Primary–Backup Protocols: Lower Bounds and Optimal Implementations

Navin Budhiraja* Keith Marzullo* Fred B. Schneider† Sam Toueg‡

Department of Computer Science
Cornell University
Ithaca NY 14853, USA

Abstract

We present a formal specification of primary–backup. We then prove lower bounds on the degree of replication, failover time, and worst-case response time to client requests assuming different failure models. Finally, we outline primary–backup protocols and indicate which of our lower bounds are tight.

Keywords: Fault-tolerance, reliability, availability, primary–backup, lower bounds, optimal protocols.

1 Introduction

One way to implement a fault-tolerant service is by using multiple servers that fail independently. The state of the service is replicated and distributed among these servers, and updates are coordinated so that even when a subset of servers fail, the service remains available.

Such fault-tolerant services have been structured in several ways. One approach is to replicate the service state at all servers and to present each client's requests to all nonfaulty servers in the same order. This service architecture is commonly called *active replication* or *the state machine approach* [Sch90] and has been widely studied from both theoretical and practical viewpoints (*e.g.*, [PSL80,CASD85,JB89]).

Another approach to building replicated services is to designate one server as the *primary* and all the others as *backups*. Clients make requests by sending messages only to the primary. If the primary fails, then a *failover* occurs and one of the backups takes over. This service architecture is commonly called the *primary-backup* or the *primary-copy* approach [AD76] and has been widely used in commercial fault-tolerant systems. However, the approach has not been analyzed as extensively as the state machine approach, and little is known of the costs and tradeoffs, the degree of replication required, or the worst-case response time for various failure models. In this paper, we derive some of these tradeoffs. For example, some primary–backup protocols use more servers than the number of failures to be tolerated [LGG+91]. We are now able to explain this phenomenon by showing that the number of servers needed depends also on the failure model.

With both active replication and the primary-backup approach, the goal is to provide a client with the illusion of a service that is implemented by a single server, despite failures. The key difference between the two approaches is how each masks failures. With active replication, the effects of failures are completely masked by voting and the service implemented does resemble that of a single nonfaulty server. With the primary-backup approach, a request to the service can be lost if it is sent to a faulty primary.[1] Thus, clients can observe the effects of failures. Periods during which requests can be lost, however, are bounded by the length of time that can elapse between failure of the primary and takeover by a backup. Such behavior is an instance of what we call *bofo* (*bounded outage finitely often*).

*Supported by Defense Advanced Research Projects Agency (DoD) under NASA Ames grant number NAG 2–593 and by grants from IBM, Siemens, and Xerox. The views, opinions, and findings contained in this report are those of the authors and should not be construed as an official Department of Defense position, policy, or decision.

†Supported in part by the Office of Naval Research under contract N00014-91-J-1219, the National Science Foundation under Grant No. CCR-8701103, DARPA/NSF Grant No. CCR-9014363, and by a grant from IBM Endicott Programming Laboratory.

‡Supported in part by NSF grants CCR-8901780 and CCR-9102231 and by a grant from IBM Endicott Programming Laboratory.

[1] Of course, the client can subsequently resend a copy of that request to the new primary.

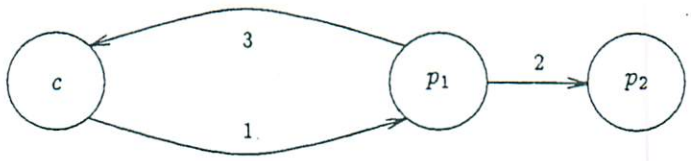

Figure 1: A Simple Primary–Backup Protocol.

For our purposes, a server is a program that repeatedly accepts a request from a client, computes, and sends a reply to the client. The reply can depend on all previous requests that the server has received. Thus, Pb4 states that even though a primary–backup service may suffer outages, any reply it sends must be consistent with all earlier requests to which the server replied.

Property Pb4 is not implementable if the number of failures (that is, number of servers and communication components that fail) can not be *a priori* bounded. This is because an unbounded number of servers would be required. In a real system, one can provide service outages of bounded lengths by bounding the rate of failures and allowing reintegration of recovered servers and communication links. We do not address failure rates or reintegration in this paper.

