A server only votes to replace the forwarder if it has not received a response from the forwarder within a timeout period. Note that this technique is not applicable to the (Byzantine, benign) case, since the forwarder may be Byzantine faulty and cannot be trusted to find a correct peer. We also note that we can reduce the worst-case number of virtual links through which the logical machine must cycle via the following optimization. When a forwarder detects that a peer is faulty, it locally broadcasts a message indicating that the peer should be skipped by other forwarders. The next forwarder then picks up where the last forwarder left off. In this way, one can think of the logical machine as rotating through a single sequence of peers. Note that subsequent forwarders may eventually send to peers that were previously diagnosed as faulty, because a correct peer may be diagnosed as faulty due to a transient network partition.

5.5 Client Updates

Our architecture guarantees that if the system is stable and a client is connected to a stable site, the client will be able to order its update. Since BLink provides efficient communication between logical machines, it is technically possible to treat each client as a non-replicated logical machine and use BLink to provide Byzantine fault-tolerant communication between clients and logical machines consisting of servers. However, using BLink in this manner requires (1) extra overhead that increases normal-case latency, (2) sending threshold-signed acknowledgements from the LM to the client, and (3) a separate queue for each client. Therefore, our architecture includes a specialized protocol, CLink, which guarantees that clients will be able to efficiently and quickly inject updates into the system. CLink only guarantees that a logical machine will order the client's update. Once this occurs, the wide-area protocol running on the logical machines uses techniques similar to BFT to guarantee global ordering.

CLink consists of two components. The first component, CLink-1, allows a server that receives a new, correctly signed client update to force the update to be ordered by the server's logical machine. CLink-1 can be used by any server, regardless of whether the server is in the leader LM or one of the non-leader LMs. The second component, CLink-2, is a simple optimistic forwarding protocol that typically allows a non-leader LM server to forward a client update directly to the leader LM without requiring that the server's LM locally order the update. This mechanism adds as little latency as possible and requires no cryptographic operations at the client's LM (if the LM is a non-leader). In cases where optimistic forwarding fails because a malicious server in the forwarding path drops the message, the client retransmits its update, and its logical machine will locally order the update and then use the BLink protocol to transmit the update to the leader LM. This procedure guarantees that the client's update will be propagated to the leader LM, but requires additional latency and processing overhead.

The CLink-1 protocol ensures that a logical machine will locally order and process any client update received by one of the LM's correct servers. The leader LM uses CLink-1 on all such updates, while non-leader LMs use CLink-1 only for those updates retransmitted by a client; retransmitted updates are marked with a *retransmit* flag. We now describe the actions taken when a server r invokes CLink-1. Upon receiving an update, u, server r generates an *Ordering_Request(*id_r, u, seq_r*)* message, signs it, and sends it to the other servers in the site. seq_r is a local sequence number, generated by r, that is incremented each time r sends a new *Ordering_Request* message.

When a server receives an *Ordering_Request* from r, it stores the message and forwards it to the leader. Additionally, the server sets a timeout on the message (where the server expects the message to be ordered within the timeout period) if it has executed all *Ordering_Request* messages from server r up to and including $seq_r - 1$. The

leader attempts to order the *Ordering_Request* messages from each server in round-robin fashion; it decides whether to propose an *Ordering_Request* for a server s as follows. If the leader does not have an *Ordering_Request* from server s, then it moves to the next server modulo N. Otherwise, let *Ordering_Request OR(*id_s*,* u*,* seq_s*)* be the *Ordering_Request* message from server s with the lowest sequence number. The leader proposes OR if seq_s is one greater than the sequence number of the last *Ordering_Request* executed or proposed from s. When a server executes an *Ordering_Request(*id_s*,* u*,* seq_s*)* message, it cancels the timeout associated with the execution of that message, if one is set. If the server has the next *Ordering_Request* message from s, it sets a timeout on that message. Note that, if server s is Byzantine, some server might have received an *Ordering_Request* message from s with the same sequence number (seq_s) but a different update. In this case, the server cancels its timeout but has explicit proof that s is corrupt and can broadcast the proof to the rest of the servers. Finally, we note that CLink-1 can be optimized by including only a digest of the update in the *Ordering_Request* message, reducing the amount of bandwidth consumed when more than one server includes the same update in an *Ordering_Request*.

