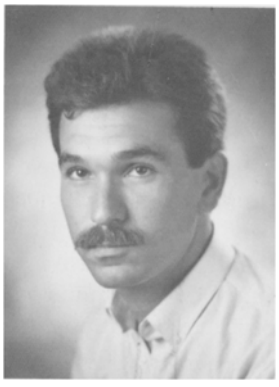
Distributed Computing (1988) 2:177–189

DISTRIBUTED COMPUTING

Reliable broadcasts and communication models: tradeoffs and lower bounds*

Özalp Babaoğlu[1], **Pat Stephenson**[1]**, and **Rogério Drummond**[2]***

[1] Department of Computer Science, Cornell University, Ithaca, NY 14853-7501, USA
[2] Computer Science Department, Universidade Estadual de Campinas – UNICAMP, 13081 Campinas, São Paulo, Brazil



Özalp Babaoğlu is Associate Professor in the Department of Computer Science at Cornell University, Ithaca, New York. His research interests include distributed systems, fault tolerance, performance evaluation and modeling. He received a BS in electrical engineering from George Washington University, Washington, D.C. in 1976. From the University of California, Berkeley, he received a MS in 1977 and a PhD in 1981, both in computer science. While at Berkeley, he designed and implemented the virtual memory extensions to VAX Unix that came to be known as 3. 0bsd.

Pat Stephenson is a Doctoral Candidate in the Computer Science Department at Cornell University, Ithaca, New York. His research interests include distributed systems and fault tolerance. In 1983, he received a B.A. (Mod.) in computer science from Trinity College, Dublin, Ireland. He received his MS in computer science from Cornell in 1986. He is currently working on new tradeoffs in the design of fault-tolerant algorithms.

Offprint requests to: Ö. Babaoğlu

* Partial support for this work was provided by the National Science Foundation under Grants DCR-86-01864 and MCS-82-10356 and AT&T under a Foundation Grant

** Supported partially by the Defense Advanced Research Projects Agency (DoD) under ARPA order 5378, Contract MDA903-85-C-0124, and partially by an IBM graduate fellowship. The views, opinions and findings contained in this report are solely those of the authors and should not be construed as an official Department of Defense position, policy, or decision

*** Supported partially by the CAPES and CNPq agencies of the Ministry of Education of Brazil

Rogério Drummond is Assistant Professor in the Computer Science Department at the Universidade Estadual de Campinas (UNICAMP), São Paulo, Brazil. His interests include distributed computing, fault tolerance and operating systems. He received a BS and a MS in computer science from the Universidade Estadual de Campinas in 1978 and 1980, respectively. In 1986 he received a PhD in computer science from Cornell University. He is currently working on integrated environments for the development of software and hardware.

Abstract. *Reliable Broadcast* is a mechanism by which a processor in a distributed system disseminates a value to all other processors in the presence of both communication and processor failures. Protocols to achieve Reliable Broadcast are at the heart of most fault-tolerant applications. We characterize the execution time of Reliable Broadcast protocols as a function of the communication model. This model includes familiar communication structures such as fully-connected point-to-point graphs, linear chains, rings, broadcast networks (such as Ethernet) and buses. We derive a parameterized protocol that implements Reliable Broadcast for any member within this class. We obtain lower bound results that show the optimality of our protocols. The lower bound results identify a time complexity gap between systems where processors may only fail to send messages, and systems where processors may fail both to send and to receive messages. The tradeoffs that our results reveal between performance, resiliency and network cost offer many new alternatives previously not considered in designing fault-tolerant systems.

Key words: Distributed systems – Fault tolerance – Byzantine Agreement – Hardware-software tradeoffs

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 is suitable for the geographic distribution that is inherent in a large number of applications. The lack of shared memory and random communication delays make programming such systems difficult.

Continued and correct operation are important requirements for many computer systems [25, 16]. 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 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 – a collection of 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 can simplify programming fault-tolerant distributed applications [17, 24, 8, 28, 7]. One such primitive is the *Reliable Broadcast* (also called *Byzantine Agreement* [19, 23, 11]). Formally, a protocol implements Reliable Broadcast (RB) if it guarantees the following three properties:

RBA: (*Agreement*) In response to a broadcast, all correct processors accept[3] the same message.

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

RBT: (*Termination*) There exists a finite time (known a priori) by which all correct processors accept some message.

[3] 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

Given an implementation for this protocol, we can use the following methodology for designing fault-tolerant distributed applications [17, 24, 8, 13]

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

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

- Each processor maintains a local copy of the information that consitutes the global state and updates it as indicated by incoming messages.
- 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, they need to accept them in the same order. The appropriate primitive is called an *atomic broadcast* [8, 7, 5] 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. In systems with point-to-point communication structures, proposed RB protocols are expensive in execution time. More significantly, lower bound results prove that faster execution times are impossible [14, 10, 12, 9]. We note that the *expected* execution time of RB protocols can be reduced by either allowing non-deterministic computation steps (such as flipping coins) or by using "early-stopping" protocols [22, 6, 9]. Unfortunately, the worst case execution times are not improved by these alternatives. In [2], we described fast RB protocols that exploited communication architectures other than point-to-point networks. These were the first results suggesting that performance could be "bought" by investing in the appropriate communication hardware. In this paper, we complete the characterization of the time complexity of RB protocols with respect to the communication model. Our results allow a system builder to explore tradeoffs between execution time, resiliency, and the properties of the underlying communication network when considering alternate designs.