2.1 A Simple Primary–Backup Protocol

As an example of a service based on the primary–backup approach, consider the following protocol, which tolerates a single server crash. Assume that all communication is over point-to-point nonfaulty links and that each link has an upper bound δ on message delivery time[4]. Refer to Figure 1. There is a primary server p_1 and a backup server p_2 connected by a communications link. A client c initially sends all requests to p_1 (indicated by the arrow labeled 1 in the figure). Whenever p_1 receives a request, it

- processes the request and updates its state accordingly,
- sends information about the update to p_2 (message 2 in the figure),
- without waiting for an acknowledgement from p_2, sends a response to the client (message 3 in the figure).

The order in which these messages are sent is important because it guarantees that, given our assumption about failures, if the client receives a response, then either p_2 has received message 2 or p_2 has crashed.

Server p_2 updates its state upon receiving update messages from p_1. In addition, p_1 sends messages to p_2 every τ seconds. If p_2 does not receive such a message for $\tau + \delta$ seconds, then p_2 becomes the primary. Once p_2 has become the primary, it informs the clients (who update their copies of *Dest*) and begins processing any subsequent requests sent by them.

We now show that this protocol satisfies our characterization of a primary–backup protocol. Property Pb1 requires that there never be two primaries. This is satisfied by the following definitions of *Prmy*:

$$Prmy_{p_1} \stackrel{\text{def}}{=} (p_1 \text{ has not crashed})$$
$$Prmy_{p_2} \stackrel{\text{def}}{=} (p_2 \text{ has not received a message from } p_1 \text{ for } \tau + \delta)$$

The predicate $Prmy_{p_1} \wedge Prmy_{p_2}$ is always false in a system executing our protocol, and hence Pb1 is satisfied. The *failover time* for this protocol is the longest interval during which $\neg Prmy_{p_1} \wedge \neg Prmy_{p_2}$ can hold, and it is $\tau + 2\delta$ seconds. Property Pb2 follows trivially from the description of the protocol. Property Pb3 is true because requests are not sent to p_2 until after p_1 has failed. Finally, Pb4 requires that the protocol implements a single bofo server for some values of k and Δ. Since p_1 sends message 2 before message 3, it will never be the case that p_1 sends a response to the client, and p_2 does not get information about that response from p_1. Using this fact, it can be shown that the service behaves like a single (k, δ)–bofo server. To compute k and Δ, we can let $k = 1$ and so it suffices to compute the

[4]To simplify exposition, we assume that the maximum message delay between the clients and the servers is the same as the delay between the servers. However, our results can be easily extended to the case when the delays are different.

longest interval during which a client request may not elicit a response. Assume that p_1 crashes at time t_c. Any request sent at $t_c - \delta$ or later may be lost since p_1 crashes at t_c. Furthermore, p_2 may not learn about p_1's crash until $t_c + \tau + 2\delta$, and clients may not learn that p_2 is the primary for another δ. So, the total period during which a request may not elicit a response is $t_c - \delta$ through $t_c + \tau + 3\delta$: the service is equivalent to a single $(1, \tau + 4\delta)$–bofo server.

3 The Model

We consider a system consisting of n servers and a set of clients. We assume that server clocks are perfectly synchronized with real time.[5] Clients and servers communicate by exchanging messages through a completely connected point-to-point network.[6] Each message sent is enqueued in a queue maintained by the receiving process, and a process accesses its message queue by executing a **receive** statement. We assume that links between processes are FIFO (*i.e.* if p_i sends message m followed by m' to process p_j, then p_j will never receive m after m') and there is a known constant δ such that if processes p_i and p_j are connected by a (nonfaulty) link, then a message sent from p_i to p_j at time t will be enqueued in p_j's queue at of before $t + \delta$.

We are interested in identifying the costs inherent in primary–backup protocols, and so we assume that it takes no time for a server to compute a response. Our theorems characterize lower bounds, and so they are not invalidated by servers that require a substantial amount of time to compute a response.

We model an execution of a system by a *run*, which is a sequence of timestamped events involving clients, servers, and the message queues. These events include sending messages, enqueuing messages, receiving messages, and internal events that model computation at processes. Two runs σ_1 and σ_2 of the system are defined to be *indistinguishable* to a process p if the same sequence of events (with the same timestamps) occur at p in both σ_1 and σ_2. We assume that if two runs σ_1 and σ_2 are indistinguishable to p and p has the same initial state in both runs, then at any time t the state of p at t in σ_1 is the same as the state of p at t in σ_2. It is not hard to extend the definition of indistinguishability to handle nondeterministic servers; the current definition does not.

