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

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.,[5, 7, 27, 35]) 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 [20, 37] 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 to implement the logical machine from the protocol running on top of it. Representative systems include Voltan [6], Immune [30], BASE [35], Starfish [19], and Thema [28], which are described in more detail in Section 8. 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 po-

[*]This publication was supported by Grant 0430271 from the National Science Foundation. Its contents are solely the responsibility of the authors and do not necessarily represent the official view of Johns Hopkins University or the National Science Foundation.

tential 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 [5], 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 [29] 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.

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 a factor of 4 when running a composition with the same level of fault tolerance.

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. We also benchmarked a Byzantine over Byzantine composition, which provides fundamentally stronger fault tolerance than Steward, since Steward cannot survive a site compromise. While the systems are not strictly comparable because they offer different guarantees, the Byzantine over Byzantine composition performs 3 times better than the original Steward and achieves 35 percent lower performance than the new version of Steward that uses amortizing optimizations.

The remainder of this paper is presented as follows. In Section 2, we provide background on the state machine

[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 [5], which uses it sparingly to send global view change messages.

replication protocols used in the implementation of our composable architecture, as well as on Steward. 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. Section 8 describes related work, and Section 9 concludes the paper.

2 Background

While our composable architecture can use any of a number of state machine replication protocols as the local or wide-area protocol, this paper focuses on the use of Paxos [21, 22] and BFT [7]. This section provides an overview of Paxos and BFT. We also describe Steward [5], our previous wide-area hierarchical replication system. We benchmark our composable architecture using Paxos and BFT and compare it to the performance of Steward in Section 7.

Paxos [21, 22] 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 [7] 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 [35] 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 [5] is a hierarchical SM 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 which are associated with global view changes. 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 SM replication service. Steward can withstand f out of $3f + 1$ Byzantine failures within each site but cannot survive even a single site compromise.

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 [8]) or an update reconciliation mechanism (as in [4]) to reconcile states after a merge.

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, and the system can run either a benign fault-tolerant or a Byzantine fault-tolerant wide-area protocol. We classify both servers and sites as either correct or faulty (benign or Byzantine). A correct server adheres to its protocol specification. A benign faulty server can crash but otherwise adheres to the protocol. A Byzantine server can deviate from its protocol specification in an arbitrary way.

In what follows, we assume that Paxos is used as our benign fault-tolerant protocol and BFT is used as our Byzantine fault-tolerant protocol. Different protocol choices may require different assumptions (e.g., some Byzantine fault-tolerant protocols require a smaller fraction of faulty servers). A site running Paxos locally is benign faulty if more than f servers in the site are benign faulty, where the site has $2f + 1$ servers. A site running Paxos locally is Byzantine faulty if at least one server is Byzantine. Otherwise, the site is correct. A site running BFT is Byzantine faulty if more than f servers in the site are Byzantine, where the site has $3f + 1$ servers; otherwise the site is correct. When run on the wide area, Paxos can tolerate F benign faulty sites, where there are $2F + 1$ sites, but cannot tolerate a single Byzantine site; BFT can tolerate F Byzantine sites.

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. A client receives a reply to its update after the update has been globally ordered and executed. 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. A client retransmits its last update if no reply is received within a timeout period. 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 are computationally bounded such that they cannot subvert these cryptographic mechanisms. When BFT is deployed within a site, the servers in that site use an ($f + 1$, $3f + 1$) threshold digital signature scheme [38]. 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.*

When running Paxos on the wide area, these safety conditions hold as long as no site is Byzantine. When running BFT on the wide area, the conditions hold as long as no more than F sites are Byzantine. We refer to these conditions as the *fault assumptions needed for safety*. Since no asynchronous, fault-tolerant replication protocol tolerating even one failure can always be both safe and live [16], we provide liveness under certain synchrony conditions We first 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 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 $r = F + 1$ 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 W the STABLE-CONNECTED-SITES.

