Time-Communication Tradeoffs for Reliable Broadcast*

Özalp Babaoğlu
Rogerio Drummond†

TR 85-687
June 1985
(Revised December 1985)

Department of Computer Science
Cornell University
Ithaca, New York 14853

*Partial support for this work was provided by the National Science Foundation under Grant MCS 82-10356.

†Current address: Computer Science Department, State University of São Paulo at Campinas, São Paulo, Brazil. Supported through a Fellowship from the CAPES Agency of the Government of Brazil.

Time-Communication Tradeoffs for Reliable Broadcast Protocols*

Özalp Babaoğlu
Rogério Drummond †

Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.

ABSTRACT

In the *Reliable Broadcast Problem*, a processor disseminates a value to all other processors in a distributed system where both processors and communication components are subject to failures. Solutions to this Reliable Broadcast problem are at the heart of most fault-tolerant applications. We characterize the execution time of Reliable Broadcast protocols as a function of the properties of the underlying communication network. The class of networks considered includes familiar communication structures constructed out of fully-connected point-to-point graphs, linear chains, rings, broadcast networks (such as Ethernet) and buses. We derive a protocol that implements Reliable Broadcast for any member within this class. The execution time of the protocol is a linear function of the two parameters that characterize each network instance. The hardware-software tradeoffs that are revealed between performance, resiliency and network cost offer many new alternatives previously not considered in designing fault-tolerant systems.

December 16, 1985

---

* Partial support for this work was provided by the National Science Foundation under Grant MCS 82-10356.

† Current address: Computer Science Department, State University of São Paulo at Campinas, São Paulo, Brazil. Supported through a Fellowship from the CAPES Agency of the Government of Brazil.

1. Introduction

A *distributed (computing) system* is a collection of autonomous processors that share no memory, do not have access to a global clock, and communicate only by exchanging messages. This model of computing matches the geographic distribution that is inherent in a large number of applications better than centralized systems. The lack of shared memory and random communication delays make programming such systems difficult.

Continued and correct operation are important requirements for many computing systems engaged in process control [Spe] and database applications [Kim]. Unfortunately, given a finite amount of hardware, it is impossible to construct a computing system that never fails. The best we can hope to achieve are systems that continue correct operation with respect to some specification such as "with high probability" or "as long as the number of faulty components during some time interval is small." Replication is a common technique to realize such goals. The fault tolerance requirements usually dictate that the replicated system not rely on the correctness of any single component for its correct operation. Consequently, when viewed at an appropriate level of abstraction, the replicated system has the same properties as a distributed system—replicated processors with no shared resources that communicate through a network. The presence of failures adds to the difficulty of programming fault-tolerant distributed systems.

Recently, much effort has gone into identifying primitives that simplify implementing fault-tolerant distributed applications [Lam, SL, CAS, Svo]. One such primitive is the *Reliable Broadcast* (also called *Byzantine Agreement* [LSP, Fis]). Formally, a protocol implements Reliable Broadcast (RB) if it guarantees the following two conditions:

RBA: *(Agreement)* In response to a broadcast, all correct processors accept† the same message.

RBV: *(Validity)* If the broadcasting processor, called the *sender*, is correct, then all correct processors accept the message that was broadcast.

Given an implementation for this protocol, we can use the following methodology for designing fault-tolerant distributed applications [Lam, SL, CAS, GMP]:

(i) Program the application assuming that each processor has direct access to the (same) global system state at all times.

(ii) Use a RB protocol to realize this assumption as follows:

- Each processor maintains a local copy of the information that constitutes the global state and updates it as instructed by incoming messages.

† We distinguish between a processor "receiving" and "accepting" a message. "Receive" is the general communication primitive supported by the network whereas "accept" is implemented by the reliable broadcast protocol. Therefore, it is possible for a processor to receive a message but not to accept it.

- Whenever the processor performs a local computation that modifies the global state, it disseminates the change to all other processors using RB.

