An Attack-Resilient Architecture for Large-Scale Intrusion-Tolerant 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-2009-5
October 2009

Abstract

This paper presents the first architecture for large-scale, wide-area intrusion-tolerant state machine replication that is specifically designed to perform well even when some of the servers are Byzantine. The architecture is hierarchical and runs attack-resilient state machine replication protocols within and among the wide-area sites. Given the constraints of the wide-area environment, we explore the challenges and tradeoffs of building inter-site communication protocols that use wide-area bandwidth efficiently yet can resist attempts to degrade performance. The paper provides evidence that the optional use of simple dependable components, whose compromise or malfunction cannot cause inconsistency in the replicated service, can significantly improve performance when the system is under attack.

1 Introduction

Much of our critical infrastructure is controlled by large software systems whose participants are distributed across the Internet. As our dependence on these critical systems grows, we require them to meet more and more stringent availability and performance requirements, even in the face of attacks, including those mounted by malicious insiders. This paper is about how to architect large-scale, survivable replication systems that guarantee correctness, availability, and good performance even when some of the servers are compromised.

The last decade has seen the introduction of two distinct generations of intrusion-tolerant state machine replication protocols. The first generation of protocols focused on achieving *safety* (i.e., consistency) as long as the system's fault assumptions hold, and *liveness* (i.e., the eventual execution of each submitted operation) when the network is sufficiently synchronous. Beginning with Castro and Liskov's BFT protocol [14] in 1999 and continuing throughout the 2000s (e.g., [17, 18, 20, 33]), protocols in this class were shown to be capable of providing good performance in fault-free executions in small-scale, local-area network settings. In parallel, the Steward system [8] and customizable replication architecture in [6] showed how to leverage a hierarchical architecture to improve the scalability of these systems in large wide-area deployments.

All of the protocols described above share a common problem: While they perform well when all of the servers behave correctly, their performance can be made dramatically worse when one or more servers actually has a Byzantine fault. For this reason, the next generation of protocols has focused on achieving stronger performance guarantees than liveness when the system is under attack; we call such protocols *attack-resilient*. The Prime system [7] was the first intrusion-tolerant replication protocol to provide a meaningful performance guarantee in the face of Byzantine servers by bounding the amount of delay that they can cause. The Aardvark protocol of Clement et al. [15] can guarantee meaningful throughputs over sufficiently long periods, and it provides important system engineering techniques that can significantly improve robustness to flooding-based attacks. New protocols in this direction, such as the Spinning protocol of Veronese et al. [32], continue to explore the terrain.

Despite their attack resilience, these second-generation protocols employ flat architectures that are not well-suited to the large-scale, wide-area deployments needed by our critical infrastructure. This paper presents the first architecture for large-scale, wide-area intrusion-tolerant state machine replication that is specifically designed to perform well even when some of the servers are Byzantine, thus unifying the first-generation large-scale systems with the second-generation attack-resilient systems.

Our system uses a hierarchical architecture and is

*This publication was supported by grant 0716620 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.

suited to wide-area deployments consisting of several sites, each with a cluster of replication servers, all of which participate in a system-wide replication protocol. Unfortunately, achieving system-wide attack resilience is not as simple as deploying attack-resilient protocols in each level of the hierarchy (i.e., within each site and on the wide area). As the paper demonstrates, a critical component of the architecture that must be hardened against performance degradation is the mechanism by which two sites communicate, which we call the *logical link protocol*. The logical link protocol defines which physical machines pass wide-area messages on behalf of the site and to which machines they send. Given that the performance of wide-area replication tends to be constrained by the limited wide-area bandwidth between sites, the challenge is to build a logical link that is attack-resilient *and* that uses wide-area bandwidth efficiently so that performance remains acceptably high both when the system does and does not exhibit Byzantine faults. Existing approaches achieve one but not the other: Having many servers send on behalf of the site (e.g., [12, 23]) masks the behavior of faulty senders but can be inefficient, while having one elected server pass messages on behalf of the site (e.g., [6]) is efficient but vulnerable to performance degradation when the server is faulty.

If each site had access to a hardened forwarding device capable of sending wide-area messages exactly once and in a timely manner, it would be relatively straightforward to achieve attack resilience while using wide-area bandwidth efficiently. However, if the compromise of such a device can cause inconsistency in the replicated service (as in [28]), then deploying such a trusted forwarder can improve performance but potentially decrease the system's robustness. Therefore, this paper explores the design space of how to build efficient, attack-resilient logical links *without* increasing the system's vulnerabilty to safety violations. In essence, we consider how close one can get to the benefits of a trusted forwarder without suffering its drawbacks.

We explore the tradeoffs of deploying three logical link protocols, each offering different levels of performance and requiring different levels of assumptions about the environment. The first approach is an erasure encoding based logical link that does not require any special components or additional assumptions but which has the highest bandwidth overhead of the three protocols we consider. The second approach demonstrates that by assuming a functional broadcast hub in each site (where each local server receives a copy of any message that passes through the hub), one can significantly improve throughput both in fault-free and under-attack executions. The third approach shows that by assuming each correct site has access to a simple forwarding device capable of counting and sending messages, the system can achieve optimal wide-area bandwidth usage without decreasing robustness. Because of the cryptographic protection (i.e., threshold signatures) used on inter-site messages, the compromise of the simple forwarding devices cannot lead to safety violations (although it can impact performance negatively).

We discuss the tradeoffs and practicality of the logical links and evaluate their performance in a prototype implementation, both in fault-free and under-attack scenarios. Our results provide evidence that it is possible to construct a large-scale wide-area replication system that achieves reasonable performance under attack, and that leveraging dependable components implementing fairly limited functionality can significantly improve the performance of a fault-tolerant distributed system.

We note that all three logical link protocols are generic and can be of use in any application where sets of machines need to pass messages to each other in an attack-resilient way. Thus, they may shed some insight relevant to constructing intrusion-tolerant systems that goes beyond state machine replication.

The remainder of this paper is presented as follows. Section 2 describes our system model. Section 3 presents our hierarchical architecture and describes the components that must be hardened against performance degradation. In Section 4 we describe three mechanisms for achieving attack-resilient wide-area (inter-site) communication. Section 5 shows how all of the pieces fit together and describes the service properties achieved by the resulting system. In Section 6 we evaluate the performance of our prototype implementation. Section 7 describes related work, and Section 8 concludes the paper.

2 System Model

We assume a Byzantine fault model in which servers and clients are either *correct* or *faulty*; correct processes follow their protocol specification exactly, while faulty processes can deviate from the protocol arbitrarily. We consider a system with N sites, denoted S_1 through S_N, distributed across a wide-area network. Each site, S_i, has $3f_i + 1$ servers. If S_i is a correct site, then no more than f_i of its servers are faulty; if S_i is a Byzantine site, then any number of its servers may be faulty, modeling situations where entire sites can be compromised. We denote F as an upper bound on the number of Byzantine sites and assume that the total number of sites is equal to $3F + 1$. For simplicity, we assume in what follows that all sites tolerate the same number of faults, f, and have the same number of servers, $3f + 1$. The solutions presented can be extended to the more general setting, where sites may have different numbers of servers.

Servers communicate by passing messages. Messages may be delayed, lost, or duplicated. All messages sent

between servers are digitally signed. We assume that digital signatures are unforgeable without knowing a server's private key. We use an $(f+1,\ 3f+1)$ threshold digital signature scheme for generating threshold signatures on inter-site (wide-area) messages. Each site has a public key, and each server within a site is given a secret share that can be used to generate partial signatures. We assume threshold signatures are unforgeable without knowing the secret shares of $f+1$ servers within a site. We also employ a cryptographic hash function for computing message digests.

Clients submit read-only (*query*) and read/write (*update*) operations to the system by communicating with the servers in their local site. Any number of clients may be faulty. Our system totally orders the submitted operations so that the servers can execute them in the same order and remain consistent.[1] The system as a whole meets the modified linearizability condition specified in [14], which states that the replicated service acts like a centralized implementation that atomically executes operations one at a time. This safety property holds in all executions in which there are at most F Byzantine sites. We state the liveness and performance guarantees of our system in Section 5.

Our system can be deployed with one of several logical link protocols for inter-site communication, two of which rely on *dependable components*. In the hub based logical link (see Section 4.2), each site is equipped with a broadcast hub through which incoming and outgoing wide-area traffic passes. In the dependable forwarder based logical link (see Section 4.3), each site is equipped with a dependable forwarding (DF) device that sends and receives inter-site messages on behalf of the site. We assume each DF shares a symmetric key with each other DF and with each local server for computing message authentication codes. The failure (crash or compromise) of the dependable components can impact performance and liveness but cannot lead to safety violations.

3 Building an Attack-Resilient Architecture for Wide-Area Replication

In this section we describe the components of an attack-resilient architecture for large-scale wide-area replication. Our architecture builds on our previous work on wide-area Byzantine replication [6, 8], which demonstrated the performance benefit of using hierarchy to reduce wide-area message complexity. We first provide background on the hierarchical architecture and then discuss how to harden it against performance degradation.