DEFINITION 3.3 L1 - GLOBAL LIVENESS: *If the system is stable with respect to time T and the fault assumptions needed for safety are met, then if, after time T, a stable server in the* STABLE-CONNECTED-SITES *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 [20, 37]. 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 SM 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 [3] for a more formal discussion of the safety and liveness properties. In the remainder of this section, we first review how we use the SM 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 [35] 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 [34]. 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 messages 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

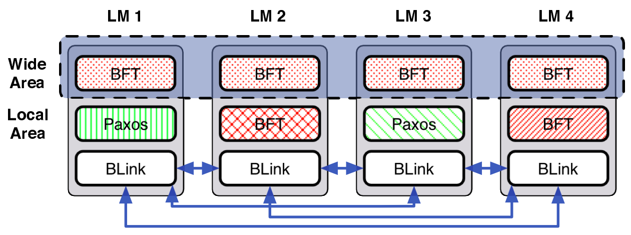

Figure 1: An example composition of four logical machines, each comprising several physical machines. The LMs receive wide-area protocol messages via BLink, which passes these messages to the local-area ordering protocol (an independent instance of either Paxos or BFT). The local-area protocol passes locally ordered messages up to the wide-area protocol (a single global instance of BFT), which executes them immediately. If a state transition causes the wide-area protocol to send a message, the LM generates a threshold signed message and passes it to Blink, which reliably transmits it to the destination logical machine.

well-known, flat replication protocols: Paxos [21, 22] as our benign fault-tolerant protocol, and BFT [7] as our Byzantine fault-tolerant replication protocol. 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.

Figure 1 shows a representative system having four logical machines, each running an independent local ordering protocol. The logical machines run a single instance of BFT on the wide-area to globally order client updates. Each LM can be configured with any number of physical machines. Since the wide-area protocol is BFT, the system can withstand a complete site compromise.

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). Client updates are sent from a local server to the leader site using a separate protocol, which is described in [3]. 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

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., [6, 28, 30]) 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 [5] 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 several sub-protocols; the sub-protocol deployed between the sending and the receiving logical machines is based on the fault tolerance method employed in each site: (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 in Section 5.2.

5.1 (Byzantine, Byzantine) Sub-protocol

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.

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. The BLink logical link is shown in Figure 2.

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 logical 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 (we define this notion more precisely below).

The properties of the logical link are based on (1) how one defines the set of virtual links that compose the logical link and (2) the order through which the sending LM proceeds through the virtual links in this set. We consider the logical link between two logical machines, LM_A and LM_B, where LM_A has $A' = 3F_A + 1$ servers, and LM_B has $B' = 3F_B + 1$ servers, with $F_A \geq F_B$. In our construction, the set of virtual links in each logical link is simply the set of all $A' \cdot B'$ possible virtual links constructed by choosing a server in LM_A and a server in LM_B. We define a *selection order* for virtual links as an infinite sequence $\langle v_0, \ldots \rangle$ of virtual links; the LM cycles through the set of virtual links according to this sequence.

We now describe the selection order used by our logical links. In what follows, $\mathrm{LCM}(x,\ y)$ denotes the least common multiple of x and y, and $\mathrm{GCD}(x,\ y)$ denotes the greatest common divisor of x and y. We define A' *series* of virtual links, each series indexed by $s \in \{0, \ldots, A'-1\}$. Within each series, there are $\mathrm{LCM}(A', B')$ virtual links, with each virtual link in the series indexed by $i \in \{0, \ldots, \mathrm{LCM}(A', B') - 1\}$. We denote virtual link i in series s as $L_{s,i}$ and define it to connect the server in LM_A with server id $i + s \bmod A'$ to the server in LM_B with server id

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

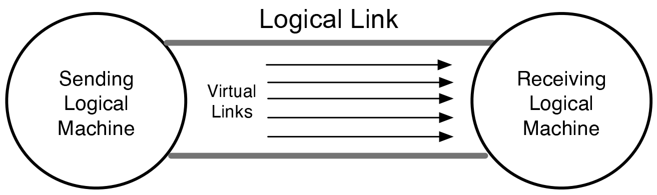

Figure 2: A logical link in the (Byzantine, Byzantine) case is constructed from $(3F_A + 1) \cdot (3F_B + 1)$ virtual links. Each virtual link consists of a forwarder and a peer. At any time, one virtual link is used to send messages on the logical link. A virtual link that is diagnosed as potentially faulty is replaced.