For many applications, correct processors are not only required to accept the same messages but accept them in the same order. The required primitive is called an *atomic broadcast* [CAS] and can be implemented on top of a reliable broadcast protocol by using timestamps.

This design methodology places RB at the heart of a fault-tolerant distributed application implementation. Consequently, the overall performance of the application will be determined primarily by the performance of the RB protocol implementation. In systems with point-to-point communication structures, proposed RB protocol implementations are expensive in execution time. More significantly, lower bound results prove that faster execution times are impossible [Had, DS, FL]. We note that randomization can be used to speed up the expected (but not the worst case) execution time of RB protocols [Bra]. In [BD], we described fast RB protocols that exploited communication architectures other than point-to-point networks. This was the first result suggesting that performance could be "bought" by investing in the appropriate communication hardware. In this paper, we explore general tradeoffs between execution time, resiliency, and the properties of the underlying communication network with respect to RB protocols.

2. Models and Assumptions

A processor is said to be *correct* if its behavior conforms to an abstract specification (usually in the form of an algorithm). Each time that the behavior of a processor differs from its specification, a *failure* is said to occur; the processor is called *faulty* after the first failure. In our system both processors and communication components can fail. Note that a single faulty component can generate many failures. In this paper, we consider systems subject to *omission* failures [Had], where a faulty processor may fail to send some of the messages prescribed by its protocol, but messages it does send are always correct. Failures in the communication components cause messages to be lost; all messages that are delivered are delivered unchanged. Simple network protocols that approximate this behavior of the communication media are well known [Tan].

We make the following additional assumptions about the system:

NA: *(Network Assumption)* Despite processor and communication component failures, the network remains connected and guarantees a bounded delivery time for all messages.

CA: *(Clock Assumption)* Correct processors have local clocks that differ by at most ε units and have bounded drift with respect to (unobservable) absolute time.

NA requires that the network have redundant paths. Protocols that achieve CA in the presence of faulty components are known [HSS, LMS, ST].

For clarity, we present our protocols as distributed computations executing in lockstep units called *rounds* (i.e., assuming $\varepsilon = 0$). Informally, a round is the minimum length of time required for each processor to send messages to all other processors, receive any messages destined for it and perform a given amount of local computing. In Section 7 we show how to transform such round-based protocols into equivalent ones that work in a system where $\varepsilon > 0$.

3. Broadcast Networks

A common architecture for a distributed system consists of several processor clusters on a common network where the intra-cluster communication takes place over shared, multiple-access media that support broadcasts [Sta]. Figure 1 illustrates such an architecture with three clusters of sizes n_1, n_2

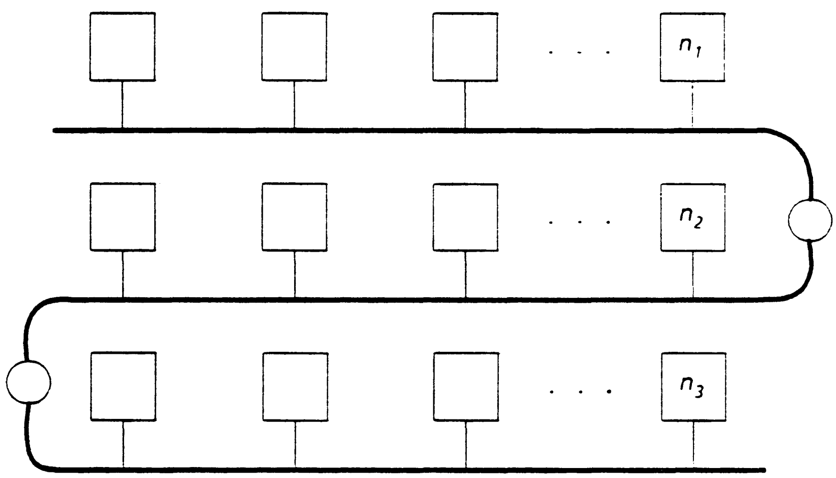

Fig. 1. *A broadcast network-based distributed system with three clusters*

and n_3 processors. This broadcast network-based architecture encompasses a wide range of designs. For example, the clusters in Figure 1 could represent geographically distant local area networks connected through gateways (such as the Xerox Internet comprising a large number of Ethernets [MB]). Alternatively, each cluster could be a single, tightly-coupled multiprocessor with an internal bus interconnect and inter-cluster links implemented as bus adaptors.