The remainder of this paper is organized as follows. The next section defines the failures we will consider and states some initial assumptions. Sect. 3 introduces broadcast networks in the context of Reliable Broadcasts. In Sect. 4, we discuss special cases of our network characterization corresponding to point-to-point and full broadcast networks. Section 5 is the development and correctness proof of a parameterized protocol that can achieve RB in any system encompassed by our characterization. The next two sections refine the basic protocol so as to increase its domain of applicability. We relax the assumption

regarding the network diameter in Sect. 6. In Sect. 7 we relax the assumption regarding clock synchrony and consider a broader class of failures. Section 8 contains the lower bound proofs. We prove the optimality of all of the protocols developed in the earlier sections by showing that their execution times match the respective lower bounds. Section 9 concludes the paper.

2 Models and assumptions

A processor or a communication component is said to be *correct* if its behavior conforms to an abstract specification (usually in the form of a program). Each time that the behavior of a processor differs from its specification, a *failure* is said to occur; the processor is called *faulty* after its first failure. Through out the paper, we let t denote the maximum number of faulty processors. In our system both processors and communication components can fail. Note that a single faulty component can generate many failures. If a processor fails by not sending some of the messages prescribed by its specification, it is said to exhibit *omission* failures [14]. The messages it does send are always correct. Initially, we consider only systems subject to omission failures. Failures in the communication components may cause messages to be lost; all messages that are delivered are delivered unaltered. Simple network protocols that approximate this behavior of the communication media are well known [29]. In Sect. 8 we will discuss the implications of relaxing our assumptions about the behavior of faulty processors.

We make the following additional assumption about the system:

NA: (*Network Assumption*) Effective communication is always possible between a correct processor and all other processors, despite failures. There is a known upper bound on the time required for such communication.

Achieving NA requires that the network provide redundant paths between processors such that partitioning does not occur. It does not require, however, that the network be fully connected.

For clarity, we present our protocols as distributed computations executing in lock-step units called *rounds* (i.e., processors have perfectly-synchronized clocks). 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 computation. In Sect. 7 we discuss how such a round-based protocol can be transformed into an equivalent one that works in a system where the clocks are not perfectly synchronized.

3 Broadcast networks

A common architecture for a distributed system consists of multiple processor clusters on a common network where the intra-cluster communication takes place over shared, multiple-access media that support broadcasts [27]. Figure 1 illustrates such an architecture with three clusters of sizes n_1, n_2 and n_3 processors. This broadcast network-based architecture encompasses a wide range of distributed system designs. For example, the clusters in Fig. 1 could represent geographically distant local area networks connected through gateways (such as the Xerox Internet comprising a large number of Ethernets [20]). Alternatively, each cluster could be a single, tightly-coupled multiprocessor with a bus interconnect and intercluster 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.

This property ensures that despite any possible failure, a processor cannot send conflicting messages to the other procesors 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*. We assume that every processor receives its own broadcasts regardless of failures. The *broadcast degree* of a network is defined to be the size of the smallest possible receiving set, for any broadcast where the message is received by at least one processor other than the broadcaster. More formally, let $B(p, i)$ denote the set of processors that receive processor p's ith broadcast such that $|B(p, i)| < 1$. Then, the

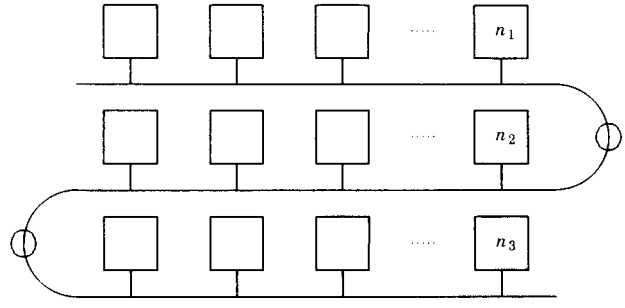


Fig. 1. A broadcast network-based distributed system with three clusters

the broadcast degree, b, of the network is

$$b = \min_{p,i} |B(p,i)|.$$

The receiving set itself may vary from one broadcast message to another as well as from one broadcaster to another. For a network to have broadcast degree b, all we require is that each of these receiving sets contain at least b processors. Given the Network Assumption, any broadcast by a correct processor reaches all processors. Thus, the broadcast degree of a network is a characterization of the behavior of faulty processors in that network. Note that failures manifest themselves in defining a particular broadcast degree for the network. For the example depicted in Fig. 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 will be $b = \min(n_1, n_2, n_3)$.

Clearly, a single communication failure anywhere in the network is sufficient to partition the system in Fig. 1. In general, satisfying the Network Assumption (NA) requires redundant communication paths. A *redundant broadcast network* interconnects processors through a sufficient number of broadcast channels, 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 (same) message over all of the channels to which the processor is connected. However, no assumptions are made regarding the order in which channels are selected or the possibility of faulty processors omitting broadcasts over some of the channels. 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 all processors. 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 broadcasts during the protocol execution, the messages 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. In the worst case, the broadcast network may allow a faulty processor to send a message to only one processor. As we will show in the following sections, $t+1$ rounds are necessary and sufficient to achieve RB in this case. Note that this time complexity for RB matches that of [12, 10, 14] for various failure models in point-to-point networks.

What remains to be resolved is the execution time required for an 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 broadcasts through partial broadcasts

Initially, in addition to NA, we will assume that a **broadcast** reaches all processors belonging to its receiving set in the same round. This assumption will be relaxed in Sect. 6. Note that in the case of a correct processor broadcasting, the receiving set includes all other processors. Consequently, any protocol where processors simply echo a message received in the previous round will guarantee that all correct processors receive that message one round after the first correct processor does so. 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 $t < b < n$. 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 second round. They accept v if a message is received in the first or second rounds. Otherwise they accept

```
Initialization:
For all processors
    α ← ?;

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

Round i, 2 ≤ i ≤ m:
    if α = ? and received message μ in round i − 1 then
        broadcast(μ);
        α ← μ;
    fi;

At the conclusion of round m:
    if α = ? and received message μ in round m then
        α ← μ;
    fi;
    if α = ? then
        accept δ;
    else
        accept α;
    fi;
```

Fig. 2. Protocol P1 with parameter m: reliable broadcast for partial broadcast networks

the default value. In [2], 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 Fig. 2 implements RB for broadcast networks with broadcast degrees in the range $2 \leqq b \leqq t+1$. Note that this protocol closely resembles the one presented in [14]. The following theorem establishes its correctness.