[1] As in BFT [14], an optimistic protocol can be used to respond to queries without totally ordering them. The optimistic protocol may fail if there are concurrent updates, in which case the query can be resubmitted as an update operation and totally ordered.

3.1 Background: Logical Machines

The physical machines in each site are converted into a *logical machine* that is capable of processing protocol events (i.e., message reception and timeout events) just as a physical machine would. Each logical machine acts as a single participant in a global, wide-area replication protocol that runs among the logical machines. Thus, the logical machines process wide-area protocol events.

In order to support the abstraction of a logical machine, the physical machines in each site run a local state machine replication protocol to totally order any event that would change the state of the logical machine. Thus, the local state machine replication protocol orders events corresponding to either the reception of a message or the firing of a timeout by the logical machine. A physical machine processes a global protocol event when it *locally executes* it, which occurs after the machine learns of the event's local ordering and after it has locally executed all previous events. The local and global protocols are cleanly separated, allowing one to plug in different protocols in each site and on the wide area.

When the logical machine processes an event, it may generate and send a message in the global protocol. Before the message can be sent on the wide area, the physical machines implementing the logical machine run a protocol to generate a threshold signature on the message. The threshold signature proves that at least one correct physical machine in the site assents to the content of the associated message, preventing faulty machines in correct sites from sending spurious messages that purport to be from the logical machine. Once a message is threshold signed, it can be sent to its destination sites according to the communication patterns of the global replication protocol; we say that the message is sent over a *logical link* that exists between each pair of sites.

3.2 Components of the Architecture

There are four pieces of our attack-resilient architecture that must be hardened against performance degradation: the global state machine replication protocol, the local state machine replication protocol, the threshold signature protocol, and the logical links that connect the logical machines. Making the logical links attack-resilient is of critical importance, and we defer a discussion of this topic until Section 4. The threshold signature protocol can be hardened by using a scheme in which partial signatures are *verifiable*, meaning they carry proofs of correctness that can be used to detect (and subsequently blacklist) faulty servers that submit invalid partial signatures. Subsequent messages from blacklisted servers are ignored, preventing them from repeatedly disrupting threshold signature generation. A representative example of such a scheme (and the one used in our implementation) is Shoup's threshold RSA signature scheme [26].

In the remainder of this section, we describe the implications of deploying different local and global state machine replication protocols, both to motivate the choices we made in our implementation and to point out how the choices impact the attack resilience of the system as a whole. Our goal is not to invent new attack-resilient state machine replication protocols but rather to illustrate the tradeoffs of deploying different existing protocols.

We know of two state machine replication protocols that do not rely on trusted components and which offer provable performance guarantees even when some participants are Byzantine: Prime [7] and Aardvark [15]. Informally, Prime bounds the latency of operations submitted to and subsequently introduced by correct servers, while Aardvark guarantees that over sufficiently long periods, system throughput will be within a constant factor of what it would be with only correct servers participating in the protocol. While other protocols (e.g., [22, 32]) may be difficult to attack in practice, in this paper we restrict our attention to Prime and Aardvark, because comparing the differences between them is enough to give the flavor of the various design choices at issue.

Global State Machine Replication Protocol: Since we explicitly allow that some of the logical machines can be Byzantine, the global replication protocol must be attack-resilient, just as if it were running among physical machines instead of logical machines.

We chose to use Prime as our global protocol, rather than Aardvark, because Prime makes more efficient use of wide-area bandwidth, which is likely to be the performance bottleneck in wide-area replication systems. In Aardvark, the primary is responsible for disseminating client requests (by batching them into PRE-PREPARE messages), limiting throughput when performance is bandwidth-limited to the number of requests that can be disseminated by the primary per second. In contrast, each Prime participant disseminates requests from its own clients. Therefore, assuming the majority of outgoing bandwidth is used to disseminate requests (which we expect to be the case for all but very small request sizes), the peak throughput of Prime has the potential to be larger by a factor of the number of sites in the system.[2]

We note that in evironments where the risk of total site compromise is small, the global protocol can be benign fault-tolerant rather than Byzantine fault-tolerant and attack-resilient; this was the approach taken in Steward [8]. This results in a more efficient protocol that requires only two wide-area crossings. The logical link protocols described in Section 4 remain an integral part of the architecture to avoid performance degradation, even when a benign fault-tolerant global protocol is used.

Local State Machine Replication Protocol: The performance of the local state machine replication protocol determines the processing capability of the logical machine. Put another way, attacks that degrade the throughput or increase the request latency of the local protocol can result in a logical machine that takes longer to process global protocol events. Such attacks have both practical and theoretical implications. Practically, they result in performance degradation in the global replication service, even if enough sites are correct and the network is sufficiently stable. Theoretically, they may cause the timing assumptions of the global protocol to be violated, which may threaten system-wide liveness or performance guarantees. Therefore, it is important to deploy a local state machine replication protocol that will result in a logical machine with the performance and timing properties needed by the global protocol.

As we discuss in Section 5, when Prime is deployed as the global replication protocol, it requires that the logical machine be able to process certain messages in a bounded time; this is precisely the property that a Prime-based logical machine provides when (1) all events are introduced by at least one correct server, which our architecture guarantees, and (2) there is sufficient local bandwidth to avoid queuing, which is likely to be the case in well-provisioned LANs.

Despite the strong throughput guarantee that Aardvark makes over sufficiently long intervals, it has the potential for institutionalized periods of low throughput during the grace periods that begin views with faulty primaries. This means that Aardvark does not guarantee that *individual requests* are executed in a timely manner, even though long-term overall throughput is high. This can enable faulty local primaries in *correct sites* to cause the global protocol to take more expensive execution paths (i.e., cause a correct leader site to be suspected). Although individual requests may also be delayed in Prime when the local leader is faulty, the key difference is that Prime will eventually settle on leaders that do not cause delay, while Aardvark will perpetually be vulnerable to periods in which latency is temporarily increased. While the global instance of Prime can be configured to tolerate latency variability, increasing this variability increases the attacker's ability to cause delay and should be avoided if possible. Therefore, we chose Prime as our local state machine replication protocol.

4 Attack-Resilient Logical Links

The physical machines within a site construct and threshold sign global protocol messages after locally executing

[2] One might try to close this gap by relying on clients to disseminate requests and having the primary propose an order on batches of digests (as in [13]). However, if faulty clients disseminate their requests to only $f+1$ correct servers, they can cause the other f correct servers to learn the order of a request without having the request itself. These servers cannot execute subsequent requests until they fill this hole, which may lead to repeated performance degradation.

global protocol events. This raises the question of how to pass the threshold-signed message from the sending logical machine to a destination logical machine. Each correct server that generates the threshold-signed message is capable of passing it to any server in the destination site. We must define a *logical link protocol* to dictate which local server or servers send, what they send, and to which server or servers they send it.

The challenge in designing a logical link protocol is to simultaneously achieve attack resilience and efficiency. Existing approaches used in logical machine architectures (e.g., [6, 12, 23]) achieve one but not the other. For example, if $f + 1$ physical machines in the sending site each transmit the threshold-signed message to $f + 1$ physical machines in the receiving site, then at least one correct machine in the receiving site is guaranteed to receive a copy of the message – at least one of the senders is correct, and at least one of that correct machine's receivers is correct. Such a logical link is attack-resilient, because faulty machines cannot prevent a message from being successfully transmitted in a timely manner, but the protocol pays a high cost in wide-area bandwidth, transmitting each message up to $(f + 1)^2$ times.

Due to the overhead of sending messages redundantly, our previous work [6] adopted a different approach, called the BLink protocol, whereby the physical machines in each site elect one machine to act as a *site forwarder*, charged with the responsibility of sending messages on behalf of the site. The physical machines also choose the identifier of the machine in the receiving site with which the forwarder should communicate. The non-forwarders use timeouts, coupled with acknowledgements from the receiving site, to monitor the forwarder and ensure that it passes messages at some minimal rate. If the current (forwarder, receiver) pair is deemed faulty, a new pair is elected.

BLink is efficient but not attack-resilient: the forwarder and receiver can collude to remain in power as long as they ensure that the forwarder collects acknowledgements just before the timeout expires, resulting in much lower throughput and higher latency on the logical link than correct machines would provide. Using a more aggressive approach to monitoring (by attempting to determine how fast the forwarder should be sending messages) requires additional timing and bandwidth assumptions which may be difficult to realize in practice.

Note that BLink struggles in the presence of Byzantine faults because it was built to ensure liveness, not to achieve attack resilience. Liveness requires the logical link to make minimal progress – and, for this purpose, a coarse-grained timeout works well. BLink obtains high normal-case performance by depending on the site forwarder to pass messages, but giving a single machine this power is precisely what makes the protocol vulnerable to performance degradation by a malicious forwarder.