We permit clients to send any request at any time. This means that the server cannot depend on the clients, for example, to store the state for the service or to make requests with some fixed period. By making assumptions about the behavior of clients, it is possible to devise protocols that violate our lower bounds.

Define $\prec$ to be the *potential causality* relation [Lam78] on server events e_1 and e_2 as follows: $e_1 \prec e_2$ iff both e_1 and e_2 occur at the same server s and e_1 occurs before e_2, or e_1 is a send event and e_2 is the corresponding receive event, or $(\exists e\colon e_1 \prec e \wedge e \prec e_2)$. Informally, we say a request m is an *update request* if it changes the state of the service in such a way that the responses to subsequent requests depend on m. Consequently, for m an update request with associated response r (and where $e(m)$ and $e(r)$ are the events in the run associated with the receipt of m and the sending of r respectively), all request/response pairs m'/r' where m' was sent after $e(r)$ have $e(m) \prec e(r')$. A primary-backup protocol is trivial to implement if there are no update requests, and so we assume that update requests exist.

We assume that failures occur independently from each other. We consider the following hierarchy of failure models:

Crash failures: A server may fail by halting prematurely. Until it halts, it behaves correctly. After it halts, a timeout can detect this fact.[7]

Crash+Link failures: A server may crash or a link may lose messages (but not delay, duplicate or corrupt messages).

Receive-Omission failures: A server may fail not only by crashing, but also by omitting to receive some of the messages directed over a nonfaulty link.

Send-Omission failures: A server may fail not only by crashing, but also by omitting to send some of the messages over a nonfaulty link.

[5] Extension to the case where clocks are only approximately synchronized [LMS85] is discussed in [Bud93].

[6] Another approach would be assume that servers are interconnected with redundant broadcast busses [BD85,Cri90]. We have not pursued this approach.

[7] The lower bounds we derive for crash failures also hold for fail-stop failures [SS83] except for the bound on failover time. The lower bound on failover time depends on the maximum duration between when a server p_i fails and when $failed_i$ becomes true.

General-Omission failures: A server may exhibit send-omission and receive-omission failures.

Note that crash+link failures and the various types of omission failures are quite different. Although these failure models represent loss of messages, each mode is dealt with by a different masking technique. In particular, crash+link failures can be masked by adding redundant communication paths, while omission failures can only be masked by adding redundant servers so that faulty processes can detect their own failure and halt. We discuss these masking techniques in Section 5.

Failures are counted by the number of failing components (either servers or links). For example, in a protocol that can tolerate one send-omission failure, only one server can commit send omission failures but it can omit to send an arbitrary number of times. We assume that no more than f_s servers can be faulty, and for crash+link failures that no more than f_l links can be faulty.

4 Lower Bounds

For each failure model, we now give lower bounds for implementing a single (k, Δ)–bofo server using the primary–backup approach. The bounds are presented as a set of theorems (without proofs). The appendix describes our proof methods and the proofs can be found in [BMST92b].

4.1 Bounds on Replication

The first bound is obvious.

Theorem 1 *Any primary–backup protocol tolerating f_s crash failures requires $n \geq f_s + 1$.*

Theorems 2 and 3 depend on two parameters of primary–backup protocols. Let Γ be the maximum time that can elapse between any two successive client requests (possibly from different clients), and let D be a duration such that if some server s becomes the primary at time t_0 and remains the primary through time $t \geq t_0 + D$ when a client c_i sends a request, then $Dest_i = s$ at time t. Hence, D is the minimum delay until all clients know of the new primary server. For simplicity of notation, we write $D < \Gamma$ to mean that D is bounded and Γ is either unbounded or bounded and greater than D. Note that when $D < \Gamma$ the service must be able to detect the failure of a primary and disseminate the new primary's identity to the clients without using any messages from clients.

With both send-omission failures and crash+link failures, messages may fail to reach their destinations. The following theorem shows that crash+link failures are more expensive to tolerate, as they require more replication.