Theorem 1. *For a network with broadcast degree b, where $2 \leqq b \leqq t+1$, protocol P1 implements RB in the presence of up to t omission faulty processors 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 record it in their local variables α which are accepted at the end of round m. Therefore agreement, validity and termination 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 all accept δ at the end of round m. If the sender broadcasts over any channel whose broadcast degree is n, then all processors receive the message and accept v at the end of round m.

If the sender is faulty and broadcasts some message such that 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 $\min(m, i+1)$ they all will have set their accept variable α to v.

As a final possibility, a faulty sender could broadcast such that only a subset of the correct processors receive v for the first time exactly at round m. For contradiction suppose this situation occurs. According to the protocol, a processor is allowed to echo the message only in the round immediately after the round in which it first received it. Therefore, for the above scenario to occur there must be a chain of m faulty processors $spqr \ldots z$ 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, while other correct processors do not receive any message by 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. Recall that since failures are restricted to omission, faulty processors cannot delay echoing a message they have received beyond the first round after they receive it. So none of $qr \ldots z$ could have received the initial broadcast of s, and there must have been b other faulty processors, including s and p, that received the initial broadcast of s. Consequently, at least $b+m-2$ faulty processors are needed: b in the first round and $m-2$ in the remaining rounds. This contradicts the statement of the theorem where $t \leqq b+m-3$. □

Note that the 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 correct processor can communicate with all other processors in the system. We define a *phase* to be the interval of time necessary for all processors to communicate only with their current immediate neighbors. Note that during the execution of the protocol, the original system graph representing connectivity between processors 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 a phase and a round coincide.

In this section we relax this assumption and consider systems where a single phase is no longer sufficient to implement a round. The *diameter* of a graph, denoted as d_0, is the number of edges in the longest of the shortest paths between all pairs of vertices. Using our terminology, the diameter of a network is the minimum number of phases necessary to implement a round in the absence of failures. We define the *survivor diameter*, denoted as d, to be the maximum number of phases required to achieve a round (all pairwise communications) under all permissible failure scenarios [4]. More formally, if P is the set of all processors, P_c is the set of all correct processors and $K(p, q)$ is the maximum number of phases it can take for a broadcast from processor p to reach processor q, then the survivor diameter, d, is:

$$d = \max_{p \in P,\, q \in P_c} K(p, q).$$

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 round consisting of 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, the failures in the system must be such that every processor is guaranteed to receive a message broadcast by a 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 [10].

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 all of its current immediate neighbors.

Theorem 2. *For a network with survivor diameter d and broadcast degree b, where $2 \leqq b \leqq t+1$, protocol P1 implements RB in the presence of up to t omission faulty processors 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 $i \leqq t-b+2$, then all correct processors will receive v within the next d phases. Since $i+d \leqq m$ for all possible values of i, all correct processors receive v by the end of phase m and accept it. Note that this case applies to 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 $i > t-b+2$. For this to occur, a chain of i faulty processors $spqr \ldots z$ with the same properties as the chain 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 $qr \ldots z$, by a similar argument to that in the proof of Theorem 1. Thus the total number of faulty processors necessary for this scenario is $i+b-2$. But this exceeds t for any value of $i > t-b+2$, thus leading to a contradiction.

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

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

7 Coping with clock skew and timing failures

In this section, we briefly sketch how our round-based RB protocols can be modified to work in a system that does not guarantee perfectly synchro-

nized local clocks or where processors may commit "timing failures", to be defined below. The resulting protocols no longer have the lock-step round structure and messages propagate through the system as fast as the network can transport them. We note that our protocols have a structure similar to the "diffusion-based" protocols of [8] for systems with point-to-point communication networks.

Consider a system where each correct processor has a local hardware clock that exhibits bounded drift with respect to real time. In such systems, well-known protocols [15, 18, 26, 3] may be used to achieve global clock synchronization in the presence of faulty components, such that the following assumption is satisfied:

CA: (*Clock Assumption*) Correct processors have local clocks that differ by at most ε units and have bounded drift with respect to (unobservable) real time. The parameter ε is called the *clock skew* and is known to all processors a priori.

Consider a network with survivor diameter d and broadcast degree b, where $2 \leqq b \leqq t+1$, with up to t omission faulty processors for which CA holds. We modify our round-based protocols to simply relay all messages as soon as they are received, instead of waiting for the start of the next phase. Let L be a constant known to all processors that denotes the maximum duration of each phase, as measured on any correct clock. The principal components of L are the message preparation, transmission and receipt times for adjacent processors in the network. Let T denote the starting time of an arbitrary instance of the protocol that is known to all processors. That is, each correct processor starts the protocol when its local clock reads T. Then, it is easy to show that if each correct processor terminates the protocol when its local clock reads $T+(t-b+d+2)\,L+\varepsilon$, the agreement, validity and termination conditions for RB will be satisfied.

We next consider timing failures. A *timing failure* occurs when a processor performs an action specified by its protocol, but either too early or too late [8]. Note that omission and crash failures are special cases of timing failures.