In the remainder of this section, we present and compare three new attack-resilient logical link protocols. The design of the three protocols brings to light a tradeoff between the strength of one's assumptions and the resulting performance that one can achieve, with each protocol representing a different point in the design space. All three protocols share the same goals:

Attack Resilience. Like the local and global state machine replication protocols, the logical link protocol should limit or remove the power of the adversary to cause performance degradation, without unduly sacrificing normal-case performance.

Modularity. It should be possible to substitute one logical link protocol for another without impacting the correctness of the global replication protocol, allowing deployment flexibility based on what system components one wishes to depend on. Conversely, the logical link protocol should be generic enough so that it can be used with different wide-area replication protocols.

Simplicity. Given the inherent complexity of intrusion-tolerant replication protocols, the logical link protocols should be easy to reason about and straightforward to implement.

Section 4.1 presents a logical link that requires no dependable components and that erasure encodes outgoing messages to reduce the cost of sending redundantly. Section 4.2 shows how augmenting the erasure encoding approach with a broadcast hub can improve performance in fault-free and under-attack executions. Section 4.3 describes how relying on a dependable forwarder can yield an optimal use of wide-area bandwidth without making it easier to cause inconsistency. Table 1 at the end of this section summarizes our results. We evaluate the performance of the logical links in Section 6.

4.1 Erasure Encoding Based Logical Link

We first present a simple, software-based logical link protocol. In what follows, we consider how a sending site, S, passes a threshold-signed message to a receiving site, R. We define *virtual link* i as the ordered pair $(s_i,\ r_i)$, where s_i and r_i refer to the physical machines with identifier i in sites S and R, respectively. We call s_i and r_i *peers*. Communication over the logical link takes place between peers using the set of $3f + 1$ virtual links.

Instead of having each physical machine in S transmit the full threshold-signed message to its peer in R, the physical machines first encode the message using an MDS($3f + 1,\ f + 1$) maximum distance separable erasure encoding [11, 24]. The message is encoded into $f + 1$ *message parts* and $2f$ *redundant parts* such that

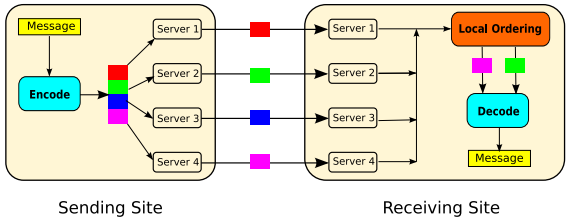

Figure 1: An example erasure encoding based logical link, with $f = 1$.

any combination of $f + 1$ parts can be used to decode and recover the original message. We number the parts 1 through $3f + 1$. To transmit an encoded message across the logical link, machine i in site S sends part i to its peer on the corresponding virtual link. The erasure-encoded parts are locally ordered in R as they arrive. When the physical machines in R locally execute $f + 1$ parts, they decode them to recover the original message, which can then be processed by the logical machine. The procedure is depicted in Figure 1.

The erasure encoding based logical link allows messages to be passed correctly and without delay. To understand why, observe that if both S and R are correct sites, then since at most f physical machines can be faulty in each site, at least $f + 1$ of the $3f + 1$ virtual links will have two correct peers (see Figure 2); we call such virtual links *correct*. Erasure encoded parts passed on correct virtual links cannot be dropped or delayed by faulty machines. Therefore, when a message is encoded, at least $f + 1$ correctly generated parts will be sent in a timely manner and subsequently received and introduced for local ordering in R. Since $f + 1$ parts are necessary and sufficient to decode, the physical machines in R will be able to decode successfully.

Each erasure encoded part is $1/(f + 1)$ the size of the original message. Since each of the $3f + 1$ servers in S sends a part, the bandwidth overhead is approximately $(3f + 1)(1/f + 1)$, which approaches 3 as f increases to infinity. The overhead is slightly greater than this because each part i carries a digital signature from server i in site S. Therefore, in the worst case, $3f + 1$ signatures must be sent for each outgoing message, compared to one if a single server were sending. In practice, the signature overhead can be amortized over several outgoing messages by packing erasure encoded parts for several messages into a single digitally signed physical message.

The erasure encoding approach also has a higher computational cost than an approach in which a single server sends messages on behalf of the site. The receiving site locally orders the incoming parts as they arrive, meaning that the reception of a message by the logical machine requires the local ordering of up to $3f + 1$ events per message. Section 5 describes implementation optimizations that can be used to mitigate this computational overhead.

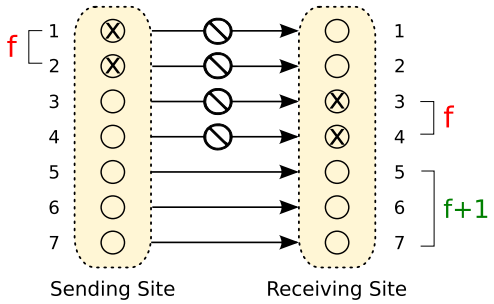

Figure 2: An erasure encoding based logical link is composed of $3f+1$ virtual links; in this example, $f = 2$. The adversary can block at most f virtual links by corrupting servers in the sending site and f virtual links by corrupting servers in the receiving site. Thus, at least $f + 1$ virtual links have two correct endpoints and can freely pass messages.

4.1.1 Blacklisting Servers that Send Invalid Parts

The preceding discussion assumed that erasure encoded parts were generated correctly. However, faulty servers may generate invalid parts in an attempt to disrupt the decoding process. Unlike partial signatures, erasure encoded parts are not individually verifiable: they do not carry proofs that they were created correctly. If a server attempts to decode a message using $f + 1$ parts but obtains an invalid message (i.e., one whose threshold signature does not verify correctly), it cannot, without further information, determine which (if any) of the parts are invalid. There are two possible cases: (1) one or more of the parts is invalid, or (2) all of the parts are valid, but the site that sent the message is faulty and encoded a message with an invalid threshold signature. Even if the server waits for additional parts to arrive, there is no efficient way for it to find a set of $f + 1$ valid parts out of a larger set. Without a mechanism for determining which parts are faulty, malicious servers can repeatedly cause the correct servers to expend computational resources (i.e., by exhaustive search) to determine which parts should be used in the decoding. If the site that sent the message is indeed faulty, then no combination of parts may decode to a valid message.

To overcome these difficulties, we augment the basic erasure encoding scheme with a blacklisting mechanism that can be used to prevent faulty servers from continually causing the message decoding to fail by submitting invalid parts. We employ both site-level and server-level blacklists. When a site is blacklisted, subsequent messages from all servers in the site are ignored. When a server is blacklisted, only messages originating from that server are ignored; messages from non-blacklisted servers in the same site continue to be processed.

In the description that follows, we consider a message being sent between two sites, S and R, where S sends an erasure-encoded message to R that results in a failed decoding. The blacklisting protocol guarantees that:

```
A1. Upon server i in site R executing a failed decoding for message from site S:
A2.   inquirySeq_{R,S}++
A3.   decodedSet ← set of f + 1 parts used in failed decoding
A4.   erasureSeq_{S,R} ← sequence number of message in question (generated by S)
A5.   unsignedInquiry ← ⟨INQUIRY, inquirySeq_{R,S}, decodedSet, erasureSeq_{S,R}, R⟩
A6.   Invoke THRESHOLD-SIGN(unsignedInquiry, i)
A7.   Stop handling messages from S except next expected INQUIRY and INQUIRY-RESPONSE

B1. Upon THRESHOLD-SIGN returning signedInquiry at server i in site R:
B2.   Send to server i in site S: signedInquiry

C1. Upon server i in site S executing ⟨INQUIRY, inquirySeq_{R,S}, decodedSet, erasureSeq_{S,R}, R⟩:
C2.   if all parts in decodedSet are valid:
C3.      SiteBlacklist ← SiteBlacklist ∪ {R}
C4.   else
C5.      invalidSet ← identifiers of local servers whose parts were invalid
C6.      fullMessage ← original message encoded with sequence number erasureSeq_{S,R}
C7.      unsignedInquiryResponse ← ⟨INQUIRY-RESPONSE, inquirySeq_{R,S}, erasureSeq_{S,R}, fullMessage, S⟩
C8.      Invoke THRESHOLD-SIGN(unsignedInquiryResponse, i)
C9.      ServerBlacklist[S] ← ServerBlacklist[S] ∪ invalidSet

D1. Upon THRESHOLD-SIGN returning signedInquiryResponse at server i in site S:
D2.   Send to server i in site R: signedInquiryResponse

E1. Upon server i in site R executing ⟨INQUIRY-RESPONSE, inquirySeq_{R,S}, erasureSeq_{S,R}, fullMessage, S⟩:
E2.   expectedSet ← computed parts from fullMessage
E3.   if all parts from expectedSet match parts in decodedSet
E4.      SiteBlacklist ← SiteBlacklist ∪ {S}
E5.   else
E6.      invalidSet ← identifiers of servers from S whose parts were invalid in decodedSet
E7.      ServerBlacklist[S] ← ServerBlacklist[S] ∪ invalidSet
E8.      if |ServerBlacklist[S]| > f
E9.         SiteBlacklist ← SiteBlacklist ∪ {S}
E10.     else
E11.        Resume executing messages from site S
```

Figure 3: Blacklisting Protocol.