Regardless of the physical realization of the architecture, we can abstract the behavior of communication in such systems as follows:

BNP: *(Broadcast Network Property)* In response to a **broadcast**, all processors that receive a message receive the same message.

BNP ensures that for all possible failures, a processor cannot send conflicting messages to other processors in a single broadcast. For a given broadcast network, the set of processors that receive the (same) message in response to a broadcast is called the *receiving set.* The *broadcast degree* of a network is defined to be a lower bound on the size of the receiving sets for all broadcasts in the network. The receiving sets may vary from one broadcast message to another as well as from one sender to another. For a network to have broadcast degree b, all we require is that each of these sets contain at least b processors. We assume that every processor receives its own broadcasts regardless of failures. Note that communication failures manifest themselves in defining a particular broadcast degree for the network. For the example depicted in Figure 1, if we assume that communication failures can occur only in the gateways (or whatever physical component the two circles are modeling), it is easy to see that the broadcast degree of the network is $b = \min(n_1, n_2, n_3)$.

It is clear that a single communication failure anywhere in the network is sufficient to partition the system in Figure 1. In general, satisfying the Network Assumption (NA) requires replication of the network components. A *redundant broadcast network* interconnects processors through a sufficient number of broadcast networks, each exhibiting the Broadcast Network Property, such that NA is satisfied. The *broadcast degree* of the entire redundant network is defined to be the minimum of the broadcast degrees of its constituents. We assume that the **broadcast** primitive, when invoked by a correct processor, functions by broadcasting the message over all of the network replicas which the processor is connected to. However, no assumptions are made regarding the order in which networks are selected or the possibility of omitting messages from certain networks by faulty processors. Multiple rings, Ethernets or buses can be used to construct redundant broadcast networks. In the rest of the paper, we will consider only redundant broadcast networks and refer to them simply as broadcast networks.

4. From Broadcasts to Point-to-Point Communication

We will initially use the broadcast degree of a network as its characterization. Clearly, for a system with n processors, $1 \leq b \leq n$. Note that for $b = n$, every broadcast is guaranteed to be received by every other processor. Therefore, a protocol in which the sender simply issues a broadcast and other processors accept any message received suffices to achieve RB since BNP with $b = n$ trivially guarantees the agreement and validity requirements. At the other extreme, when $b = 1$, there can be isolated processors and the network cannot be guaranteed to remain connected. Consequently, the smallest value of b for which assumption NA can be satisfied is 2.

Consider a broadcast network with $b = 2$. In this case, the network can only guarantee that for some broadcast during the protocol execution, the message will be received by at most one other processor. In other words, for some communications the broadcast networks degenerate into point-to-point links between two processors. As this can happen arbitrarily often, the broadcast network becomes functionally equivalent to a point-to-point network. The "logical links" that are formed in the network result in a fully-connected communication graph.

Summarizing, in a broadcast network where no communication failures occur such that $b = n$ can be attained, there exists a trivial 1-round protocol that implements RB. On the other hand, a broadcast network with $b = 2$ is logically equivalent to a point-to-point network where we know that $t+1$

rounds are necessary and sufficient to achieve RB [FL, DS, Had]. What remains to be resolved is the execution time required for a RB protocol in networks where the broadcast degree is between these two extremes. We will call such a limited broadcast capability a *partial broadcast*.†

5. Reliable Broadcast Protocols Through Partial Broadcasts

Initially, in addition to NA, we will assume that a **broadcast** performed by a correct processor reaches all other processors in the same round, regardless of the broadcast degree of the network. Consequently, any protocol where processors simply echo the messages received in the previous round will guarantee that all correct processors receive a message one round after the first correct processor receives a message. As the processor failures are restricted to omission, all of the messages generated by such a protocol can only contain the sender's value. What we have to demonstrate is an upper bound on the number of rounds by which time all correct processors can accept some default value (denoted as δ) if they have not received any messages.