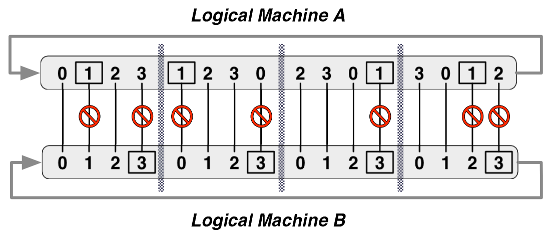

Figure 3: An example BLink logical link and selection order, with $F_A = F_B = 1$. Numbers refer to server identifiers. Boxed servers are faulty, and their associated virtual links can be blocked by the adversary. The selection order defines four series, each containing four virtual links. The order repeats after cycling through all four series.

$i \bmod B'$. We say that $L_{s,i} = L_{t,j}$ if the two virtual links connect the same pair of servers. Our construction uses the following selection order P:

$$v_{0 \le i} = L_{\lfloor i / \,\mathrm{LCM}(A',B') \rfloor, i \bmod \mathrm{LCM}(A',B')}$$

Thus, our protocol selects virtual links by taking series in ascending numerical order modulo A', starting with series 0, and within each series taking the virtual links in ascending numerical order. Figure 3 depicts an example with two logical machines, each with four servers. The selection order defines four series, each with four virtual links. Note that the servers in LM_B wrap around modulo 4, while the servers in LM_A "shift" by one position from one series to the next.

We state the following properties regarding the selection order P (proofs are provided in the extended version of this paper [3]):

1. $\forall_s \forall_i \forall_{j \neq i}$ if $s \in \{0, \ldots, A' - 1\}$ and $i, j \in \{0, \ldots, \mathrm{LCM}(A', B') - 1\}$, then $L_{s,i} \neq L_{s,j}$. In other words, each series consists of $\mathrm{LCM}(A',\ B')$ distinct virtual links.

2. $\forall_s \forall_{t \neq s} \forall_i \forall_j$, where $s, t \in \{0, \ldots, A' - 1\}$ and $i, j \in \{0, \ldots, \mathrm{LCM}(A', B') - 1\}$, if $s \not\equiv t \pmod{\mathrm{GCD}(A', B')}$ then $L_{s,i} \neq L_{t,j}$. In other words, if the indices of two series s and t are not congruent modulo $\mathrm{GCD}(A',\ B')$, then s and t contain disjoint sets of virtual links.

3. For all s, the set $S = \{s \bmod A', \ldots, (s + \mathrm{GCD}(A', B') - 1) \bmod A'\}$ of series contains $A' \cdot B'$ disjoint virtual links. In other words, proceeding through any set of $\mathrm{GCD}(A',\ B')$ consecutive series cycles through the set of all virtual links.

Given Properties 1-3, we prove the following claim about the ratio of correct virtual links (i.e., virtual links where both forwarder and peer are correct) to faulty links:

Claim: The selection order P consists of consecutive blocks of $A' \cdot B'$ virtual links, and in each block the fraction of correct virtual links is at least $4/9$.

Proof: By construction, each block consists of $\mathrm{GCD}(A', B')$ consecutive series modulo A' and hence Property 3 applies to each block. Consider any block. By Property 3, all $A' \cdot B'$ possible distinct virtual links are used exactly once in the block.

Assume that F_A servers in LM_A are faulty and F_B servers in LM_B are faulty. There are $F_A(3F_B + 1)$ virtual links that have a faulty server from LM_A. There are $F_B(3F_A + 1)$ virtual links that have a faulty server from LM_B. There are $F_A F_B$ virtual links that have both a faulty server from LM_A and a faulty server from LM_B. Taking into account the virtual links with two faulty servers, there are $F_A(3F_B + 1) + F_B(3F_A + 1) - F_A F_B$ virtual links with at least one faulty server. Let b be the fraction of virtual links with at least one faulty server. Then:

$$\begin{aligned} b &= \frac{F_A(3F_B + 1) + F_B(3F_A + 1) - F_A F_B}{(3F_A + 1)(3F_B + 1)} \\ &= \frac{5F_A F_B + F_A + F_B}{9F_A F_B + 3F_A + 3F_B + 1} \\ &\le 5/9 \end{aligned}$$ ∎

In addition to the ratio of correct virtual links to faulty virtual links, we are also interested in the maximum number of consecutive faulty links through which the LM must cycle before reaching a correct virtual link. We refer to this value as VL_{max}. In [3], we show that VL_{max} is bounded at $2F_A$.

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 suspect that the virtual link is faulty and elect the next forwarder. 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.