- If both S and R are correct, then a server in S will be proven faulty and subsequently blacklisted after generating just one invalid erasure encoded part; from then on, the server will not be able to disrupt the decoding at any receiving site.

- If S is faulty, then each faulty server in S can disrupt the decoding at most once in each receiving site before it is blacklisted by that site. If S fails to take part in the blacklisting protocol, messages from all of its servers will be ignored, except for those messages that would implicate either S as a whole or one or more faulty servers.

The intuition behind the blacklisting protocol is that a server in site R can deduce which party is at fault when a decoding fails (i.e., one or more servers in S or site S as a whole) if it has access to the original message that was encoded. The server can generate the correct parts and compare them to the parts it received and used in the decoding. There are two possible cases. If all of the parts are correct, then at least $f + 1$ servers in site S encoded a message with an invalid threshold signature. Since a correct server only encodes a message if it has a valid threshold signature, this indicates that site S is faulty. If one or more parts are invalid, then because each part is digitally signed by a server in S, the server in R can determine exactly which servers in S submitted the invalid parts and blacklist them.

Pseudocode for the blacklisting protocol is presented in Figure 3. When a server i in site R executes a failed decoding on a message sent from site S, it attempts to generate a threshold signature on an $\langle$INQUIRY, inquirySeq$_{R,S}$, decodedSet, erasureSeq$_{S,R}$, R$\rangle$ message, where inquirySeq$_{R,S}$ is a sequence number incremented each time site R sends an INQUIRY message to site S, decodedSet is the set of $(f + 1)$ parts that were used in the failed decoding, and erasureSeq$_{S,R}$ is the sequence number assigned by site S to the erasure encoded message for which the decoding failed. In addition, the server stops handling all messages from S, except for the next expected INQUIRY message or the INQUIRY-RESPONSE corresponding to the current inquiry (see below). When server i in site R generates the threshold signature for the INQUIRY message, it sends the message to server i in site S (Figure 3, Block B). Note that this message is not erasure encoded, preventing a circular dependency that could occur if the INQUIRY message itself were not properly encoded (potentially causing an inquiry for the INQUIRY message).

When the servers in S locally execute site R's INQUIRY message, they first examine the set of encoded parts to determine if any of the parts are actually invalid. If none of the parts is invalid, then site R is faulty, and site S blacklists R and stops all communication with it (Figure 3, lines C2-C3). This prevents faulty sites from generating spurious INQUIRY messages. If one or more parts are invalid, then site S generates an INQUIRY-RESPONSE message, which contains the full message

that was originally encoded. The combination of the INQUIRY message and its INQUIRY-RESPONSE proves that one or more of the servers in S is faulty; therefore, servers in S can present this proof to an administrator, who can shut down the faulty servers to prevent them from continuing to send invalid parts. Note that if site S is faulty, it may never generate an INQUIRY-RESPONSE message at all. Although site R will not be able to blacklist any servers from S in this case, R will only handle the next expected INQUIRY or INQUIRY-RESPONSE; all other messages will be dropped before being locally ordered.

Upon locally executing the INQUIRY-RESPONSE message from site S, the servers in site R use the full message to determine which of the decoded parts were invalid. If none of the parts is invalid, then site S is faulty and can be blacklisted (Figure 3, lines E3-E4). This prevents faulty sites from generating spurious INQUIRY-RESPONSE messages. Otherwise, site R blacklists those servers whose parts were invalid and resumes handling messages from site S. If the number of servers blacklisted from site S exceeds f, then site S is known to be faulty and can be blacklisted as a whole.

We impose one additional constraint on the processing of an INQUIRY message to prevent servers in a faulty receiving site from wasting the resources of correct servers in a correct sending site. Suppose site S is correct but has a faulty server p that has sent invalid parts for multiple messages, and suppose site R is faulty. Site R may generate multiple INQUIRY messages, each implying that p is faulty. This causes S to use up resources unnecessarily in order to generate INQUIRY-RESPONSE messages. For this reason, site S will only respond to an INQUIRY message if (1) it is for the next expected inquiry sequence number from R, and (2) it implicates a new faulty server. A correct site will not send an INQUIRY message with inquiry sequence number $(i + 1)$ until it has processed an INQUIRY-RESPONSE message for sequence number i. Therefore, if site S receives two INQUIRY messages that ultimately implicate the same faulty server, then site R is faulty and can be blacklisted.

4.2 Hub Based Logical Link

In this section we describe how we can improve upon the basic erasure encoding scheme presented in Section 4.1 by placing the servers within a site on a broadcast Ethernet hub.[3] Figure 4 shows the network configuration within and between two wide-area sites when the hub based logical link is deployed. The servers in each site have two network interfaces. The first interface connects each server to a LAN switch and is used for intra-site communication. The second interface connects each server to a site hub and is used for sending and receiving wide-area messages. This interface is configured to operate in promiscuous mode so that the server receives a copy of any message passing through the hub.

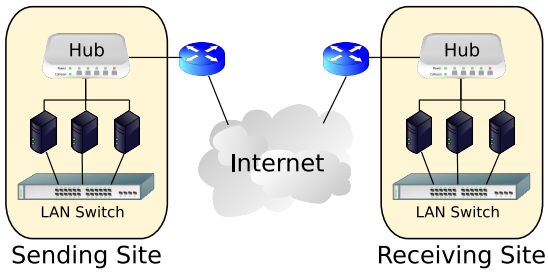

Figure 4: Network configuration of the hub based logical link.

The hub based implementation of the logical link exploits the following two properties of a broadcast hub:

Uniform Reception: Any incoming wide-area message received by one local server will be received by all other local servers.

Overhearing: Any outgoing wide-area message sent by a local server will be received by all local servers.

When integrated with the basic erasure encoding scheme, a broadcast hub yields several benefits. The Uniform Reception property implies that as long as the physical machine that sends an erasure encoded part is correct, all of the correct physical machines in the receiving site will receive the part. This means that any virtual link whose sender is correct will behave like a correct virtual link, even if the peer is faulty, provided at least one correct physical machine in the receiving site assumes responsibility for introducing the part for local ordering. Since there are at least $2f+1$ correct servers in the sending site, a threshold-signed message can be encoded into $2f + 1$ message parts and f redundant parts, where each part is $(1/(2f + 1))$ the size of the original message. This improves the worst-case overhead to approximately $(3f + 1)(1/(2f + 1))$, which approaches an overhead factor of 1.5 as f tends towards infinity, compared to an overhead factor of 3 with the basic erasure encoding scheme.

The Overhearing property enables local servers to monitor which erasure encoded parts were already sent through the hub; if enough parts were already sent, a local server need not send its own part, saving wide-area bandwidth. Of course, some of the parts that the server overhears on the hub may be faulty, and so the blacklisting protocol described in Section 4.1.1 remains a critical component of the logical link.

[3] Some newer devices are called "hubs" but actually perform learning by examining source MAC addresses to map addresses to ports, subsequently forwarding frames only to their intended destination. We explicitly refer to broadcast hubs that do not employ this optimization.

To leverage the Overhearing property, we map each outgoing message to two disjoint sets, the first with $2f + 1$ members and the second with f members. When a server encodes an outgoing message, it decides to send its part based on which set it is in. If the server is in the first set, then it sends its erasure-encoded part to its peer immediately. If the server is in the second set, then it schedules the sending of its part after a local timeout period. If, before the timeout expires, the server overhears $2f + 1$ parts on the hub from the encoding of the current message, then the server cancels transmission of its part. If the timeout expires, the server sends the part to its peer. When all of the members of the first set are correct and the timeout values are set correctly, exactly $2f + 1$ parts will be sent, each $(1/(2f + 1))$ the size of the message, which is nearly optimal; as before, the overhead is slightly higher due to the signature overhead on each part. In the worst case, all of the parts will be sent, yielding an overhead factor approaching 1.5. The overhead in practice will depend on the number of faulty servers and how well the site's timing assumptions hold.

There are two potential costs of deploying the hub based logical link: local computation and bandwidth, and latency. Since incoming wide-area messages are received on the hub, many servers in the receiving site will receive a copy of each erasure encoded part. This raises the question of which server in the receiving site should be responsible for introducing a part for local ordering. The approach we take is to assign a set of $f + 1$ servers to each incoming part, ensuring that at least one correct server will introduce each part for ordering. Duplicate copies of a part are ignored upon local execution. Thus, while the hub improves wide-area bandwidth efficiency, it increases local computation and bandwidth usage in the receiving site because it requires more events to be locally ordered. We believe this tradeoff is desirable in wide-area systems, whose performance is usually limited by wide-area bandwidth constraints.

