Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts*

Özalp Babaoğlu
Rogerio Drummond

TR 85-613
June 1984
(Revised February 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 and the CAPES Agency of the Government of Brazil through a Fellowship to the second author.

Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts

ÖZALP BABAOĞLU and ROGÉRIO DRUMMOND

Abstract—A site broadcasting its local value to all other sites in a fault-prone environment is a fundamental paradigm in constructing reliable distributed systems. Time complexity lower bounds and network connectivity requirements for reliable broadcast protocols in point-to-point communication networks are well known. In this paper we consider the reliable broadcast problem in distributed systems with broadcast networks (for example Ethernets) as the basic communication architecture. We show how properties of such network architectures can be used to effectively restrict the externally visible behavior of faulty processors. We use these techniques to derive simple protocols that implement reliable broadcast in only two rounds, independent of the failure upper bounds.

Index Terms—Fault-tolerance, distributed computing, Byzantine agreement, network partitions, Ethernet.

Partial support for this work was provided by the National Science Foundation under Grant MCS 82-10356 and the CAPES Agency of the Government of Brazil through a Fellowship to the second author.

A preliminary version of this paper was presented at the Fourth IEEE Symposium on Reliability in Distributed Software and Database Systems, October 1984, under the title "Communication Architectures for Fast Reliable Broadcasts."

The authors are with the Department of Computer Science, Cornell University, Ithaca, New York 14853. The second author is on leave from the Computer Science Department, State University of São Paulo at Campinas, Brazil.

1. Introduction

A *distributed computing system* consists of a set of autonomous processors (sites) that can communicate through a network. To cooperate, these processors must periodically coordinate their activities. One way to do this is by the use of *broadcasts*—an operation that allows a processor to communicate a local value to all other processors in the system. Since fault-tolerance is one of the primary motivations for constructing a distributed system, the semantics of a broadcast operation have to accommodate failures. In the systems we consider, both processors and communication components can fail and are said to be *faulty* after their first failure. We define the *reliable broadcast* (RB) problem in an unreliable environment as follows.

> **Reliable Broadcast.** A distinguished processor, called the *transmitter*, broadcasts a local value (drawn from a domain of at least two elements) such that the following two requirements are satisfied:
>
> *Unanimity.* All non-faulty processors decide on the same value.
>
> *Non-triviality.* If the transmitter is non-faulty, then all non-faulty processors decide on the value that was broadcast by the transmitter.

The abstract formulation of this problem was first studied by Pease *et al.* [18] under the name *Interactive Consistency* (later called *Byzantine Agreement* [14]) and subsequently by numerous other researchers (see [6] for a brief survey).

The reliable broadcast has been identified as the appropriate primitive in solutions to a large number of problems [10]. These include distributed consensus [6], distributed synchronization [20], replicated data update (e.g., distributing updates to routing tables in a packet switching network) [9], and transaction management in database systems [17].

The properties of the environment in which RB is studied have profound implications on the nature of the solution. In fact, Fischer *et al.* [8] have shown that the problem has no deterministic solution in systems where failures cannot be detected and message delivery times cannot be bounded. In this paper, we will be concerned only with deterministic protocols in synchronous systems. We define such systems in the next section.

Most of the previous formulations of the RB problem assume a reliable point-to-point network as the communication architecture. In such a system, let t denote the maximum number of processors that can can be faulty during the execution of a protocol solving RB. Lower bounds for the total number of processors participating in the protocol, and time and message complexity of the protocol have been obtained as functions of t. It is customary to measure time complexity in units called ***rounds*** consisting of message exchange and computation steps, such that during a round every processor can communicate with all others in the system. The messages sent by a processor in a given round cannot depend on the messages it receives during the same round. Of particular importance is the result that at least $t+1$ rounds are required for any solution to the RB under even the most restrictive failure modes and unconstrained amounts of information exchange [7, 4, 11].

In this paper, we dispel the belief that RB is an inherently expensive primitive by breaking the $t+1$ round time barrier. We show that by selecting communication architectures that can restrict the external behavior of faulty processors, RB can be implemented in two rounds. We argue that our assumptions about the behavior of the communication network are reasonable in the environments where RB is applicable as a primitive in the first place — highly integrated distributed systems built around local area networks.