Consider a broadcast network where $t+1 \le b \le n-1$. In this case, the initial broadcast by the sender, with local value v, is received by at least one but not necessarily all correct processors. By the above observations, all processors are guaranteed to receive the sender's message by the end of the

† The term *multicast* has also been used for such limited broadcasts.

Round 1:
For s (the sender with local value v)
    broadcast(v);
    accept v;

Round i, $2 \le i \le m$:
For all p that have not yet accepted a value
    if p received a message in round $i-1$ **then**
        let μ *denote the message received in round* $i-1$
        broadcast(μ);
        accept μ;
    fi;

At the conclusion of round m:
For all p that have not yet accepted a value
    if p received a message in round m **then**
        let μ *denote the message received in round* m
        accept μ;
    else
        accept δ;
    fi;

Fig. 2. *Protocol P1 with parameter* m*:*
Reliable Broadcast for partial broadcast networks

second round. They accept v if a message is received in the first or second round. Otherwise they accept the default value. In [BD], we have presented such two-round protocols for various processor and communication failure models and showed their optimality with respect to the number of rounds.

We now consider the remaining range of broadcast degrees. The parameterized protocol displayed in Figure 2 implements RB for broadcast networks with broadcast degrees in the range $2 \le b \le t+1$. Note that this protocol closely resembles the one presented in [Had]. The following theorem establishes its correctness.

Theorem 1. For a broadcast network with broadcast degree b such that $2 \leq b \leq t+1$ protocol P1 implements RB in $m = t-b+3$ rounds.

Proof: Let v denote the sender's local value. We proceed by case analysis. If the sender is correct, all processors receive v by the end of the first round and accept it in the following round. Therefore agreement and validity conditions are satisfied.

In the remaining cases the sender is faulty. If it sent no messages at all, no processor ever receives a message and they all accept δ by the end of round m. If the sender broadcasts over a channel whose degree is n, then all processors receive the message and accept v in the second round.

Finally, if the sender is faulty and broadcasts some message and a correct processor q receives it in round i, for $i<m$, then q will echo v in round $i+1$. By our assumption, all processors receive v in round $i+1$ and by round $min(m,i+2)$ they all will have accepted v. Thus agreement can be violated only if the first correct processor to receive v does so at round m while some other correct processors do not receive v. For contradiction suppose this situation occurs. According to the protocol, a processor can only broadcast at round i if it has received its first message at round $i-1$. Therefore, for the above scenario to occur there must be a chain of m faulty

processors $spqr \cdots z$ such that s is the sender and the kth processor receives its first message from its immediate predecessor in the chain in round $k-1$ and echoes it in round k. Some but not all correct processors receive their first message from processor z at round m. Given that the network has broadcast degree b, we show that at least $b+m-2$ faulty processors are necessary for this scenario to occur. First, the broadcast of s has to be received by b faulty processors including p but none of $qr \cdots z$. In the subsequent rounds (2 through $m-1$) the b faulty processors receiving the broadcast must contain at least one previously unaccounted faulty processor. Consequently, at least $b+m-2$ faulty processors are needed: b in the first round and $m-2$ in the remainder of the protocol. This contradicts the statement of the theorem requiring $m = t-b+3$. □

Note that execution time of the protocol increases linearly from 2 rounds to $t+1$ rounds as the broadcast degree decreases from $t+1$ to 2 (corresponding to the point-to-point case).

6. Coping with Networks of Arbitrary Diameter

Recall that within a round, each processor can communicate with all other processors. We define a *phase* to be the interval of time necessary for all processors to communicate only with their immediate neighbors in the current system connectivity graph. Note that during the execution of the protocol, the original system graph will dynamically evolve as processor and communication failures occur. Consequently, the set of immediate neighbors of a processor can change from phase to phase. The assumption we made in the previous section, whereby a broadcast by a correct processor is received by all other processors in the same round implies a system where phases and rounds are equivalent.