The other potential cost of the hub based logical link is higher latency compared to the basic erasure encoding scheme. If any of the $2f + 1$ servers in the first group do not send their parts when they are supposed to, then the servers in the second set will wait a local timeout period before transmitting their parts. In the worst case, this timeout is incurred in each round of the wide-area protocol. A system administrator whose focus is on minimizing latency may opt to configure the system so that all servers send their parts immediately, reducing delay under attack but paying a higher cost in wide-area bandwidth (yielding a fixed overhead approaching 1.5).

Finally, we note that while broadcast hubs are a natural fit for our architecture, they are somewhat dated pieces of hardware that are often replaced in favor of switches. Our system can achieve the same benefit as a hub by using any device meeting the Uniform Reception and Overhearing properties, such as network taps.

4.3 Dependable Forwarder Based Logical Link

We now consider the implications of equipping each site with a *dependable forwarder* (DF), a dedicated device that sits between the servers in a site and the wide-area network and is responsible for sending and receiving wide-area messages on the site's behalf. The basic premise is as follows. When the physical machines in a site generate a threshold-signed message, they send it to the site's dependable forwarder. When the DF receives $f + 1$ copies of the message, it sends exactly one copy of the message to the DF at each destination site. Upon receiving an incoming wide-area message, a DF disseminates it to the physical machines in the local site.

We designed the dependable forwarder to be neutral to the wide-area replication protocol being deployed. This makes it simpler to implement and reason about (by avoiding protocol-specific configuration and dependencies), as well as more generally applicable. Each local server communicates with the local DF via TCP, tagging each message with a message authentication code (MAC). The DFs send messages to each other using UDP, just as the servers would if they were communicating directly. Messages sent between DFs contain MACs.

After generating a threshold-signed wide-area message, a local server simply sends it to the DF, prepending a short header that contains (1) a sequence number, (2) a destination bitmap, and (3) the message length. The sequence number is a 64-bit integer incremented each time the server wants to send a wide-area message; since local servers generate wide-area messages in the same order, they will consistently assign sequence numbers to outgoing messages. The destination bitmap is a short bit string used to indicate to which sites the message should be sent. The header is stripped off before the DF sends the message on the wide-area network. Note that the DF does not need to verify threshold signatures or know anything about the content of the wide-area messages.

Since it is depended upon to be available, the DF should be deployed using best practices, including protecting it from tampering via physical security and access control, and configuring it to run only necessary services to reduce its vulnerability to software-based compromise. A primary-backup approach can also be used to fail-over to a backup DF in case the primary DF crashes.

As stated in Section 2, any number of dependable forwarders can be compromised without threatening the consistency of the global replication service. Thus, we rely on the DFs to run correct code and remain available, but not at the risk of making it easier to violate safety. A site whose DF has been compromised but in which f or fewer servers have been compromised can

only exhibit faults in the time and performance domains, *not* in the value domain. The reason this property holds is that the DF passes threshold-signed messages, which even a compromised DF cannot forge. We believe relying on DFs whose compromise cannot cause inconsistency, rather than on devices the system requires to be impenetrable in order to guarantee safety, is the correct approach given the strong consistency semantics required by systems that use a state machine replication service. Systems with weaker consistency requirements might relax this constraint to gain efficiency.

In order to justify the fact that system liveness and performance is placed in the hands of the dependable forwarders, it is important that their implementation be simple and straightforward so that the code can be verified for correctness. We now describe such an implementation. Each DF maintains several counters. First, the DF maintains a single counter, *lastSent*, which stores the sequence number of the last message sent on behalf of the site. The DF also maintains one counter per local server, lastReceived$_i$, which stores the sequence number of the last message received from server i. To keep track of which messages (and how many copies of them) have been received from local servers, the DF uses a two-level hash table. The first level maps message sequence numbers into a second hash table, which maps the entire message (including the prepended header) to a *slot* data structure. The slot contains a single copy of the message (stored the first time the message is received) as well as a tally of the number of copies that have been received.

Local Message Handling Protocol: Each DF is configured with a parameter, LOCAL-THRESHOLD, indicating how many copies of a message must be received from local servers before the message should be sent on the wide area. This value can be set between $f + 1$ and $2f + 1$ (inclusive). Setting LOCAL-THRESHOLD to $f + 1$ ensures that at least one correct server wants to send a message with the given content, while setting LOCAL-THRESHOLD to $2f + 1$ ensures that a majority of the correct servers want to send the given message.

The DF must be designed to use a bounded amount of memory so that faulty local servers cannot cause it to run out of resources. The DF expects to receive messages from each local server in sequence number order. A WINDOW parameter dictates how many messages above *lastSent* the DF will accept from a local server before it (temporarily) stops reading from the corresponding session, which will eventually cause the session to block until enough servers catch up and more messages can be sent (i.e., until *lastSent* increases). This guarantees that at most WINDOW slots will be allocated at any point in time.

Remote Message Handling Protocol: A strategy similar to the one described above must be used to bound the amount of resources needed by the dependable forwarder to handle messages from remote sites. The DF maintains a queue per incoming wide-area link; each queue has a bounded size. Incoming messages are placed in the appropriate queue and must be delivered to the servers in the local site; an incoming message is discarded if the corresponding queue is full. Since faulty local servers may fail to read the messages sent by the dependable forwarder, bounding the memory requirements of the DF implies that the DF must be able to "forget" about a message (i.e., perform garbage collection) before it has successfully sent it to all local servers. The DF can be configured to perform garbage collection when it has successfully written the message to between $f + 1$ and $2f+1$ local servers, depending on the requirements of the replication protocol; the former guarantees that at least one correct local server will receive the message, while the latter guarantees that a majority of correct servers will receive the message. Prime works correctly as long as one correct server receives the message, so we set the parameter to $f + 1$.

5 Putting It All Together

In this section we show how the pieces of our architecture fit together and describe the service properties provided by the resulting system. Figure 5 depicts the internal organization of a replication server. As mentioned in Section 3, although the architecture supports the deployment of different local and global replication protocols, we chose to use Prime in both levels of the hierarchy (denoted Local Prime and Global Prime in the figure).

When a Global Prime message arrives on the network (Fig. 5, bottom left), it is dispatched to Local Prime so that it can be locally ordered. Once the message has been locally executed, it is dispatched to Global Prime for processing by the logical machine. If the erasure encoded or hub based logical link is deployed, then the locally executed erasure-encoded parts are passed to the Erasure Code Services module so that they can be decoded when enough parts have been collected. The decoded event is then passed to Global Prime for processing by the logical machine. When the server generates a threshold-signed message, the message is passed to the Logical Link (Fig. 5, bottom right) so that it can be sent on the wide area.

To amortize the computational overhead of generating digital and threshold signatures, each server makes use of a Merkle Tree [21], a cryptographic data structure that can be used to sign multiple messages at once. Our previous work [6] also employed Merkle trees but only for wide-area messages; we use it here for both local and global protocol messages. Using a Merkle tree to threshold-sign wide-area messages actually increases their size slightly, because a logarithmic number of di-

Technique	Bandwidth Overhead		Local Orderings Per Message		Delay Per Message	
	Best Case	Worst Case	Minimum	Maximum	Best Case	Worst Case
Erasure Codes	$\frac{3f+1}{f+1}$	$\frac{3f+1}{f+1}$	$3f+1$	$3f+1$	None	None
Hub Optimistic, $(2f+1, 3f+1)$	1	$\frac{3f+1}{2f+1}$	$(f+1)(2f+1)$	$(f+1)(3f+1)$	None	Local Timeout
Hub Immediate, $(2f+1, 3f+1)$	$\frac{3f+1}{2f+1}$	$\frac{3f+1}{2f+1}$	$(f+1)(3f+1)$	$(f+1)(3f+1)$	None	None
Dependable Forwarder	1	1	$f+1$	$f+1$	None	None

Table 1: Summary of Inter-site Communication Techniques. In the Hub-Optimistic($2f+1$, $3f+1$) approach, a message is encoded into $3f+1$ parts, $2f+1$ of which are required to decode. $2f+1$ parts are sent immediately, and the remaining f parts may be sent after a local timeout. Hub-Immediate($2f+1$, $3f+1$) is the same as Hub-Optimistic, except that all $3f+1$ parts are sent immediately.

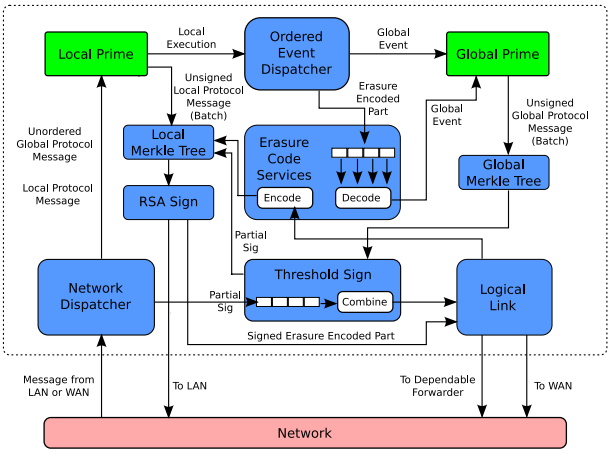

Figure 5: Internal server organization.

gests must be appended to enable signature verification. The ability to aggregate signatures is what makes the logical machine throughput high enough so that the system is bandwidth-constrained, rather than CPU constrained. Thus, it is worth paying the cost in digests to achieve much higher system throughputs.