Fairness: The third technique used by BLink 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 receives an incoming message, it disseminates the message within the site; all servers then forward the message to the leader of the agreement protocol and expect it to initiate the message for ordering such that the message 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 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. We describe our mechanism for preventing the starvation of any particular client in [3].

5.2 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. We first consider the (benign, Byzantine) and (Byzantine, benign) cases. As in the (Byzantine, Byzantine) case, the number of virtual links that compose each logical link is equal to the number of servers in LM_A times the number of servers in LM_B. In the following discussion, we assume that the number of servers in LM_A, A', is greater than or equal to the number of servers, in LM_B, B'. If LM_A runs a Byzantine fault-tolerant protocol and LM_B runs a benign fault-tolerant protocol, then $3F_A+1 \geq 2F_B+1$. Otherwise, we have $2F_A + 1 \geq 3F_B + 1$.

We use the same selection order as for the (Byzantine, Byzantine) case, and we use an argument similar to the one found in Section 5.1 to obtain the ratio of correct to faulty virtual links. In the extended version of this paper [3], we show that at least $1/3$ of the virtual links are correct. Further, when LM_A is Byzantine fault-tolerant, the maximum number of consecutive faulty links (VL_{max}) is bounded at $\max(\lfloor 2.5F_A \rfloor,\ 3F_A - 2)$; when LM_A is benign fault-tolerant, VL_{max} is bounded at $\max(2F_A - 1,\ \lfloor \frac{5}{3}F_A \rfloor)$. Intuitively, the difference in the bounds is attributed to the difference in the ratio of faulty servers within LM_B: when LM_B is Byzantine, the ratio is only $1/3$, but when LM_B is benign, the ratio is $1/2$.

In the (benign, benign) case, each logical link consists of $(2F_A + 1) \cdot (2F_B + 1)$ virtual links. This yields a ratio of $1/4$ correct virtual links. Since no server in either LM is Byzantine, it is possible to use a simple and efficient selection order to cycle through the virtual links. 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.

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

Sub-protocol	Correct links	VL_{max} upper bound
(Byz, Byz)	$4/9$	$2F_A$
(Byz, Benign)	$1/3$	$\max(\lfloor 2.5F_A \rfloor, 3F_A - 2)$
(Benign, Byz)	$1/3$	$\max(\lfloor 2.5F_A \rfloor, 3F_A - 2)$
(Benign, Benign)	$1/4$	$F_A + F_B$

Table 1: The ratio of correct virtual links and the maximum number of consecutive faulty virtual links for each BLink sub-protocol.

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. In the extended version of this paper, we show that VL_{max} is bounded at $F_A + F_B$. We can use a similar strategy in the (benign, Byzantine) case; however, the technique is not applicable to the (Byzantine, benign) case, since the forwarder cannot be trusted to find a correct peer.

We summarize our results in Table 1.

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 [29] 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 [8] 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.

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 [5]. 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. Thus, 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. Wide-area links were limited 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,

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 2: Normal-case 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 3: Number of expensive cryptographic operations that each server at the leader site does per update during normal-case operation.

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 2 shows the number of normal-case protocol rounds, where view changes do not occur, 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. 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 3 shows the computationally expensive cryptographic operations required for each update during normal-case operation at the leader site 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. Note that 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 4 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 reaches 100%, throughput plateaus. This graph shows the performance benefit of Steward's architecture. In Steward, external wide-area accept messages are not ordered before the replicas 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 3). 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 5 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 6 and 7 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 6 (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 3 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 band-