Now consider a network with survivor diameter d and broadcast degree b, where $2 \leqq b \leqq t+1$, with up to t timing faulty processors. We will augment the previous protocol such that all messages are tagged with the number of the phase during which they are broadcast. A processor relays a message by incrementing its phase number by one and then rebroadcasting it. Note that faulty processors may affect the timing of the new broadcast, but not its contents (including the phase number). To change the content of a message requires a more malicious failure than those admitted by the timing failure model. The receive operation checks the phase number of a message as it arrives. This check ensures that any message received by a correct processor in a given phase can be relayed by that processor and will be received by *all* correct processors in the next phase. Therefore, if a processor receives a message tagged with phase number k at local time τ, the message will be ignored unless $T-k\varepsilon \leqq \tau \leqq T+k(L+\varepsilon)$. Note that this is precisely the "timeliness" check developed in [8]. If a phase k message is received within the above interval, as measured on a correct processor's local clock, the processor will tag it with phase number $k+1$ and relay it immediately. The message will arrive at all correct processors within the local clock range $T-(k+1)\,\varepsilon \ldots T+(k+1)(L+\varepsilon)$ and will clearly pass the receive check for phase $k+1$.

Given this modified protocol, we can compute the worst case termination time as a function of the system parameters t, b, d, L and ε. Recall from the proof of Theorem 2 that in the worst case, a chain of $t-b+2$ faulty processors can hide the message before the first correct processor sees it. Each of these faulty processors can delay the message by ε units before relaying it while still satisfying the receive condition. Consequently, a total of $(t-b+2)(L+\varepsilon)$ time units can elapse before the first correct processor receives the message. At this point, each correct processor relays the message as soon as it receives it. As the survivor diameter of the network is d, the additional time required for all correct processors to receive and accept the message is $dL+\varepsilon$ as measured on any clock. Adding the two components, we get $(t-b+2)(L+\varepsilon)+dL+\varepsilon$ as the worst case termination time of the protocol.

Note that this termination time coincides with that of the "timing fault tolerant" RB protocol of [8] when we characterize a point-to-point network by setting $b=2$. However, when $b>2$, our protocols achieve better performance.

8 Lower bound proofs

In this section we consider lower bounds — how fast can an RB protocol run in a given fault environment and given broadcast degree? In the case of omission failures, we show that the protocols derived in the previous sections are optimal. Next we consider more general failures. We show that if the failures to be tolerated are only slightly more

general than omission failures, then RB on a network with broadcast degree $b<t$ is inherently no faster (in terms of rounds) than RB on a point-to-point network. In particular, we show that if processors may exhibit *general omission* faults – fail to send and/or *receive* messages – then RB on a network with broadcast degree $b<t$ must take $t+1$ rounds. This result is somewhat surprising; in most other environments, a weakening of the constraints on faulty processor behavior does not lead to an increase in inherent time complexity of the problem. For example, in point-to-point networks, the RB problem can be solved in $t+1$ rounds whether processors fail by crashing, send omission [14] or exhibit arbitrary failures where authentication is not available (albeit at the expense of message complexity) [19]. It is interesting to note that other problems are known to become more difficult when the failure model is relaxed from send omission to general omission. Moses and Tuttle [21] have shown that in point-to-point networks with send omission faulty processors, optimal simultaneous action protocols require local polynomial-time computation at each processor; however, if the processors can commit general omission faults, optimal simultaneous action requires that each processor solve an NP-hard problem in each round.

If the broadcast degree of the network is large enough, RB *can* be achieved in fewer than $t+1$ rounds in the presence of arbitrary processor failures. In particular, Babaoğlu and Drummond exhibit a 2-round protocol for RB that can tolerate arbitrary failures if the broadcast degree of the network is greater than $t+n/2$. Furthermore, this protocol does not require the use of message authentication.

This section is organized as follows. First we describe a model of an unreliable distributed environment, and the execution of an RB protocol in such an environment. We define the concept of "similar runs" of a protocol; two runs are similar if a correct processor in one run accepts the same message as a correct processor in the other run. For any RB protocol, all correct processors must be able to distinguish the run where no failures occur from a run where the sender is faulty and the messages it sends are never seen by any correct processor. We derive our time complexity lower bounds by showing that these two runs are similar for a small enough number of rounds in the protocol execution. We give in detail the lower bound proof for omission failures in a broadcast network with unit diameter. Then, we outline the proofs for general omission failures and networks with arbitrary diameters.

8.1 Basic definitions and model of execution

Assume that the total number of processors is n, of which t can be faulty. We denote by v any value from the domain of broadcast values, and δ is the distinguished value that may be accepted if the sender is faulty. An RB protocol is modeled as a pair of deterministic functions (A, F), local to each processor. The first of the pair, A, is the *action function*; it defines the messages to be sent in every round by a processor as a function of all messages received by that processor in previous rounds. The second function F, is the *decision function*. It maps the messages received by a processor during any execution of the RB protocol into the decision value for that processor. The decision and action functions may depend on the identity of a processor.

The execution of a protocol in a given environment is modeled as a directed graph called a *run*. A run consists of a number of *levels*. Each level, except the last, corresponds to one round in the execution of the protocol. The last level corresponds to the state of the processors at the termination of the protocol. Thus, an m round protocol will have runs consisting of $m+1$ levels. Such runs are said to be of *depth m*. Each level consists of n *nodes*, one per processor. The node representing processor p at level i is denoted p_i. Each node p_i at level i, $1 \leq i \leq m$, has outedges directed to every node q_{i+1} that processor p sends messages to in round i. The label on an edge from node p_i to q_{i+1} is the message that p sends to q in round i. The sender has a special level 1 inedge indicating the value to be broadcast. A run R *restricted to* p, denoted as pR, is the subgraph obtained from R by deleting all edges except those directed to nodes p_i, $1 \leq i \leq m+1$.