5.1 Liveness and Performance Properties

We now present the liveness and performance properties provided by our system. We first consider the performance characteristics of correct logical machines and then describe the system-wide performance guarantee.

The local instance of Prime guarantees a property called *Bounded-Delay* (originally defined in [7] but restated below). We begin by specifying the local network stability requirements needed to meet this property. We define two classes of network traffic: *timely* and *bounded*. Messages in the bounded traffic class are assumed to arrive in some unknown bounded time. This is the degree of synchrony commonly assumed in Byzantine fault-tolerant replication systems (e.g., [15, 18]). A small subset of messages (the timely messages) require a stronger degree of synchrony:

ASSUMPTION 5.1 LOCAL-PRIME-STABILITY*: There is a time after which the following condition holds for a set of at least* $2f+1$ *correct servers in the site. This set of servers is called the* stable servers.

- *For each pair of stable servers* s *and* r*, there exists a value Min_Lat(s, r), unknown to the servers, such that if* s *sends a timely message to* r*, it will arrive with delay* $\Delta_{s,r}$*, where Min_Lat(s, r)* $\leq \Delta_{s,r} \leq$ *Min_Lat(s, r)* $* K_{Local}$.

In other words, the ratio of the maximum to the minimum message delay for any timely message sent from s to r is no more than K_{Local}, a known network-specific constant accounting for latency variability. We believe LOCAL-PRIME-STABILITY can be made to hold in well-provisioned local-area networks, where latency is often predictable and bandwidth is plentiful. In addition, timely messages can be processed with higher priority to give the assumption better coverage.

When local bounded messages arrive in bounded time and Assumption 5.1 holds, the local instance of Prime makes the following performance guarantee:

DEFINITION 5.1 LOCAL-BOUNDED-DELAY*: There exists a time after which the latency for any operation introduced by a stable server is upper-bounded.*

In Prime, the upper bound is a function of the network roundtrip times (including processing delays) between correct servers, as long as the system is not saturated. Because the number of messages that need to be ordered by the logical machine is limited by the wide-area bandwidth, a well-engineered logical machine is likely to be capable of doing much more processing than it needs to do and is unlikely to become overloaded. [4]

Thus, sites in which LOCAL-PRIME-STABILITY holds and local bounded messages arrive in bounded time will eventually be able to process global protocol events within a bounded time. This is the behavior that one would expect from a single physical machine running Prime. Therefore, we can achieve a performance guarantee analogous to LOCAL-BOUNDED-DELAY at the global level by making an identical network stability assumption to Assumption 5.1, except that servers are replaced

[4] Indeed, in our own tests, performance was limited by wide-area bandwidth rather than the processing capability of the logical machine.

with sites and we use a different variability constant (i.e., K_{Global} instead of K_{Local}):

ASSUMPTION 5.2 GLOBAL-PRIME-STABILITY*: There is a time after which the following condition holds for a set of at least $2F + 1$ correct sites. This set of sites is called the* stable sites*.*

- *For each pair of stable sites S and R, there exists a value Min_Lat(S, R), unknown to servers in the sites, such that if a server in S sends a timely message to a server in R, the message will arrive with delay $\Delta_{S,R}$, where Min_Lat(S, R) $\leq \Delta_{S,R} \leq$ Min_Lat(S, R) $* K_{Global}$.*

Since GLOBAL-PRIME-STABILITY requires a relatively strong degree of timeliness, it is important to justify how it can be made to hold in practical wide-area networks. The messages requiring timeliness all have small bounded size and are only sent periodically. In practice, the system can be tuned so that the timely messages consume a small, fixed amount of bandwidth. The bounded messages (which account for almost all of the traffic) will consume any "extra" bandwidth not used for performance-critical protocol steps. To realize this separation, one can use a quality of service mechanism such as DiffServ [10], with one low-volume class for timely messages and another class for bounded messages.

Note that achieving the necessary network synchrony is not enough to meet GLOBAL-PRIME-STABILITY. The local state machine replication protocol and the logical link protocol also must not introduce unbounded delay. Fortunately, running Prime produces a sufficient degree of timeliness from the logical machine: when LOCAL-PRIME-STABILITY holds, the logical machine provides LOCAL-BOUNDED-DELAY. All three logical link protocols also provide sufficient timeliness. In the erasure encoding and dependable forwarder based logical links, the faulty servers cannot delay a message from being sent on time. In the hub based logical link, the faulty servers can only introduce a small, bounded amount of delay into the link (i.e., the value of the local timeout). Therefore, our protocols supply sufficient timeliness to achieve GLOBAL-PRIME-STABILITY.

When GLOBAL-PRIME-STABILITY holds and bounded messages arrive within a bounded time, the system makes the following performance guarantee:

DEFINITION 5.2 GLOBAL-BOUNDED-DELAY*: There exists a time after which the latency between a stable server in a stable site receiving a client request and all stable servers in all stable sites executing that request is upper-bounded.*

The system requires weaker synchrony conditions for liveness. The following liveness guarantee is met as long as GLOBAL-PRIME-STABILITY holds, even if global bounded messages arrive completely asynchronously:

DEFINITION 5.3 GLOBAL-LIVENESS*: If a stable server in a stable site receives a client request, then all stable servers in all stable sites eventually execute the request.*

6 Performance Evaluation

In this section we evaluate a prototype implementation of our attack-resilient architecture, focusing on the performance implications of deploying the logical link protocols described in Section 4.

6.1 Testbed and Network Setup

We performed our experiments on a cluster of twenty 3.2 GHz, 64-bit Intel Xeon computers. We emulated a wide-area system consisting of 7 sites, each with 7 servers. Such a system can tolerate the complete compromise of 2 sites and can tolerate 2 Byzantine faults in each of the other 5 sites. We ran one fully deployed site on 7 machines (with one server per machine) and emulated the other 6 wide-area sites using one machine per site. The remaining machines were used to run client processes and to emulate the wide-area topology. We used the Spines [4] messaging system to place bandwidth and latency constraints on the links between sites. We limited the aggregate outgoing bandwidth from each site to 10Mbps and placed 50ms delay between wide-area sites. No constraints were placed on the links between the servers in the fully-deployed site (which communicated via a Gigabit switch) or between clients and their local servers. Clients submit one update operation (containing 200 bytes of data, representative of a typical SQL query) and wait for proof that the operation was ordered before submitting their next operation. Clients were distributed as evenly as possible among the sites.

The emulated sites emulate the local ordering of wide-area protocol events based on the ordering delays measured in the non-emulated site. The wide-area messages generated by the emulated sites are exactly the same as if the sites were not emulated, except that they are not threshold signed; the messages contain 128 filler bytes to emulate the bandwidth cost of a signature, and the emulated sites busy-waited for the time required to generate partial signatures and combine them in order to emulate the computational overhead.

We used OpenSSL [2] for generating and verifying RSA signatures and for computing message digests. We used the OpenTC implementation [3] of Shoup's threshold RSA signature scheme for generating threshold signatures. We used Luby's implementation of the Cauchy-based Reed-Solomon erasure encoding scheme [1,11,24] for performing erasure coding operations.

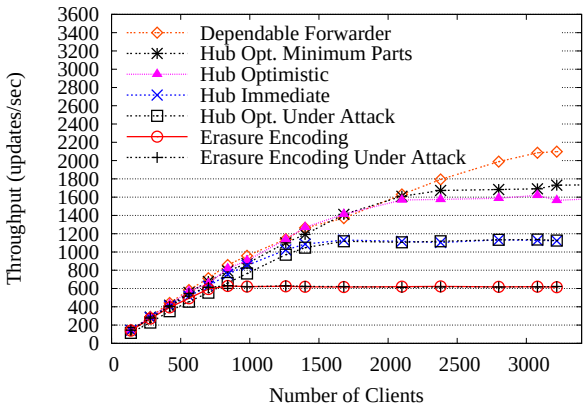

Figure 6: Throughput (updates/sec) vs. number of clients, 50ms Diameter, 10Mbps Links

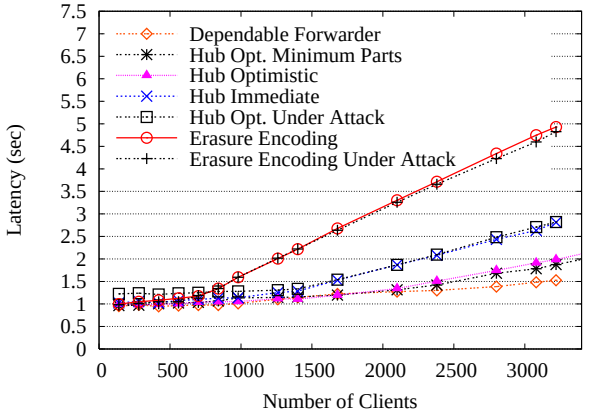

Figure 7: Latency (s) vs. number of clients, 50ms Diameter, 10Mbps Links

6.2 Test Configurations

Erasure Encoded Logical Link: In the erasure encoded logical link, the servers encode threshold-signed messages into 7 parts, and each server sends a part to its peer in the receiving site. The emulated sites send and receive all erasure encoded parts on behalf of the servers they emulate. To evaluate the performance of the logical link under attack, the faulty servers delayed sending their erasure encoded parts by 300 ms.