In this section we relax this assumption and consider systems where a single phase is no longer sufficient to implement a round. In such systems, let the *survivor diameter*, denoted as d, be the maximum number of phases required to achieve a round under all permissible failure scenarios [BDF].

The *diameter* of a graph, denoted as d_0, is defined to be the number of edges in the longest of the shortest paths between pairs of vertices. Using our terminology, the diameter of a network is the maximum number of phases necessary to implement a round in the absence of failures. It is important to note that whereas d_0 has a purely static definition, the survivor diameter of a system is defined dynamically. Consider a network with survivor diameter d. Such a network can be actually partitioned during some phases but still be connected when viewed over the entire d phases. In fact, in systems where the communication components fail and recover frequently, it is conceivable that at any given time, the network may contain processor pairs that cannot communicate. For a network to have survivor diameter d, all that is required is that the failures in the system be such that no processor is prevented from receiving a message broadcast by any other correct processor within d phases. Thus, we can relax the Network Assumption by replacing the connectivity requirement with the survivor diameter specification.

Protocol P1 of the previous section was presented in terms of rounds. Given that in general d phases are required to implement a round, protocol P1 could be used for networks with arbitrary d provided that the **broadcast** primitive is viewed as the invocation of a d-phase delivery protocol. In this approach, the total execution time of the protocol is clearly $d(t-b+3)$ phases. However, as the next result shows, we can achieve RB in a number of phases that is an additive (rather than a multiplicative) function of d by "pipelining" the communication in the network. Dolev and Strong present a similar result for arbitrary processor failures in fully reliable, point-to-point networks [DS].

In protocol P1, we simply replace rounds with phases such that m now is the number of phases and the **broadcast** primitive, when executed by a correct processor, distributes the message to its current immediate neighbors (as before).

Theorem 2. For a broadcast network with survivor diameter d and broadcast degree b such that $2 \leq b \leq t+1$ protocol P1 implements RB in $m=t-b+d+2$ phases.

Proof: Let v denote the sender's local value. We proceed by case analysis. If correct processors receive no messages by the end of phase m they all accept δ. In the remaining cases some correct processors receive v during the execution of the protocol.

If the first correct processor receives v at phase i, for $i \le t-b+2$, then all correct processors will receive v within the next d phases. Since $i+d \le m$ for all allowed values of i, all correct processors receive and accept v within m phases. Note that this case applies for correct senders, and therefore the validity condition is satisfied.

We claim that this represents the latest phase by which a correct processor can receive the message. For contradiction, suppose the first correct processor to receive v does so at phase $i > t-b+2$. A chain of i faulty processors $spqr \cdots w$ with the same properties as the one introduced in the proof of Theorem 1 is required. Also the $b-1$ processors that receive the sender's initial broadcast must all be faulty and different from $qr \cdots w$. Thus the total number of faulty processors necessary for this scenario is $i+b-2$. But this exceeds t for any value of i, a contradiction.

In summary, either a correct processor receives v by phase $t-b+2$ and all correct processors accept v, or no correct processor receives a message and all accept δ at phase m. Therefore the agreement condition is satisfied, concluding the proof. □

Now, we notice that the execution time of the protocol increases linearly with either decreasing broadcast degree or increasing survivor diameter.

7. Coping with Clocks that are not Perfectly Synchronized

Here, we briefly sketch how any protocol that requires perfectly synchronized local clocks can be modified to work in a system that only guarantees the Clock Assumption. Recall that under this assumption, local clocks of non-faulty processes differ by at most ε units (known *a priori* by all processes) and local times have bounded drift with respect to absolute time.