We conclude with a brief description of the optimistic forwarding mechanism (CLink-2), and what happens when it fails. Each client contains a randomly shuffled list of the servers in its site to which it sends updates. Similarly, each server contains a randomly shuffled list of the servers in each of the other sites. Correct clients and servers rotate through the entries in the lists when optimistic forwarding fails. When a client wants to submit an update, it sends the update to a server in its site. Then the clients sets a timeout. If the server that received the update is correct, then it forwards the update to a server in the leader LM (based on its list of servers for the current leader LM) and sets a timeout. The server in the leader LM, if correct, will generate an *Ordering_Request* message and invoke CLink-1. If the client's timeout expires, then the client generates a new signed update that is identical to the first one, except that it contains a retransmit flag. The client sends this retransmission to $f+1$ servers in its local site. Therefore, at least one correct server will invoke CLink-1, and the client's LM will locally order the retransmitted update. The LM will then use the BLink protocol to propagate the update to the leader site. The wide-area protocol is responsible for replacing a malicious leader LM, if the wide-area protocol is Byzantine fault-tolerant. The next time the client sends an update, it will attempt an optimistic send to the next server in its list. Similarly, the server that forwarded the update also uses the next server in its list if its timeout expired.

6 Performance Optimizations

Our composable architecture has significant computational overhead, because each LM 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 LM threshold signs all outgoing messages, which imposes an even greater computational cost. Consequently, we use Merkle hash trees [25] to amortize the cost of threshold signing, and we improve the performance of LM event processing via well-known aggregation techniques. 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: Instead of threshold signing every outgoing message, we generate a single threshold signature, based on a Merkle hash tree, that is used to authenticate several messages. Each outgoing message is self-contained, including everything necessary for validation (except the public key). 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 use the aggregation technique described in [7] to increase the throughput of local event processing by the LM. The LM orders several events at once, allowing the LM to order thousands of events per second over LANs while providing Byzantine fault tolerance. With this performance, it is likely that the incoming wide-area bandwidth will limit throughput.

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.

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.

7 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.

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 our composable architecture. 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, four of which use our composable architecture: Paxos/Paxos, BFT/Paxos, Paxos/BFT, BFT/BFT. The fifth is a new implementation of Steward, which includes the option of using the same optimization techniques used in our new architecture. The 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 do not significantly affect performance when our optimizations are used. We also note that BFT can use MACs, which improves its latency and results in much better performance when no aggregation is used. 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. 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

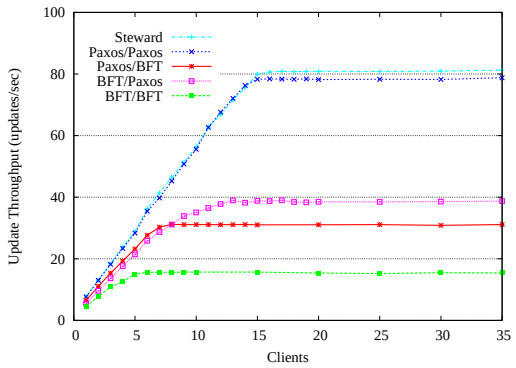

Figure 3: Throughput of Unoptimized Protocols, 50 ms Diameter

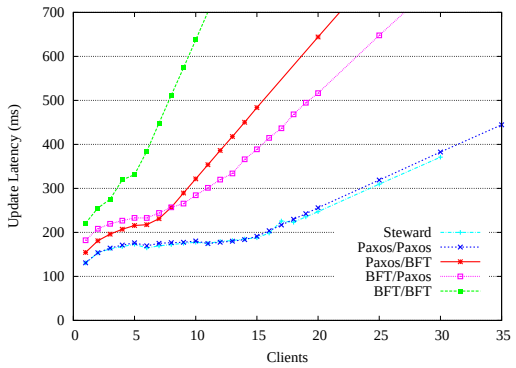

Figure 4: Latency of Unoptimized Protocols, 50 ms Diameter

when the optimizations presented in Section 6 are not used. 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.

Architectural Comparison: To evaluate the overhead of our composable architecture compared to that of Steward, we first compare the performance of the five protocols when the optimizations presented in Section 6 are not used. We wish to make clear that the unoptimized results do not reflect our architecture's actual performance. We specifically removed the optimizations to provide a clear picture of their benefits. We used a symmetric configuration where all sites are connected to each other with 50 ms (emulating crossing the continental US), 10Mbps links. Each client sends an update to a server in its site, waits for 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 protocols, 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 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, reflecting 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. The peak performance of each of the protocols is a function of the number of cryptographic operations (see 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 inject 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 6. 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.