Theorem 2 *Suppose there is at most one link between any two servers and up to f failures can occur where each failure may be either a link failure or a server failure. Then any primary–backup protocol tolerating crash+link failures and having $D < \Gamma$ requires $n \geq f + 2$.*

The assumption in this theorem that $D < \Gamma$ is significant. As we discuss in Section 5, when $D \geq \Gamma$ protocols that tolerate f crash+link failures can be constructed that use only $f + 1$ servers.

The next theorem states that additional replication is required in order to tolerate receive-omission failures.

Theorem 3 *Any primary–backup protocol tolerating receive-omission failures and having $D < \Gamma$ requires $n > \lfloor \frac{3f_s}{2} \rfloor$.*

The next lower bound holds independent of the relation between D and Γ.

Theorem 4 *Any primary–backup protocol tolerating general-omission failures requires $n > 2f_s$.*

4.2 Bounds on Blocking

Informally, a *blocking* primary-backup protocol is one in which the primary must, after receiving a request m, either receive a message from another server or simply wait an interval before it can respond to m. Consider a failure-free run of a primary-backup protocol that is handling a request. Let the time that the request is received be t_m and the time that the response is sent be t_r. We say that this protocol is *C–blocking* if it is guaranteed that $t_r - t_m \leq C$ holds. For example, any primary-backup protocol in which the primary sends information about a request to the backups and waits for acknowledgement before sending the response to the client will be at least 2δ–blocking.

As shown in Section 5, 0–blocking primary-backup protocols are possible for crash and crash+link failure models. For servers that take no time to compute the response to a request, the simple protocol tolerating crash failures presented in Section 2 is 0–blocking. We call such protocols *nonblocking* because the primary can send a reply to the client as soon as the reply is computed. Nonblocking protocols tolerating receive-omission failures also exist as long as $n > 2f_s$, but there is can be no nonblocking primary–backup protocol tolerating send-omission failures.

Theorem 5 *Any primary–backup protocol tolerating receive-omission failures with $f_s > 1$, $n \leq 2f_s$ and $D < \Gamma$ is C–blocking for some $C \geq 2\delta$.*

And, if $f_s = 1$, then the following theorem holds:

Theorem 6 *Any primary–backup protocol tolerating receive-omission failures with $f_s = 1$ and $n \leq 2f_s$ and having $D < \Gamma$ is C–blocking for some $C \geq \delta$.*

Primary–backup protocols tolerating send-omission failures exhibit the same blocking properties as those tolerating receive-omission failures:

Theorem 7 *Any primary–backup protocol tolerating send-omission failures and $f_s > 1$ is C–blocking for some $C \geq 2\delta$.*

Theorem 8 *Any primary–backup protocol tolerating send-omission failures and $f_s = 1$ is C–blocking for some $C \geq \delta$.*

4.3 Bounds on Failover Times

Recall that the failover time is the longest interval during which $Prmy_s$ is not true for any server s. In this section, we give lower bounds for failover times.

In order to discuss these bounds, we postulate a fifth property of primary–backup protocols.

Pb5: A correct server that is the primary remains so until there is a failure of *some* server or link.

This is a reasonable expectation and it is valid for all protocols that we have found in the literature.

Theorem 9 *Any primary–backup protocol tolerating f_s crash failures must have a failover time of at least $f_s\delta$.*

The failover times for all other failure models have a larger lower bound:

Theorem 10 *Any primary–backup protocol tolerating f crash+link failures has a failover time of at least $2f\delta$.*

Theorem 11 *Any primary–backup protocol tolerating f_s receive-omission failures has a failover time of at least $2f_s\delta$.*

Theorem 12 *Any primary–backup protocol tolerating f_s send-omission failures has a failover time of at least $2f_s\delta$.*

5 Outline of the Protocols

In order to establish that the bounds given above are tight, we have developed primary–backup protocols for the different failure models [BMST92a]. In this section, we outline these protocols and use them to show which of our lower bounds are tight.

Our protocol for crash failures is similar to the protocol given in Section 2. Whenever the primary receives a request from the client, it processes that request and sends information about state updates to the backups before sending a response to the client. All servers periodically send messages to each other in order to detect server failures. This protocol uses $(f_s + 1)$ servers and so Theorem 1 is tight. Furthermore, it is nonblocking and so incurs no additional delay. It has the failover time $f_s\delta + \tau$ for arbitrarily small and positive τ, and so Theorem 9 is tight.