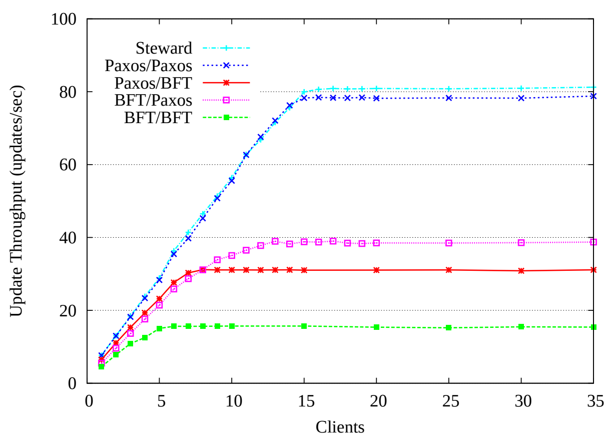

Figure 4: Throughput of Unoptimized Protocols, 50 ms Diameter

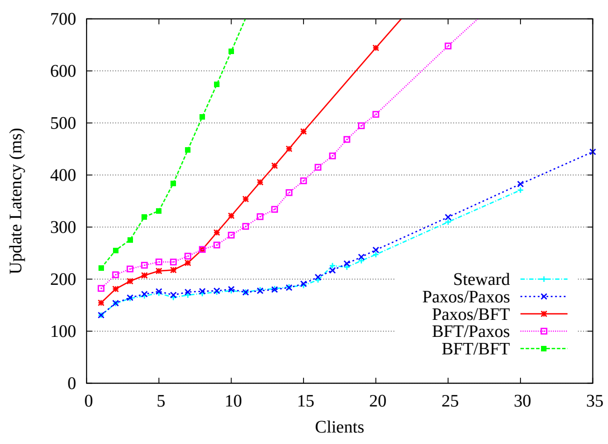

Figure 5: Latency of Unoptimized Protocols, 50 ms Diameter

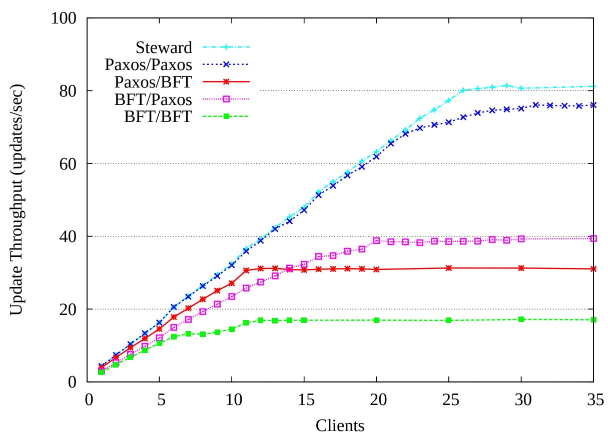

Figure 6: Throughput of Unoptimized Protocols, 100 ms Diameter

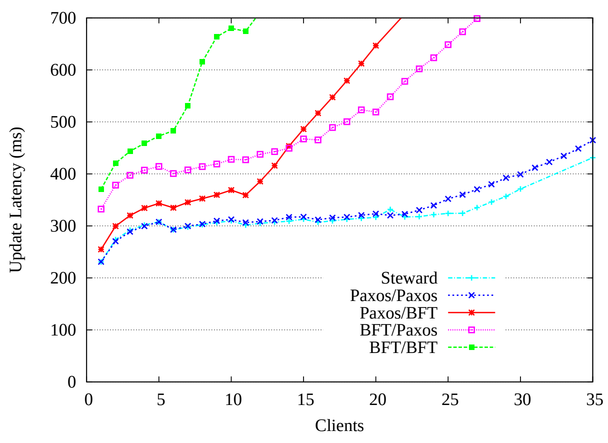

Figure 7: Latency of Unoptimized Protocols, 100 ms Diameter

width constraints. In all cases, the optimized protocols increased throughput by at least a factor of 4 compared to their unoptimized versions.

In Figures 8 and 10 (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 communication. 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 8 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