In this model, the action function A for processor p is a function from pR up to level i into the outedges and labels for each node p_i. If the action function labels an outedge with a value from the message domain, this value must be a function of previously received values. The decision function F is a map from pR into the domain of possible message values, or δ. $F(pR)$ must return δ if p has no inedges in the entire run labeled with a value from the message domain.

We now incorporate the environment into this model. The *environment* in which a protocol is designed to execute is characterized by the following:

(1) A description of the allowable behavior of correct processors. This is just a specification of the allowable action function.

(2) A description of the behavior that faulty processors may exhibit. This is just a specification of the failure model.

Both of these can be specified in terms of the outedges that may exist in any run of a protocol in the environment, and the labels these outedges may have. In every run, the outedges from nodes representing correct processors must be as specified by A. A processor p is faulty by level i in a run R if the outedges from at least one of p_j, $1 \leqq j \leqq i$, are not as specified by the action function. They must, however, be consistent with the environment specification. No more than t processors may be faulty in any run.

The conditions for RB may be restated in this model as:

RBA: (*Agreement*) In any run R, if p and q are correct processors, then $F(pR) = F(qR)$.

RBV: (*Validity*) In all runs R, where s is correct and its level 1 inedge is labeled v, $F(pR) = v$ for all correct processors p.

RBT: (*Termination*) All runs have finite depth.

Two runs R and R' of a protocol (A, F) are called *similar*, denoted $R \sim R'$, if $F(pR) = F(qR')$ for some correct processor p in R and some correct processor q in R'. By definition the similarity relation is reflexive and symmetric. Since all runs satisfy RBA, similarity is also transitive.

One particular run of any RB protocol will be of interest to us. The *correct run*, denoted γ, is an execution of the protocol where all processors are correct and behave as specified by their action functions[4]. By the RBV requirement, $F(p\gamma) = v$ for all processors p.

We first consider omission failures in a broadcast network with unit diameter and show that the protocol developed in Sect. 5 is optimal.

8.2 Lower bound proof for RB protocols with partial broadcasts

In this section we give a lower bound for the number of rounds required for the RB problem in networks with broadcast degree b. Our proofs are generalizations of those found in [10, 14] that establish similar results for other communication and failure models. Our environment is characterized as follows:

(1) All runs are of depth $t-b+2$.

(2) In level i, $1 \leqq i \leqq t-b+2$, each node p_i either has:

(a) Outedges to all nodes in level $i+1$ (node p_i is correct).

(b) Outedges to k nodes, $b \leqq k < n$, in level $i+1$, including node p_{i+1} (processor p has failed by level i).

(c) No outedges (either node p_i is correct or processor p has failed by level i).

(3) Since we consider only omission failures in a broadcast network, all outedges from a particular node must have the same label.

Intuitively, to establish a lower bound, we should consider runs where the value transmitted by the sender is not exposed to the correct processors for as long as possible. We now formalize the class of such runs. We consider all runs satisfying RBA and RBV with which we can associate a sequence of sets of processors $S(1) \ldots S(t-b+2)$, called the *possible failure sequence*, with the following properties:

(1) $|S(r)| = b + r - 2$, $\quad 1 \leqq r < t-b+2$.

(2) $S(r) \subset S(r+1)$, $\quad 1 \leqq r < t-b+2$.

(3) If processor p has failed by level r, then $p \in S(r)$.

We call all such runs the *critical run set* and denote it as C. Note that we are not restricting the behavior of any RB protocol; only the class of failures we are considering. If a particular run H is a member of C, it may have many associated possible failure sequences. Also, note that γ, the execution in which no processors fail, is a member of C, in which case we may associate any sequence S satisfying conditions (1) and (3) with this run. Recall that in a correct execution all processors must accept the value v broadcast by the sender. Therefore, $F(p\gamma) = v$ for all correct p.

We will now outline our proof. We have defined an equivalence relation (similarity) on runs in C, and we have noted that γ is a member of C. We will show below that all runs in C are equivalent. We will also show that C includes runs where the initial value is never received by any correct processor. Such runs must arrive at the default decision value. This will give a contradiction, completing the proof.

A processor p is *hidden* in level r, $1 \leqq r \leqq t-b+2$, in a run $H \in C$ as follows:

(1) In level $t-b+2$, $p \in S(t-b+2)$ and all outedges from p_{t-b+2} are directed to level $t-b+3$ nodes representing processors in $S(t-b+2)$.

(2) In level r, $1 \leqq r < t-b+2$, $p \in S(r)$ and all outedges of p_r are directed to nodes in level $r+1$ representing hidden procesors in $S(r+1)$.

[4] There is actually one correct run for each distinct input value. Our proofs do not make use of this fact, so we let γ denote the correct run for any arbitrary input value

Intuitively, a processor is hidden in a round if the information it possesses at the start of that round does not disseminate to all other processors before the termination of the protocol. We let σ denote an arbitrary run where the sender is hidden in level 1.

The following lemma shows that for any run in C, we may modify inedges of hidden processors without affecting the result of a reliable broadcast.

Lemma 1. *For any run $H \in C$, and any set of processors P in H such that all $p \in P$ are hidden in level r, there exists a run H', equivalent to H, with any single level $r-1$ edge e_p to each $p \in P$ added, removed or with a different label. (However an edge may only be removed if its source has at least b remaining outedges.) H' is identical to H in levels 1 through $r-1$, apart from the edges e_p.*

Proof. We construct a downward induction on r, the number of levels. For the base case, we may add, remove or change the label of any single inedge to any hidden processor in the last level without changing the run as seen by any other processor. Thus this is the required equivalent run H'. Now assume we can do this for all levels $r+1$ through $t-b+2$. We modify the edges one at a time. If we alter, add or remove an edge from level $r-1$ to a hidden processor in level r, we replace its outedges in level r either with the same edges with a new label or with no edges at all. However, in either case we are affecting inedges of hidden processors in level $r+1$, so we apply the induction hypothesis. Thus the final run H' will be equivalent to H. □

The following lemma formally shows the effect of a hidden sender.