Let L be a constant known to all processes that denotes the duration of each phase. Since all processes know when the protocol is to be initiated, they progress through phases in lockstep. Let τ be the starting time of an arbitrary phase, known to all processes. If local clocks are allowed to be at most ε units apart, a (slow) process could receive messages from (fast) processes as early as time $\tau-\varepsilon$, as measured on its local clock. Conversely, a (fast) process could receive messages from (slow) processes as late as local time $\tau+L+\varepsilon$. Therefore, the duration of a phase could be redefined to be $L+2\varepsilon$ such that messages received within the interval $[\tau-\varepsilon, \tau+L+\varepsilon]$ belong to the current phase. This interval consists of the two *passive* segments $[\tau-\varepsilon, \tau]$ and $[\tau+L, \tau+L+\varepsilon]$ where processes simply wait for arriving messages, and the *active* segment $[\tau, \tau+L]$ during which they execute the protocol.

Using this simple construction, a protocol requiring m phases would take $m(L+2\varepsilon)$ time units. By overlapping the passive segments of two adjacent phases, we can reduce this time to $m(L+\varepsilon)+\varepsilon$. All that is required is that messages be identified with the appropriate phase number so that late messages from the previous phase and early messages from the current phase can be distinguished.

8. Discussion

We note that the utility of any fault-tolerant protocol is inherently probabilistic, even if the protocol itself is deterministic and correct. For example, consider some protocol Π that implements an operation Ω in a system subject to failures described by Φ. The probability with which Π correctly achieves Ω is the probability that actual failures during the protocol execution conform to Φ. Clearly, as the description of failures that a protocol must tolerate becomes less specific, the design has to become more conservative (and probably more expensive). For instance, a protocol designed to tolerate the requirement "at most 3 faulty processors" can in all likelihood tolerate many more than 3 faulty processors provided they occur in configurations that are not "worst case." If one can quantify the probability with which such configurations occur, the same protocol may be acceptable as the solution to more demanding fault-tolerance requirements.

For the systems we have been considering, the failure description Φ consists of the triple (t, b, d). Given a description of the system components and the static communication topology, for most cases, one can derive upper bounds for t and d and a lower bound for b to base the design on. However, this approach will result in designs that are overly conservative. We are working on deriving the joint probability distribution for the system to exhibit a given number of faulty processors, broadcast degree and survivor diameter during an execution of the RB protocol [Dru]. With this characterization, one can select an instance of the RB protocol from the entire design space based on the desired confidence level for correct execution.

9. Conclusions

We have studied the reliable broadcast problem in distributed systems where the communication network is characterized by its broadcast degree and survivor diameter. Conventional network topologies such as fully connected point-to-point graphs, simple linear chains and rings correspond to the special cases $(b=2,\ d=1)$, $(b=2,\ d=n-1)$ and $(b=2,\ d=\lceil (n-1)/2 \rceil)$, respectively, of this characterization. In general, the class of networks that are generated for arbitrary b and d correspond to common architectures consisting of a collection of (possibly non-homogeneous) broadcast networks that are interconnected through gateways where any of the network components may fail.

We have exhibited a simple protocol to implement reliable broadcast in systems based on such communication networks. The protocol degenerates into the familiar solutions for RB at the extreme points of the characterization—fully connected point-to-point networks and full broadcast networks. In both cases, our protocol matches the established execution time lower bounds for RB [Had, BD].

Given that ring, Ethernet and bus type communication structures, which support varying degrees of partial broadcasts, are extremely common in practical distributed systems, our results have wide applicability. When synthesizing a fault-tolerant application within one of these environments, a designer now has the option to trade performance and resiliency for network hardware costs. For a desired fault tolerance, a spectrum of performances can be "bought" by investing in the appropriate network structure.

Acknowledgments

The authors would like to thank Fred Schneider and Pat Stephenson for commenting on earlier versions of this paper.

References

[BD] Babaoğlu, Ö. and Drummond, R. Streets of Byzantium: Network architectures for fast reliable broadcast. *IEEE Trans. Software Eng.*, vol. SE-11 (June 1985), 546-554.