In order for the protocol to tolerate crash+link failures, we add additional servers (as few as one such servers when $f_l \leq f_s$). By Theorem 2, these servers are necessary. The additional servers ensures that there is always at least one nonfaulty path between any two correct servers, where a path contains

failure model	degree of replication	amount of blocking	failover time
crash	$n > f_s$	0	$f_s\delta$
crash+link	$n > f + 1$ †	0	$2f\delta$
receive omission	$n > \lfloor \frac{3f_s}{2} \rfloor$ * †	$\delta \quad f_s = 1$ † $2\delta \quad f_s > 1$ * †	$2f_s\delta$
send omission	$n > f_s$	$\delta \quad f_s = 1$ $2\delta \quad f_s > 1$	$2f_s\delta$
general omission	$n > 2f_s$	$\delta \quad f_s = 1$ $2\delta \quad f_s > 1$	$2f_s\delta$

* Bound not known to be tight.
† $D < \Gamma$.

Table 1: Lower Bounds.

zero or more intermediate servers. The crash failure protocol outlined above is now modified so that a primary ensures any message sent to a backup is sent across at least one nonfaulty path. This protocol uses $(f+2)$ servers and so Theorem 2 is tight. Furthermore, it is nonblocking and so incurs no additional delay. It has the failover time $2f\delta + \tau$ for arbitrarily small and positive τ, and so Theorem 10 is tight.

Most of our protocols for the various kinds of omission failures apply translation techniques [NT88] to the crash failure protocol outlined above. These techniques ensure that a faulty server detects its own failure and halts, thereby translating a more severe failure to a crash failure. The translations of [NT88] assume a round-based protocol. Since our crash failure protocol is not round-based, we must modify the translations so that a server can send and receive messages at any time rather than just at the beginning or the end of a round. This is not difficult to do. All of these resulting omission–failure protocols have failover time $2f_s\delta + \tau$, and thus Theorems 11 and 12 are tight. The protocol for send-omission failures uses $f_s + 1$ servers and is $2\delta + \tau$–blocking. Furthermore, we also have a send-omission protocol for $f_s = 1$ that is δ–blocking. Thus, Theorems 7, 8 and 12 are tight. The protocol for general-omission failures also uses $2f_s + 1$ servers and is $2\delta + \tau$–blocking, and so Theorem 4 is tight, and Theorems 7 and 12 are tight for general-omission failures as well.

We have not been able to determine whether Theorems 3 and 5 are tight. Our protocol tolerating receive-omission failures uses $2f_s + 1$ servers whereas the lower bound in Theorem 3 only requires $n > \lfloor \frac{3f_s}{2} \rfloor$. We have constructed receive-omission protocols for $n = 2$, $f_s = 1$ and $n = 4$, $f_s = 2$ but have not been able to generalize these protocols. The protocols in this region have the odd property that a nonfaulty primary can cede to a faulty primary, and so we do not expect such protocols to have much practical importance. However, the protocol for $n = 2$, $f_s = 1$ is δ–blocking and so Theorem 6 is tight.

Table 1 summarizes all of our results.

6 Discussion

This paper gives a formal characterization for primary–backup protocols in a system with synchronized clocks and bounded message delays. It presents lower bounds on the degree of replication, the blocking time, and the failover time under various kinds of server and link failures. A set of primary–backup protocols is outlined and shows which of our lower bounds are tight.

It is instructive to compare our results to existing primary–backup protocols. The two-server primary–backup protocol that tolerates crash+link failures presented in [Bar81] seemingly contradicting Theorem 2. However, this protocol assumes that there are two links between the two servers, effectively masking a single link failure. Hence, only crash failures need to be tolerated, and this can be accomplished using only two servers (Theorem 1).

A more ambitious primary–backup protocol is presented in [LGG+91]. This protocol tolerates the following failure model (quoted from [LGG+91]):

> The network may lose or duplicate messages, or deliver them late or out of order; in addition it may partition so that some nodes are temporarily unable to send messages to some other nodes. As is usual in distributed systems, we assume the nodes are fail-stop processors and the network delivers only uncorrupted messages.

This failure model is incomparable with those in the hierarchy we presented. However, the protocol does tolerate general-omission failures and has optimal degree of replication for general-omission failures as it uses $2f_s + 1$ servers.