Lemma 2. *In the run where the sender is hidden in level 1, all correct processors must accept the default value. In other words, $F(p\sigma)=\delta$ for all correct p.*

Proof. Since the sender is hidden in level 1, only processors in $S(r)$, for any r, may have inedges labeled with values from the message domain. Any processor not in $S(r)$ is correct, and thus must have $F(p\sigma)=\delta$. By RBA, all correct processors have $F(p\sigma)=\delta$. □

The following lemma is the core of the proof. It shows how we may transform any run in C to correct or hide any processor through graph transformations, and preserve equivalence.

Lemma 3. *For every run $H \in C$, for any processor p at any level r, $1 \leqq r \leqq t-b+2$:*

(1) There exists a run $H' \in C$, equivalent to H, such that:
- *(a) H' is identical to H in levels 1 through r, apart from outedges from p_r.*
- *(b) Processor p is correct in H' during and after level r.*
- *(c) All processors are correct after level r in H'.*

(2) There exists a run $H' \in C$, equivalent to H, such that for any legal assignment to $S(r)$:
- *(a) H' is identical to H in levels 1 through r, apart from level r outedges from nodes representing processors in $S(r)$.*
- *(b) All processors in $S(r)$ are hidden at level r in H'.*

Proof. We prove the lemma by downward induction on r, the number of levels.

Base case. Let $r=t-b+2$. For part (1), consider a faulty processor at level $t-b+2$, missing some level $t-b+2$ outedges specified by the action function A. If we add these edges, one at a time, at each step there will be correct processors that see the same local run (i.e., the run restricted to that processor) afterwards as before. Therefore the final run H' will be equivalent to H. For part (2), consider any legal assignment to $S(t-b+2)$. We wish to hide all processors in this set. We consider each processor, one at a time, and remove level $t-b+2$ outedges, one at a time. At each point, there will be correct processors which see the same local run afterwards as before. We continue this process until no processor in $S(t-b+2)$ has a level $t-b+2$ outedge to a processor not in $S(t-b+2)$. Therefore the final run H' is equivalent to H, and all processors in $S(t-b+2)$ are hidden.

Induction case. Given the hypothesis for all levels greater than r, we show how to construct the required H' for conditions 1 and 2 as follows:

(1) Here we are trying to correct an arbitrary processor p in level r.

(1.1) Correct all processors in levels $r+1$ through $t-b+2$. We can do this by (1) of the induction hypothesis.

(1.2) There are now a maximum of $S(r)$ faulty processors in the run, since any processor that became faulty after level r has been corrected. Since $|S(r)|<|S(r+1)|$, there is at least one correct processor q in $S(r+1)$.

(1.3) For a faulty processor p proceed as follows:

If p is missing *all* its outedges then

(1.3.1) Hide any b processors at round $r+1$. We can do this by the induction hypothesis and the fact that $b<|S(r)|$.

(1.3.2) Add b edges from p_r to the newly hidden processors in round $r+1$, by Lemma 1.

(1.3.3) Correct all processors in round $r+1$, by the induction hypothesis.

While there is a level r outedge of p_r missing:

(1.3.4) Select a missing outedge e (with respect to the action function A) of p_r directed to p'_r.

(1.3.5) Let $S'(r+1)=(S(r+1)-\{q\})\cup\{p'\}$.

(1.3.6) Hide p' at level $r+1$, by (2) of the induction hypothesis.

(1.3.7) Insert the edge e into the graph, by Lemma 1.

(1.3.8) Correct all processors at level $r+1$, by (1) of the induction hypothesis. The resulting run is a member of C, since we can use $S'(r+1)$ instead of $S(r+1)$. Note that by a similar argument to (1.2) above, there is still a correct processor q in $S'(r+1)$. Thus we can continue this process until p_r has all its outedges.

(1.4) Processor p is correct in the resulting run since we have added all missing edges, and all processors are correct after level r. It thus satisfies part (1) of the induction hypothesis.

(2) Here we want to hide all the processors in $S(r)$ at level r.

(2.1) Correct all processors at level $r+1$, by part (1) of the induction hypothesis.

(2.2) Hide all processors in $S(r+1)$ at level $r+1$, by part (2) of the induction hypothesis.

(2.3) Add any missing edges from every $p\in S(r)$ to any $p'\in S(r+1)$, by Lemma 1.

(2.4) Correct all processors at level $r+1$, by part (1) of the induction hypothesis.

(2.5) Now all processors in $S(r)$ have at least b edges to processors in $S(r+1)$ and all processors are correct after level r. The following transformations remove all edges from all $p\in S(r)$ to any $p'\notin S(r+1)$. We generate a series of equivalent runs $H_0H_1\ldots H_j$ with associated failure sequences $S_0S_1\ldots S_j$, where $H_0=H$ and $S_0=S$. We note that since we do not change the failure behavior of any run before level r, and only of processors in $S_0(r)$, we have $S_0(r)=S_1(r)=\ldots=S_j(r)$.

(2.6) For each $p\in S_0(r)$, for each edge e from p directed to $p'\notin S_0(r+1)$:

(2.6.1) Since all processors are correct after level r, there is some $q_i\in S_i(r+1)$ that is correct.

(2.6.2) Let $S_{i+1}(r+1)=(S_i(r+1)-\{q_i\})\cup\{p'\}$.

(2.6.3) Hide $S_i(r+1)$ at level $r+1$, by the induction hypothesis. Now p' is hidden.