[BDF] Broder, A., Dolev, D., Fischer, M., Simons, B. Efficient fault tolerant routings in networks. In *Proceedings of 16th ACM Symposium on Theory of Computing* (1984), 536-541.

[Bra] Bracha, G. An $O(\lg n)$ expected rounds randomized Byzantine Generals protocol. *Proc. 17th ACM Symposium on Theory of Computing*, Providence, Rhode Island, (May 1985), 316-326.

[CAS] Cristian, F., Aghili, H., Strong, R. and Dolev, D. Atomic Broadcasts: From simple message diffusion to Byzantine Agreement. Research Rep. RJ 4540 (48668), IBM Research Lab., San Jose, California, December 1984.

[Dru] Drummond, R. Network architectures for fault-tolerant distributed computing. Ph.D. Thesis, Computer Science Dept., Cornell University, Ithaca, NY (in preparation)

[DS] Dolev, D. and Strong, H.R. Authenticated algorithms for Byzantine agreement, *SIAM J. on Comput. 12*, 4 (November 1983), 656-666.

[Fis] Fischer, M. The consensus problem in unreliable distributed systems (A Brief Survey), Tech. Rep. YALEU/DCS/RR-273, Dept. of Computer Science, Yale University, New Haven, Connecticut, June 1983.

[FL] Fischer, M. and Lynch, N. A lower bound for the time to assure interactive consistency. *Inform. Proc. Letters 14*, 4 (April 1982), 183-186.

[GMP] Garcia-Molina, H, Pittelli, F. and Davidson, S. Applications of Byzantine Agreement in database systems, Tech. Rep. TR 316, Princeton University, Princeton, New Jersey, June 1984.

[Had] Hadzilacos, V. Issues of fault tolerance in concurrent computations. Ph.D Thesis, Tech. Rep. TR-11-84, Aiken Computation Laboratory, Harvard University, Cambridge, Mass., June, 1984.

[HSS] Halpern, J.Y., Simons, B., Strong, H.R. and Dolev, D. Fault-tolerant clock synchronization. In *Proceedings of 3rd ACM Symposium on Principles of Distributed Computing*, Vancouver, B.C., Canada (August 1984), 89-102.

[Kim] Kim, W. Highly available systems for database applications, *ACM Computing Surveys*, vol. 16, no. 1, (March 1984), 71-98.

[Lam] Lamport, L. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. Prog. Lang. Syst. 6*, 2 (April 1984), 254-280.

[LMS] Lamport, L. and Melliar-Smith, P.M. Byzantine clock synchronization. In *Proceedings of 3rd ACM Symposium on Principles of Distributed Computing*, Vancouver, B.C., Canada (August 1984), 68-74.

[LSP] Lamport, L., Shostak, R. and Pease, M. The Byzantine Generals problem. *ACM Trans. Prog. Lang. Syst. 4*, 3 (July 1982), 382-401.

[MB] Metcalfe, R. and Boggs, D.R. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM 19*, 7 (July 1976), 396-403.

[SL] Schneider, F.B. and Lamport, L. Paradigms for distributeed programs. In *Distributed Systems: Methods and Tools for Specification*, Paul, M. and Siegert H.J. (Eds.), Springer-Verlag Lecture Notes in Computer Science Vol. 190.

[Spe] Spector, A.Z. Computer software for process control, *Scientific American*, vol. 251, no. 3, (September 1984), 174-187.

[ST] Srikanth, T.K. and Toueg, S. Optimal Clock Synchronization, *Proc. 4th Symposium on the Principles of Distributed Computing*, Minaki, Canada (August 1985).

[Sta] Stallings, W. Local networks, *ACM Computing Surveys*, vol. 16, no. 1, (March 1984), 3-41.

[Svo] Svobodova, L. Resilient distributed computing. *IEEE Trans. Software Eng.*, vol. SE-10 (May 1984), 257-268.

[Tan] Tanenbaum, A. *Computer Networks*. Prentice Hall, Englewood Cliffs, N.J., 1981.