In Theorem 2, we assumed that $D < \Gamma$. This assumption is crucial: we have constructed a two-server primary–backup protocol tolerating one crash+link failure for which $D \geq \Gamma$. Recall that link failures are masked by adding redundant paths between the servers. Our two-server crash+link protocol essentially uses the path from the primary to the backup through the client as the redundant path. Thus, there appears to be a tradeoff between the degree of replication and the time it takes for a client to learn that there is a new primary.

The lower bounds on failover times given in Section 4.3 assume Pb5. We have constructed primary–backup protocols that have failover times smaller than the lower bounds given in Section 4.3, and as expected these protocols do not satisfy Pb5. This smaller failover time is achieved at a cost of an increased variance in service response time.

Finally, we have attempted to give a characterization of primary–backup that is broad enough to include most synchronous protocols that are considered to be instances of the approach. There are protocols, however, that are incomparable to the class of protocols we analyze [BJ87]. In addition, the protocols in [OL88,MHS89] are incomparable since they were developed for an asynchronous setting. Such protocols cannot be cast in terms of implementing a (k, Δ)–bofo service for finite values of k and Δ. We are currently studying possible characterizations for a primary–backup protocol in an asynchronous system and expect to extend our results to this setting.

Acknowledgements

We would like to thank Mike Reiter and the anonymous conference referees for their comments on earlier drafts of this paper.

References

[AD76] P.A. Alsberg and J.D. Day. A principle for resilient sharing of distributed resources. In *Proceedings of the Second International Conference on Software Engineering*, pages 627–644, October 1976.

[Bar81] J.F. Barlett. A nonstop kernel. In *Proceedings of the Eighth ACM Symposium on Operating System Principles, SIGOPS Operating System Review*, volume 15, pages 22–29, December 1981.

[BD85] Özalp Babaoğlu and Rogério Drummond. Streets of Byzantium: Network architectures for fast reliable broadcasts. *IEEE Transactions on Software Engineering*, 11(6):546–554, June 1985.

[BEM91] Anupam Bhide, E.N. Elnozahy, and Stephen P. Morgan. A highly available network file server. In *USENIX*, pages 199–205, 1991.

[BJ87] Kenneth P. Birman and Thomas A. Joseph. Exploiting virtual synchrony in distributed systems. In *Eleventh ACM Symposium on Operating System Principles*, pages 123–138, November 1987.

[BMST92a] Navin Budhiraja, Keith Marzullo, Fred Schneider, and Sam Toueg. Optimal primary–backup protocols. Technical report, Cornell University, Ithaca, N.Y., 1992.

[BMST92b] Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. Primary–backup protocols: Lower bounds and optimal implementations. Technical Report TR 92-1265, Cornell University, Ithaca, N.Y., January 1992. Revised July 1992.

[Bud93] Navin Budhiraja. *Primary Backup in Synchronous and Asynchronous Systems.* PhD thesis, Cornell University, Department of Computer Science, 1993. In preparation.

[CASD85] Flaviu Cristian, Houtan Aghili, H. Ray Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *Proceedings of the Fifteenth International Symposium on Fault-Tolerant Computing*, pages 200–206, Ann Arbor, Michigan, June 1985. A revised version appears as IBM Technical Report RJ5244.

[Cen87] IBM International Technical Support Centers. IBM/VS extended recovery facility (XRF) technical reference. Technical Report GG24-3153-0, IBM, 1987.

[Cri90] Flaviu Cristian. Synchronous atomic broadcast for redundant broadcast channels. *Journal of Real-Time Systems*, 2:195–212, September 1990.

[JB89] Thomas Joseph and Kenneth Birman. *Reliable Broadcast Protocols*, pages 294–318. ACM Press, New York, 1989.

[Lam78] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[LGG+91] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, and Michael Williams. Replication in the Harp file system. In *Proceedings of the 13th Symposium on Operating System Principles*, pages 226–238, 1991.

[LMS85] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[MHS89] Timothy Mann, Andy Hisgen, and Garret Swart. An algorithm for data replication. Technical Report 46, Digital Systems Research Center, 1989.

[NT88] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed systems. In *Proceedings of the Seventh ACM Symposium on Principles of Distributed Computing*, pages 248–262, Toronto, Ontario, August 1988. ACM SIGOPS-SIGACT.

[OL88] B. Oki and Barbara Liskov. Viewstamped replication: A new primary copy method to support highly available distributed systems. In *Seventh ACM Symposium on Principles of Distributed Computing*, pages 8–17, Toronto, Ontario, August 1988. ACM SIGOPS-SIGACT.