In Figures 7 and 9 (discussed below), we include two theoretical throughput upper bounds of a Paxos/BFT composition in which LMs redundantly send physical messages over the wide area to ensure reliable inter-LM commu-

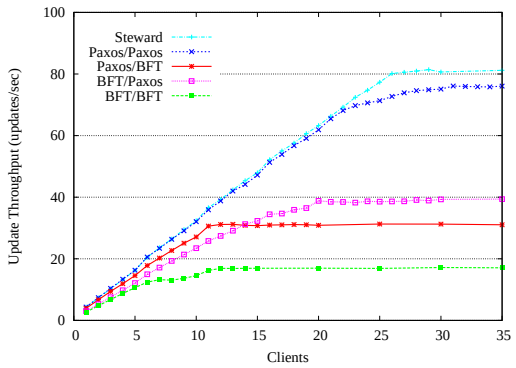

Figure 5: Throughput of Unoptimized Protocols, 100 ms Diameter

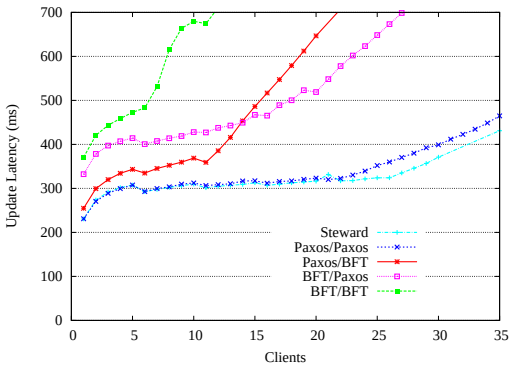

Figure 6: Latency of Unoptimized Protocols, 100 ms Diameter

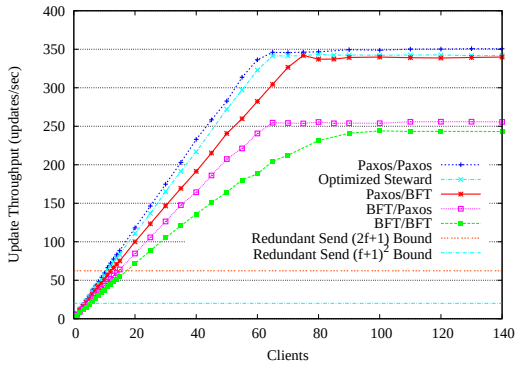

Figure 7: Throughput of Optimized Protocols, 50 ms Diameter

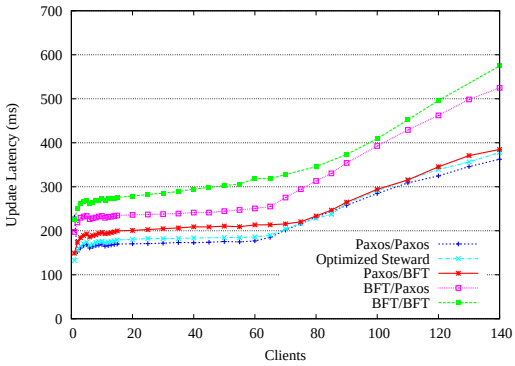

Figure 8: Latency of Optimized Protocols, 50 ms Diameter

nication. We computed the maximum throughput by assuming that the wide-area Proposal message sent from the leader site contains at least a signed update from the client and an RSA signature from the LM (456 bytes total). We present bounds based on (1) an $(f+1)^2$ protocol where the leader site would need to redundantly send 36 of these messages to each of the other 4 sites per update and (2) a $(2f+1)$ peer protocol where the leader site would redundantly send 11 messages to each site per update. The second protocol was included within the original Steward system for use during view changes, but we are unaware of any other systems that use it. The upper bound is the throughput at which the leader site's outgoing link reaches saturation. The difference between the redundant send upper bounds and the performance of Paxos/BFT (with BLink) attests to the importance of the BLink protocol.

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 this computational bottleneck. 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, because 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. Although aggregation is commonly associated with an increase in latency, the optimized protocols have similar or lower latency compared to the unoptimized