(2.6.4) Since e is now directed at a hidden processor, we may remove it by Lemma 1.

(2.6.5) Correct all processors in $S_{i+1}(r+1)$ by the induction hypothesis. We now have H_{i+1}.

(2.7) Let $S'_j(r+1)=S_j(r)\cup\{q_0\}$. This is clearly a possible value for H_j's failure sequence since all processors are correct in level $r+1$.

(2.8) Hide all processors in $S'_j(r+1)$ at level $r+1$, by the induction hypothesis.

The resulting run satisfies part (2) of the induction hypothesis. All processors in $S(r)$ now only communicate with processors in $S'(r+1)$ that are hidden. Thus all processors in $S(r)$ are hidden. □

Theorem 3. *In a network with broadcast degree b, there exists no protocol that achieves reliable broadcast in fewer than $t-b+3$ rounds, where t is the maximum number of omission faulty processors.*

Proof. Given any run $H\in C$, we may apply Lemma 3 repeatedly to transform it to an equivalent run in which the sender is hidden. In this run, by Lemma 2, the value accepted by the correct processors must be δ. However, $\gamma\in C$ and $F(p\gamma)=v$, by RBA. Thus we cannot achieve the agreement requirement in C, where processors are only permitted $t-b+2$ rounds of communication. □

8.3 Lower bound proofs for other fault models

We now consider the effects of slightly weakening our fault model to allow other types of failures. We show that a very slight weakening of the fault model leads to a large increase in the inherent time complexity of the problem. More specifically, if faulty processors may omit to *receive* messages, as well as send them, then the lower bound for RB becomes $t+1$ rounds, even in a broadcast environment. An intuitive justification for this result goes as follows. If a network has broadcast degree b, where $b<t$, faulty processors must send at least b copies of every message. However, if other faulty processors may omit to receive the message, there may be circumstances where a message will only be received by one other processor, and the network will be logically reduced to a point-to-point network. We formalize this line of reasoning in our

next theorem. Since the model and reasoning we use are very similar to the previous theorem, we simply give the changes to the model and outline the proof of the theorem.

A run is defined exactly as before, with one addition. Each edge in the graph is colored *green* or *red*. A green edge indicates that the target of the edge received the message. A red edge indicates that the target of the edge omitted to receive the message. Thus, only faulty processors have red inedges, but any processor may have a red outedge. All action and decision functions are applied to *green* inedges only, and action functions may *not* depend on the color of outedges.

We now define the set of runs we will consider, the *critical run set*. As before, we consider runs where a value transmitted by a faulty sender can be kept from the correct processors for as long as possible. We construct a scenario where one new processor per round commits a send omission failure. In the early rounds, under this scenario a partial broadcast would reach a correct processor. So we augment the send omission failures with enough receive omission failures to prevent this from happening. Formalizing, we consider all runs of depth t, satisfying RBA and RBV, with which we can associate a sequence of processors $S(1)\ldots S(t)$ such that:

(1) $|S(r)|=b$ if $r<b$, $|S(r)|=r$ otherwise
(2) $S(r)\subset S(r+1)$, for all r such that $1\leqq r<t$.
(3) If processor p has failed by level r, then $p\in S(r)$.
(4) No more than r processors commit send omission failures by level r (i.e., have a missing outedge with respect to the action function).
(5) Any processor that commits a send omission failure does not commit a receive omission failure.
(6) At most $b-r-1$ processors commit receive omission failures in level r, for $r<b-1$. No processor commits receive omission failures in level r for $r\geqq b-1$.

We call all runs satisfying these criteria the set C'.

We now define what it means for a processor to be hidden in a set of runs in C'. A processor p is hidden in level r in a run $H\in C'$ as follows:

(1) In level t, all faulty processors that only have outedges to processors in $S(t)$ are hidden.
(2) A processor is hidden in level r, where $1\leqq r<t$ iff all *green* outedges from the processor in level r are directed to processors in $S(r+1)$ that are hidden in $S(r+1)$.

Using the methods of the previous section, we may now prove that all runs in C' are equivalent. We omit the proof due to its detail and similarity to the previous proof. Since C' includes both a run where the sender is hidden, and a run where no processors fail, this leads us directly to:

Theorem 4. *In a network where up to t processors are subject to general omission failures, with broadcast degree b, $b<t$, there exists no protocol that achieves reliable broadcast in fewer than $t+1$ rounds.*

Finally we consider the case where the diameter of the network is greater than one. Again we show that the protocol previously developed in Sect. 6 is optimal.

Theorem 5. *In a broadcast network with broadcast degree b and survivor diameter d, where $b<t$, RB cannot be achieved in $t+d-b+1$ phases in the presence of t omission faulty processors.*

Proof outline. We prove by induction on d. If $d=1$, the result follows from Theorem 4. Assume the result holds for $d-1$. Now consider an environment of diameter d, denoted E_d. Given any RB protocol for this environment we divide the processors in any run of this protocol into two sets: Those processors that are up to $d-1$ phases away from the sender, denoted P and the rest, denoted P' (all one phase from some member of P). Now by the induction hypothesis, an RB protocol among the processors in P will take $t-b+d+1$ phases in some cases. Therefore, some processors in P may not receive a transmitted value until this phase. Any processors in P' that are only connected to such processors in P will not receive the value until phase $t-b+d+2$. A full proof can be given using the methods of Sect. 8.2. □

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 special cases 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 con-

nected point-to-point networks and full broadcast networks. In both cases, our protocol matches the established execution time lower bounds for RB [14, 10, 2]. We were able to extend the basic round-based protocol to incorporate arbitrary network survivor diameter, clock skew and tolerate timing failures through stepwise refinement. This methodology not only allows us to argue the correctness of the final protocol in steps, it renders the presentation much more understandable.