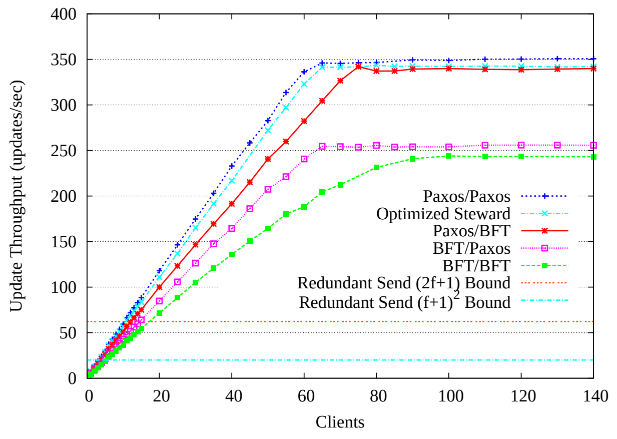

Figure 8: Throughput of Optimized Protocols, 50 ms Diameter

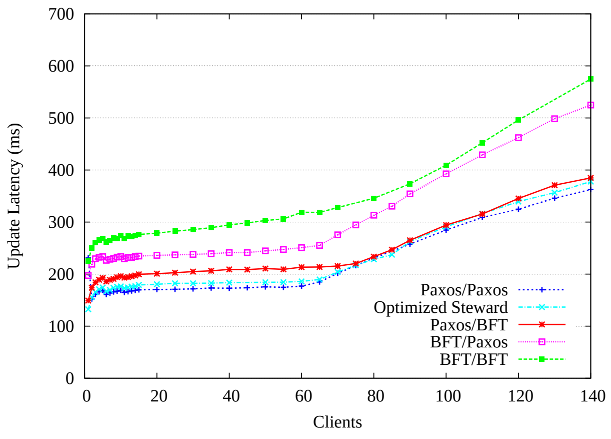

Figure 9: Latency of Optimized Protocols, 50 ms Diameter

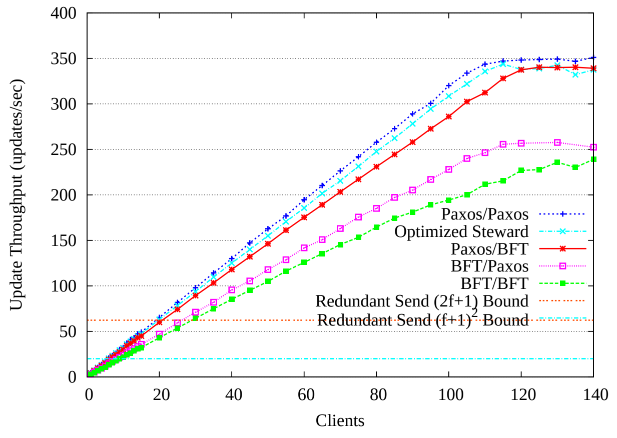

Figure 10: Throughput of Optimized Protocols, 100 ms Diameter

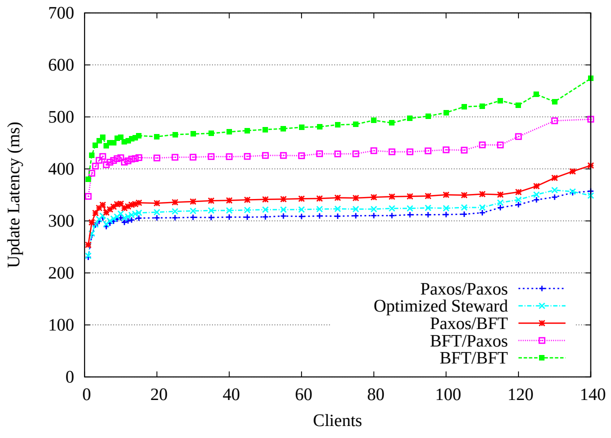

Figure 11: Latency of Optimized Protocols, 100 ms Diameter

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 9 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 variants. An LM locally orders 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 that of the unoptimized protocols, because the optimized protocols have much higher maximum throughput. Figures 10 and 11 show 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 Related Work

State Machine Replication: Lamport [20] and Schneider [37] 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 SM approach has been used in many systems to construct fault-tolerant logical machines out of collections of physical machines. We mention several of these systems here.

Schlichting and Schneider [36] 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 Delta-4 system [31] uses an intrusion-tolerant architecture and provides services for data authentication, storage, and authorization. Like our current work, the system constructs logical entities out of multiple physical machines via the SM approach; it also employs protocols to make communication between the logical entities efficient. However, these protocols assume that the communicating parties are fail-silent, whereas our current work constructs Byzantine fault-tolerant links between logical entities.

The Voltan system of Brasileiro, et al. [6] uses the SM 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. Each message send from one logical node to another requires sending four physical messages over the network, reducing the system's applicability to bandwidth-constrained wide-area environments.

The Starfish system of Kihlstrom and Narasimhan [19] builds an intrusion-tolerant middleware service by using a hierarchical membership structure and end-to-end intrusion detection. The system uses a central, hardened core that offers strong security guarantees. The core is augmented by "arms," with weaker security guarantees, that can be removed in the case of a security breach.

The Thema system of Merideth, et al. [28] uses SM replication to build Byzantine fault-tolerant Web Services. Standard Web Service clients access the Byzantine fault-tolerant service using a client library. Thema allows Byzantine fault-tolerant services to safely access non-replicated Web Services.