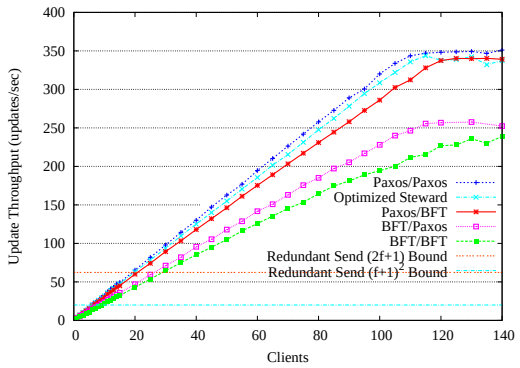

Figure 9: Throughput of Optimized Protocols, 100 ms Diameter

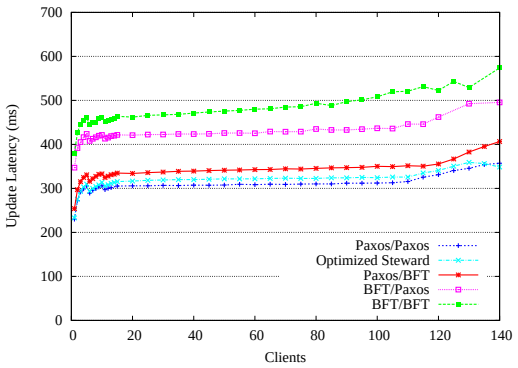

Figure 10: Latency of Optimized Protocols, 100 ms Diameter

variants. An LM 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: Our optimized composable architecture achieves practical performance, with throughputs of hundreds of updates per second, even while offering the strong security guarantees of BFT/BFT. The performance of Paxos/BFT represents a factor of 4 improvement compared with the previous state of the art for wide-area Byzantine replication (i.e., unoptimized Steward). The performance of the unoptimized protocols is computationally limited and reflects the cost associated with achieving composability and flexibility. Our results show that the optimizations effectively eliminate this performance bottleneck.

8 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: $\Delta_{mess} = (MAX_{vl} + 1)(L + (MAX_f + 2)T_{local})$, where MAX_{vl} denotes the maximum value of C_{max} across all logical links, L denotes the maximum latency due to network round trip and acknowledgment timer granularity, MAX_f is the maximum number of faults tolerated by any site, and T_{local} denotes the timeout used to detect a failed local leader. The (MAX_{vl}) term reflects the maximum number of virtual links through which a logical machine might need to rotate before reaching a correct one. The remaining term is the BLink protocol timeout, which is dependent on L and T_{local}. T_{local} is multiplied by $(MAX_f + 2)$ because there must be enough time to rotate through up to MAX_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.

9 Discussion

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 7, 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 [23] 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 [35] can also be used effectively within a site, as part of our local-area protocol. Verrisimo's work on hybrid architectures [8, 34] 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. For example, a composition could use a state machine replication protocol as the local-area protocol and a benign fault-tolerant anti-entropy protocol [13] on the wide area.

10 Conclusions

This paper presented a customizable, scalable replication architecture, tailored to systems that span multiple wide-area sites. Our architecture constructs logical machines (enhanced for use on wide-area networks) out of the physical machines in each site using the state machine approach, enabling free substitution of the fault tolerance method used in each site and in the wide-area replication protocol. We presented BLink, a new Byzantine fault-tolerant communication protocol that provides efficient and reliable wide-area communication between logical machines. BLink was shown to be a critical addition to the logical machine abstraction for wide-area networks, where bandwidth constraints limit performance. An experimental evaluation showed that our optimized architecture achieves a 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] F. V. Brasileiro, P. D. Ezhilchelvan, S. K. Shrivastava, N. A. Speirs, and S. Tao. Implementing fail-silent nodes for distributed systems. *IEEE Transactions on Computers*, 45(11):1226–1238, 1996.

[6] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI: Symposium on Operating Systems Design and Implementation*. USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS, 1999.

[7] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] R. Friedman and E. Hadad. Fts: A high-performance corba fault-tolerance service. In *7th IEEE International 10 Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002), 2002; 61–68.*, 2002.
[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.
[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] K. P. Kihlstrom and P. Narasimhan. The starfish system: Providing intrusion detection and intrusion tolerance for middleware systems. In *WORDS*, pages 191–199. IEEE Computer Society, 2003.
[16] Lamport. Paxos made simple. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 32, 2001.
[17] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
[18] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.
[19] D. Malkhi and M. Reiter. Byzantine quorum systems. *Journal of Distributed Computing*, 11(4):203–213, 1998.
[20] 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.
[21] 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.
[22] 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.
[23] J.-P. Martin and L. Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Secur. Comput.*, 3(3):202–215, 2006.
[24] M. G. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware forweb-service applications. In *SRDS '05: Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 131–142, Washington, DC, USA, 2005. IEEE Computer Society.
[25] R. C. Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, 1979.
[26] P. Narasimhan, K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. Providing support for survivable CORBA applications with the immune system. In *International Conference on Distributed Computing Systems*, pages 507–516, 1999.
[27] 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.
[28] 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.
[29] 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.
[30] 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.
[31] 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.
[32] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.
[33] V. Shoup. Practical threshold signatures. *Lecture Notes in Computer Science*, 1807:207–223, 2000.
[34] P. Verissimo. Uncertainty and predictability: Can they be reconciled. In *Future Directions in Distributed Computing*, number 2584 in LNCS. Springer-Verlag, 2003.
[35] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault-tolerant services. In *SOSP*, 2003.