Hub Based Logical Link: We emulated the use of a hub by having servers (1) locally broadcast outgoing wide-area messages before sending them and (2) locally broadcast incoming wide-area messages before processing them. Servers were assigned to either the first or second sending group based on their server identifiers and sequence numbers contained in the messages. We used a similar strategy to assign the responsibility of proposing incoming messages for local ordering to 3 servers.

We tested the hub based logical link in four configurations. The first is designated as Hub-Optimistic. Wide-area messages are encoded into 7 parts, 5 of which are needed to decode. 5 servers send their parts immediately, and the other 2 only send their parts if they do not overhear enough parts before their local timeout expires. All servers were assumed to be correct. Servers in the second group used a local timeout of 25 ms. This value was chosen after experimentation as one that would not allow faulty servers to cause too much delay when the system is under attack, but which was usually long enough so that correct servers in the second group would not have to send their parts. We observed correct servers to send their parts between 0% and 10% of the time. Emulated sites conservatively sent additional parts 10% of the time.

In the second configuration, Hub-Immediate, all servers were correct and sent their parts immediately. Thus, this configuration does not utilitize the monitoring of outgoing wide-area messages. In the third configuration, we ran an attack on the Hub-Optimistic logical link. Faulty servers in the first group delayed sending their parts by 100 ms, causing correct servers in the second group to have to send their parts because their local timeouts expired. Finally, we tested the performance in a hypothetical scenario when all servers are assumed to be correct and the timeout is set perfectly, so that extra parts are never sent. This configuration is denoted Hub-Optimistic-Minimum-Parts.

Dependable Forwarder Based Logical Link: We emulated the wide-area message patterns of a DF by having one chosen server send and receive wide-area messages on behalf of the site. As above, $f + 1$ servers propose incoming messages for local ordering based on their server identifiers and the message sequence numbers.

6.3 Evaluation

Figure 6 shows system throughput, measured in update operations per second, as a function of the number of clients. Figure 7 shows the corresponding latency, measured in seconds. As expected, the dependable forwarder deployment achieves the best performance, becoming bandwidth-constrained at a peak throughput of 2100 updates/sec. Latency remains relatively stable and is below 1.5 seconds with 3000 clients. Hub-Optimistic-Minimum-Parts and Hub-Optimistic achieve the next best performance, reaching peak throughputs at 1730 and 1600 updates/sec, respectively. Hub-Optimistic-Minimum-Parts demonstrates how the hub based logical link performs with no faults and a perfect timeout. Since the emulated sites in Hub-Optimistic acted conservatively and sent an extra part (beyond the required 5) with 10% probability, a more accurate emulation would bring its performance slightly closer to Hub-Optimistic-Minimum-Parts. The difference between Hub-Optimistic-Minimum-Parts and the dependable forwarder configuration is due to the bandwidth overhead for digital signatures. An average of roughly 2.5 encoded parts were packed into each physical message; more aggressive packing would further reduce the overhead.

Figures 8 and 9 show the performance of the hub con-

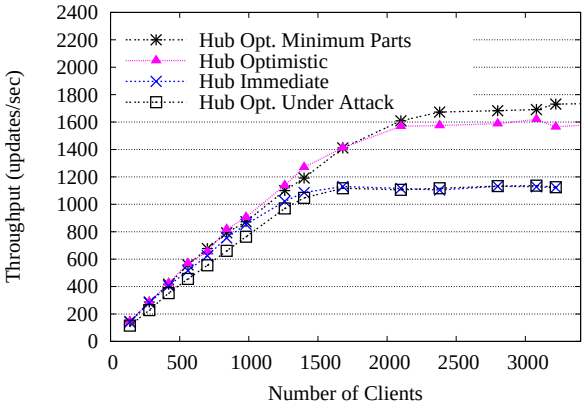

Figure 8: Throughput (updates/sec) vs. number of clients, 50ms Diameter, 10Mbps Links, Hub Configurations

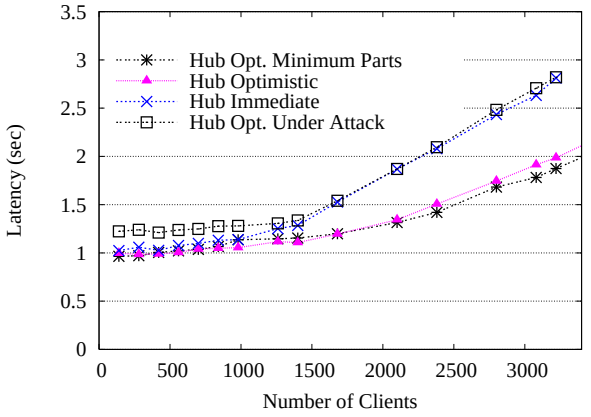

Figure 9: Latency (s) vs. number of clients, 50ms Diameter, 10Mbps Links, Hub Configurations

figurations in isolation so that the effects can be seen more clearly. The Hub-Immediate and Hub-Optimistic-Under-Attack configurations achieved a bandwidth-constrained throughput plateau at 1120 updates/sec. We expected these two configurations to reach the same peak throughput because all servers send a part for each message in both configurations, thus consuming the same amount of outgoing bandwidth. Note that Hub-Optimistic-Under-Attack has a slightly lower slope than Hub-Immediate, reflecting the additional latency incurred by a local timeout per wide-area round. The effect can be seen in Figure 9, as the latency in the attack scenario is between 150 and 200ms higher than in Hub-Immediate until the curves meet when the system becomes saturated. Using a higher local timeout value would increase the peak throughput of Hub-Optimistic slightly, but it would also create additional latency and decrease the slope of the Hub-Optimistic-Under-Attack curve. This reflects the tradeoff between obtaining better fault-free performance and making the protocol more vulnerable to performance degradation under attack.

Finally, the erasure encoded logical link configurations obtained bandwidth-constrained peak throughputs at around 620 updates/sec. As expected, the attack on the erasure encoded logical link had almost no impact on performance. The fact that faulty servers delay the sending of their parts does not prevent 5 correct parts (only 3 of which are needed to decode) from being sent to the receiving site in a timely manner. In fact, the under-attack performance is slightly higher because a larger percentage of the site's outgoing bandwidth is dedicated to parts from correct servers.

Discussion: Our results demonstrate two main points. First, the logical links are effective in mitigating performance attacks on the hierarchical architecture's inter-site communication, while still allowing reasonable fault-free and under-attack performance by using wide-area bandwidth efficiently. Second, making slightly stronger assumptions about the resources available for building a logical link can significantly improve performance. A simple broadcast hub can yield fault-free performance close to the performance achieved when a dependable forwarder sends parts on behalf of the site. Even when under attack, the peak throughput of the hub based logical link only degrades by between 30 and 40 percent, while resulting in a relatively small increase in latency.

Attacks on a flat deployment of Prime (whose effects were shown in [7]) can be mounted against both levels of the hierarchy. In one attack, a faulty leader can add two message delays, plus an aggregation delay. In another attack, the faulty servers can cause the correct servers to consume bandwidth for message reconciliation (i.e., to bring each other up to date). When the delay attack is mounted in the local site, the logical machine processing time increases by a delay whose duration is dominated by the aggregation constant (30ms in our implementation). Since local bandwidth is plentiful, the reconciliation attacks do not have a significant impact on performance within the local site. The same attacks can be mounted on the wide area and have an impact similar to when they are mounted against physical machines. The attacks can decrease throughput by approximately a factor of 2 and can increase update latency by two wide-area message delays plus an aggregation constant (roughly 200ms in our implementation).

7 Related Work

Attack-Resilient State Machine Replication: Recent work has focused on protocols that can perform well even in uncivil executions. Aiyer et al. [5] suggested rotating the primary to mitigate its attacks. The Prime system [7] formalized the need for more performance-oriented correctness criteria to augment traditional liveness properties. The Aardvark system of Clement et al. [15] proposed building robust Byzantine fault-tolerant

systems that sacrifice some normal-case performance to guarantee acceptable performance when under attack. Aardvark incorporates important system engineering techniques that can be used to improve robustness. Such techniques can also be applied to our Prime-based logical machines. The Spinning protocol of Veronese et al. [32] constantly rotates the primary to reduce the impact of faulty servers. Singh et al. [27] demonstrate how the performance of different protocols can degrade under unfavorable network conditions.