The MAFTIA system of Verissimo, et al. [40] uses architectural hybridization to build mechanisms for intrusion tolerance by transforming untrusted components into trusted components. The hybrid architecture is built in the wormhole model [39], where different parts of the system operate under different fault assumptions and are thus resilient to different types of attack. For example, if trusted components (e.g., a reliable channel, a processor whose results can be trusted) are available, the system can be configured to run protocols that take advantage of them to achieve increased performance (e.g., [9]). In contrast, our system assumes all components are untrusted, allowing a similar performance-resilience tradeoff by deploying either a benign or a Byzantine fault-tolerant protocol in each site and on the wide area.

Byzantine Fault-tolerant SMR Protocols: Although our prototype system deployed BFT [7], we mention several other Byzantine fault-tolerant agreement protocols that can be used for SM replication.

Doudou, et al. [13] decompose the problem of Byzantine fault-tolerant SMR via a series of abstractions. The replication problem is reduced to an atomic multicast protocol, which itself is composed of a reliable multicast component and a solution to the weak interactive consistency problem. The latter uses a Byzantine failure detector for detecting mute processes (i.e., processes from which, from some time on, a correct process stops receiving messages).

Yin, et al. [41] 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.

Correia, et al. [10] reduce the number of replicas needed for SM replication from $3f+1$ to $2f+1$ by augmenting the Byzantine, asynchronous model with a distributed trusted component, the Trusted Timely Computing Base (TTCB). The local TTCBs of the replication servers are assumed not to be malicious, and they communicate over a synchronous control network providing real-time delay guarantees. The TTCBs run a fault-tolerant protocol to assign an ordering to protocol messages, and these protocol messages are then exchanged over the asynchronous payload network.

Martin and Alvisi [27] recently introduced a two-round Byzantine consensus algorithm, which uses $5f+1$ servers to overcome f faults. Their approach trades potentially lower availability for increased performance. The protocol is well-suited for use in our architecture as the wide-area protocol, since it would reduce the number of wide-area crossings from three to two.

Fault-tolerant CORBA: State machine replication has also been used to increase the fault-tolerance and availability of CORBA services.

The Object Group Service (OGS) of Felber, et al. [15] provides a composable, modular architecture for replicating CORBA objects. The OGS implements several component CORBA services, such as group multicast, group membership, and distributed consensus, which are then composed to implement group communication services; this group communication service is then used for replication.

The FTS system of Friedman and Hadad [17] uses active replication to construct a lightweight fault tolerance service for CORBA. The system survives network partitions, allowing updates in a single partition but allowing other partitions to remain alive until they reconnect.

The Immune system of Narasimhan, et al. [30] provides support for survivable CORBA applications by replicating both client and server objects. When a replicated client objects invokes an operation on a replicated server object, each client object sends a message to each server object via the

SecureRing multicast protocol [18], and the servers employ majority voting to mask faulty behavior; the responses are sent from server to client in similar fashion. While the logical machines in our architecture could use SecureRing to communicate with one another (with one group for each pair of neighboring logical machines), doing so would result in many redundant messages being sent over the wide-area network during normal-case operation, greatly limiting performance.

Byzantine Fault-tolerant Group Communication: Also related to our work are group communication systems resilient to Byzantine failures [12, 14, 18, 32, 33].

Both Rampart [33] and SecureRing [18] provide services for messaging (atomic multicast) and membership. Rampart uses a designated server in each view, the sequencer, to assign an ordering to messages, while SecureRing orders messages via a logical token ring. Both systems rely on a Byzantine failure detector for liveness and maintain safety provided that there are fewer than one third faulty servers.

The ITUA system [12, 32], developed by BBN Technologies and the University of Illinois at Urbana-Champaign, focuses on providing intrusion-tolerant group services. It employs the principle of unpredictable adaptability to overcome intelligent adversaries.

Drabkin, et al. [14] describe a Byzantine version of the JazzEnsemble system, providing a formal definition of Byzantine virtual synchrony. The system uses the idea of fuzzy membership: each node is given an indication of how fuzzy the other group members are, with low fuzziness indicating a well-responding server and high fuzziness indicating a server that is not very responsive. Detection of Byzantine behavior in this context is encapsulated by fuzzy mute and fuzzy verbose failure detectors.

Quorum Replication: Quorum systems obtain Byzantine fault tolerance by applying quorum replication methods [23]. Examples of such systems include Phalanx [26] and Fleet [24, 25]. The HQ protocol [11] 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.