[PSL80] M. Pease, R. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[Sch90] Fred B. Schneider. Implementing fault tolerant services using the state machine approach: A tutorial. *Computing Surveys*, 22(4):299–319, December 1990.

[SS83] Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, August 1983.

Appendix: Method of Proof

The proofs of our theorems are very similar: we assume that the theorem is false and construct a set of runs. We then use an indistinguishablity argument to derive a contradiction. The following proof of Theorem 5 illustrates this technique.

Theorem 5 *Any primary–backup protocol tolerating receive-omission failures with $f_s > 1$, $n \leq 2f_s$ and $D < \Gamma$ is C–blocking for some $C \geq 2\delta$.*

Proof: For contradiction, suppose there is a primary–backup protocol for $n \leq 2f_s$ and $f_s > 1$ that is C–blocking where $C < 2\delta$. Partition the servers into two sets A and B where $|A| = f_s$ and $|B| = n - f_s \leq f_s$. We construct three runs. In all three runs, assume that all messages sent between servers take δ to arrive.

σ_1: There are no failures and all client requests take δ to arrive. Moreover, clients send update requests until some request m evokes a response r. Let m be received at time t_m by server $p \in A$ and r be sent at time t_r by a different server $q \in A$. Notice that since the protocol is C–blocking where $C < 2\delta$, $t_r - t_m < 2\delta$. Also, since by construction all requests take δ to arrive, all client requests sent after time $t_m + \delta$ will be received after time t_r.

σ_2: Identical to σ_1 until p receives m at time t_m. At this point in σ_2, all servers in A are assumed to crash and clients are assumed to send no request during the interval $[t_m + \delta..t_r]$. Finally, after time t_r clients are assumed to repeatedly send requests at intervals of at least d where $0 < d \leq \Gamma - D > 0$ as follows. A request is sent at time t if no request has been sent in $[t - d..t)$ and one of the following rules hold.

1. A server $s \in B$ is the primary during the interval $[t - D..t]$. This request arrives immediately and is enqueued (at s, by Pb3 and the definition of D).

2. There is no primary in B at time t. This request arrives immediately by Pb3 will never be enqueued at any server.

3. A server $s \in B$ is the primary at time t but another server $s' \in B$ is the primary immediately after time t. If the request is sent to s, then it arrives after t, and if it is sent to any other server it arrives immediately. In both cases, it arrives at a server that is no the primary, and so will not be enqueued (again, by Pb3).

Notice that eventually, there will be a response (say r') in σ_2 because the protocol satisfies Pb4, and by construction it must be from a request sent by rule 1.

σ_3: The same as σ_2, except that the servers in A *do not* crash at time t_m. Instead, the servers in B commit receive failures on all messages sent after t_m by servers in A. Clients send requests at the same times as in σ_2 which arrive using the same rules as σ_2.

Now, consider these three runs. By construction, the runs are identical up to time t_m. Since all server messages take δ to arrive, clients cannot distinguish σ_1 and σ_3 through $t_m + \delta$, and so clients send the same requests to the same servers in both σ_1 and σ_3. Similarly, since all server messages take δ to arrive, the servers in B cannot distinguish between σ_1 and σ_3 through $t_m + \delta$. Therefore, since $t_r - t_m < 2\delta$, p (the server that received request m at time t_m in σ_1) and q (the server that sent response r at time t_r in σ_1) cannot distinguish between σ_1 and σ_3 through time t_r, and so q sends response r in σ_3 as well. Then, using an argument similar to the one in Theorem 2, servers in B cannot distinguish σ_2 and σ_3, and so response r' also occurs in σ_3. However, $\neg(e(m) \prec e(r'))$ which violates the assumption that m is an update request. □

In run σ_3 of the above proof, a faulty server from set B becomes the primary even though the original primary (p in set A) has committed no failure. It is always possible to construct such a run for $n \leq 2f_s$. This is a disconcerting property: there does not exist a primary–backup protocol that tolerates receive-omission failures with $n \leq 2f_s$ in which a primary cedes only when it fails. Moreover, this lower bound is tight—we have constructed a receive-omission primary–backup protocol with $n = 2f_s + 1$ in which a primary cedes only when it fails.