Several open questions remain. Firstly, can the techniques used to speed up the execution of RB in conventional point-to-point networks, such as randomized and early-stopping algorithms, be applied in this environment to make our algorithms even faster? Secondly, although 2-round algorithms exist when $b \geq t + n/2$ [2], no algorithm is yet known that can both tolerate arbitrary failures and execute in less than $t+1$ rounds when $t \leq b < t + n/2$. Next, a more careful accounting of the source of failures in broadcast networks may give us new insight into the design of fault-tolerant broadcast networks and reliable broadcast algorithms to execute on them. Finally, the overall reliability of fault-tolerant systems built on these protocols is also open [1].

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.

Acknowledgements. We are grateful to Mohamed Gouda and the referees for their valuable comments.

References

1. Babaoğlu Ö (1987) On the reliability of consensus-based fault-tolerant distributed computing systems. ACM Trans Comput Syst 5(4)
2. Babaoğlu Ö, Drummond R (1985) Streets of Byzantium: network architectures for fast reliable broadcast. IEEE Trans Software Eng SE-11(6):546–554
3. Babaoğlu Ö, Drummond R (1987) (Almost) no cost clock synchronization. Proc 17th Symp Fault Tolerant Comput, Pittsburgh, Pennsylvania (July 1987) pp 42–47
4. Broder A, Dolev D, Fischer M, Simons B (1984) Efficient fault tolerant routings in networks. Proc 16th ACM Symp Theory Comput, pp 536–541
5. Birman KP, Joseph T (1987) Reliable communication in the presence of failures. ACM Trans Comput Syst 5(1):47–76
6. Bracha G (1985) An $O(\lg n)$ expected rounds randomized Byzantine Generals protocol. Proc 17th ACM Symp Theory Comput, Providence, Rhode Island (May 1985) pp 316–326
7. Chang JM, Maxemchuck NF (1984) Reliable broadcast protocols. ACM Trans Comput Syst 2(3):251–273
8. Cristian F, Aghili H, Strong R, Dolev D (1985) Atomic broadcasts: From simple message diffusion to Byzantine Agreement. Proc 15th Symp Fault Tolerant Comput, Ann Arbor, Michigan (June 1985) pp 200–206
9. Dolev D, Reischuck R, Strong HR (1982) 'Eventual' is earlier than 'immediate'. Proc 23rd Symp Foundat Comput Sci, Chicago, Illinois (November 1982) pp 196–203
10. Dolev D, Strong HR (193) Authenticated algorithms for Byzantine Agreement. SIAM J Comput 12(4):656–666
11. Fischer M (1983) The consensus problem in unreliable distributed systems (A Brief Survey). Tech Rep YALEU/DCS/RR-273, Dept Comput Sci, Yale University, New Haven, Connecticut (June 1983)
12. Fischer M, Lynch N (1982) A lower bound for the time to assure interactive consistency. Inf Proc Lett 14(4):183–186
13. Garcia-Molina H, Pittelli F, Davidson S (1984) Applications of Byzantine Agreement in database systems. Tech Rep TR 316, Princeton University, Princeton, New Jersey (June 1984)
14. Hadzilacos V (1984) Issues of fault tolerance in concurrent computations. Ph.D Thesis, Tech Rep TR-11-84, Aiken Computation Laboratory, Harvard University, Cambridge, Mass (June 1984)
15. Halpern JY, Simons B, Strong HR, Dolev D (1984) Fault-tolerant clock synchronization. Proc 3rd ACM Symp Principles Distributed Comput, Vancouver, B.C., Canada (August 1984) pp 89–102
16. Kim W (1984) Highly available systems for database applications. ACM Comput Surv 16(1):71–98
17. Lamport L (1984) Using time instead of timeout for fault-tolerant distributed systems. ACM Trans Prog Lang Syst 6(2):254–280
18. Lamport L, Melliar-Smith PM (1984) Byzantine clock synchronization. Proc 3rd ACM Symp Principles Distributed Comput, Vancouver, B.C., Canada (August 1984) pp 68–74
19. Lamport L, Shostak R, Pease M (1982) The Byzantine Generals problem. ACM Trans Prog Lang Syst 4(3):382–401
20. Metcalfe R, Boggs DR (1976) Ethernet: Distributed packet switching for local computer networks. Commun ACM 19(7):396–403
21. Moses Y, Tuttle M (1986) Programming simultaneous actions using common knowledge (Algorithmica, to appear). Preliminary version available in Proc 27th Ann IEEE Symp Foundat Comput Sci (October 1986) pp 208–221
22. Rabin M (1983) Randomized Byzantine generals. Proc 24th Symp Foundat Comput Sci, Tucson, Arizona (November 1983) pp 403–409
23. Strong HR, Dolev D (1983) Byzantine agreement. Digest of Papers, Spring Compcon 83, San Francisco, California (March 1983) pp 77–81
24. Schneider FB, Lamport L (1982) Paradigms for distributed programs. In: Paul M, Siegert HJ (eds) Distributed Systems: Methods and Tools for Specification. Lect Notes Comput Sci, vol 190
25. Spector AZ (1984) Computer software for process control. Scientific American 251(3):174–187
26. Srikanth TK, Toueg S (1985) Optimal clock synchronization. Proc 4th Symp Principles Distributed Comput, Minaki, Canada (August 1985) pp 71–86
27. Stallings W (1984) Local networks. ACM Comput Surv 16(1):3–41
28. Svoboda L (1984) Resilient distributed computing. IEEE Trans Software Eng SE-10(3):257–268
29. Tanenbaum A (1981) Computer Networks. Prentice Hall, Englewood Cliffs, NJ