9 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, B. Coan, J. Kirsch, and J. Lane. Customizable fault tolerance for wide-area replication. Technical Report CNDS-2007-3, Johns Hopkins University, www.dsn.jhu.edu, August 2007.

[4] 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.

[5] 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 *Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN'06)*, pages 105–114, Philadelphia, PA, USA, June 2006. IEEE Computer Society.

[6] 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.

[7] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the 1999 Symposium on Operating Systems Design and Implementation (OSDI '99)*, pages 173–186, New Orleans, LA, USA, 1999. USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS.

[8] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002.

[9] M. Correia, L. C. Lung, N. F. Neves, and P. Veríssimo. Efficient byzantine-resilient reliable multicast on a hybrid failure model. In *Proceedings of the 21st Symposium on Reliable Distributed Systems (SRDS '02)*, pages 2–11, Suita, Japan, Oct. 2002.

[10] M. Correia, N. F. Neves, and P. Veríssimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*, pages 174–183, Florianpolis, Brazil, 2004. IEEE Computer Society.

[11] 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 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*, pages 177–190, Seattle, WA, Nov. 2006.

[12] 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.

[13] A. Doudou, R. Guerraoui, and B. Garbinato. Abstractions for devising byzantine-resilient state machine replication. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, pages 144–153, Nurnberg, Germany, 2000. IEEE Computer Society.

[14] V. Drabkin, R. Friedman, and A. Kama. Practical byzantine group communication. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS '06)*, page 36, Lisboa, Portugal, 2006. IEEE Computer Society.

[15] P. Felber, R. Guerraoui, and A. Schiper. Replication of CORBA objects. In *Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems*, pages 254–276, London, UK, 1999. Springer-Verlag.

[16] 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.

[17] R. Friedman and E. Hadad. FTS: A high-performance CORBA fault-tolerance service. In *Proceedings of the 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '02)*, pages 61–68, San Diego, CA, USA, 2002. IEEE Computer Society.

[18] 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.

[19] K. P. Kihlstrom and P. Narasimhan. The Starfish system: Providing intrusion detection and intrusion tolerance for middleware systems. In *Proceedings of the 8th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '03)*, pages 191–199, Guadalajara, Mexico, 2003. IEEE Computer Society.

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

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

[22] L. Lamport. Paxos made simple. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 32:18–25, 2001.

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

[24] 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.

[25] D. Malkhi, M. Reiter, D. Tulone, and E. Ziskind. Persistent objects in the Fleet system. In *Proceedings of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX II)*, volume 2, pages 126–136, June 2001.

[26] D. Malkhi and M. K. Reiter. Secure and scalable replication in Phalanx. In *Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems (SRDS '98)*, pages 51–58, West Lafayette, IN, USA, 1998. IEEE Computer Society.

[27] J.-P. Martin and L. Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

[28] M. G. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service applications. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 131–142, Orlando, FL, USA, 2005. IEEE Computer Society.

[29] R. C. Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, Stanford University.

[30] P. Narasimhan, K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. Providing support for survivable CORBA applications with the Immune system. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, pages 507–516, Austin, TX, USA, 1999.

[31] D. Powell, D. Seaton, G. Bonn, P. Veríssimo, and F. Waeselynck. The Delta-4 approach to dependability in open distributed computing systems. In *Proceedings of the 18th IEEE International Symposium on Fault-Tolerant Computing (FTCS)*, pages 246–251, June 1988.

[32] 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 *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN '02)*, pages 229–238, Bethesda, MD, USA, June 2002.

[33] 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.

[34] 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.

[35] R. Rodrigues, M. Castro, and B. Liskov. BASE: using abstraction to improve fault tolerance. In *Proceedings of the 18th ACM symposium on Operating systems principles (SOSP '01)*, pages 15–28, Banff, Alberta, Canada, 2001. ACM Press.

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

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

[38] V. Shoup. Practical threshold signatures. *EUROCRYPT 2000, Lecture Notes in Computer Science*, 1807:207–220, 2000.

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

[40] P. Veríssimo, N. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I. Welch. Intrusion-tolerant middleware: The road to automatic security. *IEEE Security & Privacy*, 4(4):54–62, 2006.

[41] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault-tolerant services. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 253–267, Bolton Landing, NY, USA, October 2003.