Wide-Area Intrusion-Tolerant Replication: Steward [8] used a hierarchical architecture to scale intrusion-tolerant replication to large, multi-site deployents. The customizable architecture in [6] generalized Steward by using a local state machine replication protocol in each site to cleanly separate the local and global protocols. The use of state machine based logical machines has been well-studied in the literature (e.g., [12, 30]). Our current architecture builds on the customizable architecture, running Prime in both levels of the hierarchy. However, we show how to harden the architecture by building attack-resilient logical links. The ShowByz system of Rodrigues et al. [25] supports a large deployment consisting of many replicated objects. ShowByz adjust the BFT quorum size to decrease the likelihood that the fault assumptions of any replicated group are violated.

Protocols in a Hybrid Failure Model: Veríssimo [29] formalized the notion of a hybrid failure model in which different parts of the system operate under different failure and timing assumptions. Correia et al. [16] developed a wormhole-based intrusion-tolerant state machine replication protocol. The protocol makes use of a Trusted Multicast Ordering (TMO) service that runs between trusted components, collecting hashes of atomically multicast messages. When the TMO receives enough copies of the hash, it assigns an ordering to the message. Our dependable forwarder implementation uses a somewhat similar idea, sending the message over the wide-area network when it receives enough copies, but it can be compromised without violating safety.

Bessani et al. [9] build a protection service for critical infrastructure systems. When a message passes from an unprotected to a protected realm, it must be approved by $f + 1$ replicas to ensure that it conforms to policy. Each replica has access to a trusted component that stores a shared symmetric key. The component will only generate a MAC on a message when it collects $f + 1$ copies from different replicas. The system also uses a hub to allow messages to be received by all replicas without modifying legacy components.

Survivable Spread [28] provides an intrusion-tolerant replication service for wide-area networks where at least one node per site is assumed to be impenetrable. Inter-site communication is handled by trusted forwarders running on secure servers. The RAM system of Mao et al. [19] deploys one server in each site and assumes each server is equipped with a trusted attested append-only memory device that signs outgoing messages, allowing other sites to verify the correctness of the messages' contents and enabling reductions in latency. The EBAWA protocol of Veronese et al. [31] uses a trusted Unique Sequential Identifier Generator to constrain the behavior of faulty servers, allowing fewer wide-area rounds.

8 Conclusions

This paper presented an attack-resilient architecture for large-scale intrusion-tolerant replication. We described three logical link protocols for efficient, attack-resilient inter-site communication, and we considered the practical and theoretical implications of deploying different state machine replication protocols in the hierarchical architecture. Our experimental results showed the performance benefits that can be realized by making slightly stronger assumptions about one's environment, without making it easier for faulty servers to cause inconsistency.

References

[1] Cauchy-based reed solomon codes, http://www.icsi.berkeley.edu/ luby/cauchy.tar.uu.

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

[3] The OpenTC library, http://projects.cerias.purdue.edu/ds2.

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

[5] AIYER, S., ALVISI, L., CLEMENT, A., DAHLIN, M., MARTIN, J.-P., AND PORTH, C. BAR fault tolerance for cooperative services. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)* (2005), ACM, pp. 45–58.

[6] AMIR, Y., COAN, B., KIRSCH, J., AND LANE, J. Customizable fault tolerance for wide-area replication. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS '07)* (Beijing, China, 2007), pp. 66–80.

[7] AMIR, Y., COAN, B., KIRSCH, J., AND LANE, J. Byzantine replication under attack. In *Proceedings of the 38th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '08)* (Anchorage, AK, USA, June 2008), pp. 197–206.

[8] AMIR, Y., DANILOV, C., KIRSCH, J., LANE, J., DOLEV, D., NITA-ROTARU, C., OLSEN, J., AND ZAGE, D. Scaling Byzantine fault-tolerant replication to wide area networks. In *Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN'06)* (Philadelphia, PA, USA, June 2006), pp. 105–114.

[9] BESSANI, A., SOUSA, P., CORREIA, M., NEVES, N. F., AND VERISSIMO, P. The CRUTIAL way of critical infrastructure protection. *IEEE Security and Privacy 6*, 6 (Nov-Dec 2008), 44–51.

[10] BLAKE, S., BLACK, D., CARLSON, M., DAVIES, E., WANG, Z., AND WEISS, W. RFC 2475: An architecture for differentiated services, Dec. 1998.

[11] BLOMER, J., KALFANE, M., KARPINSKI, M., KARP, R., LUBY, M., AND ZUCKERMAN, D. An xor-based erasure-resilient coding scheme. Tech. Rep. TR-95-048, International Computer Science Institute, August 1995.

[12] BRASILEIRO, F. V., EZHILCHELVAN, P. D., SHRIVASTAVA, S. K., SPEIRS, N. A., AND TAO, S. Implementing fail-silent nodes for distributed systems. *IEEE Transactions on Computers 45*, 11 (1996), 1226–1238.

[13] CASTRO, M. *Practical Byzantine Fault Tolerance*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusettsl, 2001.

[14] CASTRO, M., AND LISKOV, B. Practical Byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99)* (1999), pp. 173–186.

[15] CLEMENT, A., WONG, E., ALVISI, L., DAHLIN, M., AND MARCHETTI, M. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation* (Berkeley, CA, USA, 2009), USENIX Association, pp. 153–168.

[16] CORREIA, M., LUNG, L. C., NEVES, N. F., AND VERÍSSIMO, P. Efficient byzantine-resilient reliable multicast on a hybrid failure model. In *Proceedings of the 21st Symposium on Reliable Distributed Systems (SRDS '02)* (Suita, Japan, Oct. 2002), pp. 2–11.

[17] COWLING, J., MYERS, D., LISKOV, B., RODRIGUES, R., AND SHRIRA, L. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)* (Seattle, WA, Nov. 2006), pp. 177–190.

[18] KOTLA, R., ALVISI, L., DAHLIN, M., CLEMENT, A., AND WONG, E. Zyzzyva: speculative Byzantine fault tolerance. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)* (2007), pp. 45–58.

[19] MAO, Y., JUNQUEIRA, F. P., AND MARZULLO, K. Towards low latency state machine replication for uncivil wide-area networks. In *Proceedings of the 5th Workshop on Hot Topics in System Dependability (HotDep '09)* (2009).

[20] MARTIN, J.-P., AND ALVISI, L. Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing 3*, 3 (2006), 202–215.

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

[22] MONIZ, H., NEVES, N. F., CORREIA, M., AND VERÍSSIMO, P. Randomized intrusion-tolerant asynchronous services. In *Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN'06)* (2006), pp. 568–577.

[23] NARASIMHAN, P., KIHLSTROM, K. P., MOSER, L. E., AND MELLIAR-SMITH, P. M. Providing support for survivable CORBA applications with the Immune system. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)* (Austin, TX, USA, 1999), pp. 507–516.

[24] PLANK, J. S., AND XU, L. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications. In *Proceedings of the 5th IEEE International Symposium on Network Computing and Applications (NCA '06)* (2006), pp. 173–180.

[25] RODRIGUES, R., KOUZNETSOV, P., AND BHATTACHARJEE, B. Large-scale Byzantine fault tolerance: Safe but not always live. In *Proceedings of the 3rd Workshop on Hot Topics in System Dependability (HotDep '07)* (2007), p. 17.

[26] SHOUP, V. Practical threshold signatures. *Lecture Notes in Computer Science 1807* (2000), 207–220.

[27] SINGH, A., DAS, T., MANIATIS, P., DRUSCHEL, P., AND ROSCOE, T. Bft protocols under fire. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (2008), pp. 189–204.

[28] THE BOEING COMPANY. Survivable Spread: Algorithms and assurance argument. Tech. Rep. Technical Information Report Number D950-10757-1, July 2003.

[29] VERÍISSIMO, P. Travelling through wormholes: a new look at distributed systems models. *SIGACT News 37*, 1 (2006), 66–81.

[30] VERÍSSIMO, P., NEVES, N., CACHIN, C., PORITZ, J., POWELL, D., DESWARTE, Y., STROUD, R., AND WELCH, I. Intrusion-tolerant middleware: The road to automatic security. *IEEE Security & Privacy 4*, 4 (2006), 54–62.

[31] VERONESE, G. S., CORREIA, M., BESSANI, A. N., AND LUNG, L. C. Highly resilient services for critical infrastructures. In *Proceedings of the Embedded Systems and Communications Security workshop (ESCS '09)* (2009).

[32] VERONESE, G. S., CORREIA, M., BESSANI, A. N., AND LUNG, L. C. Spin one's wheels? Byzantine fault tolerance with a spinning primary. In *Proceedings of the 28th International Symposium on Reliable Distributed Systems (SRDS '09)* (2009).

[33] YIN, J., MARTIN, J.-P., VENKATARAMANI, A., ALVISI, L., AND DAHLIN, M. Separating agreement from execution for Byzantine fault-tolerant services. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)* (Bolton Landing, NY, USA, October 2003), pp. 253–267.