2. Motivation

Arbitrary processor failures in a distributed system using a point-to-point communications network are troublesome for several reasons. First, a faulty processor can send conflicting information along disjoint routes to other processors. Secondly, unless the communication graph is fully connected, faulty processors on the route of a message can tamper with it. Finally, faulty processors can generate spurious messages and pretend to be forwarding them on behalf of other processors.

The visible external behavior that faulty processors may exhibit can be effectively restricted through the use of certain mechanisms. For example, if the system supports an authentication mechanism such as digital signatures [3], faulty processors are prevented from altering messages and from spontaneously generating spurious messages. This restriction on the behavior of a faulty processor results in algorithms that are simpler and that can tolerate more faults than their counterparts without authentication [4]. In what follows, we propose communication architectures as mechanisms to further restrict the behavior of faulty processors.

Our motivation is based on the following observation. If we can design a communication network architecture such that

(i) there exists a single route between any given processor and all other processors, and

(ii) messages never need to be forwarded (all processors are effectively immediate neighbors of each other),

then the behavior of a faulty processor can be limited. In particular, designs based on such architectures need not worry about message tampering and confounding through conflicting messages. *Broadcast networks*, such as ALOHA [1], DCS [5] and Ethernet [16] come very close to realizing these properties.

We pursue this approach through the following organization. The next section is a description of the global system properties and classification of the processor failures that can occur. In Section 4, the major communication network components are identified and their failure behaviors discussed. In Section 5 we introduce the class of *R-redundant* broadcast networks. Sections 6 and 7 contain our main results—protocols to solve RB along with proofs of correctness in R-redundant broadcast networks. Network partition failures are examined in Section 8. Finally, we extend our results to accommodate arbitrary failure behavior of the communication components. A comparison of our results with those of point-to-point communication networks concludes the paper. We note that other researchers have also proposed using broadcast networks to implement reliable broadcasts however under much stricter failure assumptions [2, 15].

3. Failure Modes and System Properties

In a distributed systems, the only visible behavior of a processor is the sequence of messages it sends. Consequently, the various failure modes of processors (with respect to a given protocol) can be classified as follows:

> *Crash Fault.* A faulty processor stops sending messages after some arbitrary time, never to resume again.
>
> *Omission Fault.* A faulty processor omits sending some of the messages prescribed by its protocol. The messages it does send are always correct.
>
> *Malicious Fault.* A faulty processor exhibits arbitrary behavior. Such a processor can withhold messages that it is required to send, send messages not prescribed by its protocol, and even collude with other (faulty) processors to disrupt the system.

A *non-faulty* processor sends exactly those messages that are prescribed by its protocol—no more and no less. Clearly, a protocol that can tolerate Malicious Faults can also tolerate either Omission or Crash Faults. Similarly, a protocol that can tolerate Omission Faults will tolerate Crash Faults.

We assume that the distributed system under consideration possesses the following properties:

> *Authentication Assumption.* Messages can be signed using digital signatures [3].
>
> *Network Assumption.* In the absence of failures, the communication system guarantees bounded delivery times for messages to all processors.
>
> *Clock Assumption.* Non-faulty processors maintain local times that do not differ by more than a know constant and are within a known constant around (unobservable) absolute time.

In systems composed of homogeneous processors that are in close physical proximity (as they are in local area networks), these assumptions seem quite reasonable. Halpern *et al.* [12] have developed an efficient algorithm to achieve the clock assumption in the presence of faulty processors as long as the first two assumptions hold. Recently, Srikanth and Toueg [21] have shown that clock synchronization can be achieved without authentication as long as only one third of the total number of processors can be faulty. In a future paper, we will show how clock synchronization can be realized efficiently and without authentication by exploiting the properties of the networks we are about to propose. Without loss of generality, we will assume that clocks are perfectly synchronized so that we can think of the distributed computation proceeding in lock-step rounds.

Algorithms for approximating the Authentication Assumption are well known. Depending on the degree of maliciousness that processors may exhibit, messages can be augmented with checksums, cyclic redundancy checks, or digital signatures based on public-key cryptography techniques [19].

4. Broadcast Networks

Broadcast networks have become a popular topology for local area networks. Some of the reasons for this popularity are simplicity, high reliability, high performance and very low cost. Perhaps the best known example of this class of networks is the Ethernet. Figure 1 shows a typical distributed system configuration based on a broadcast network.

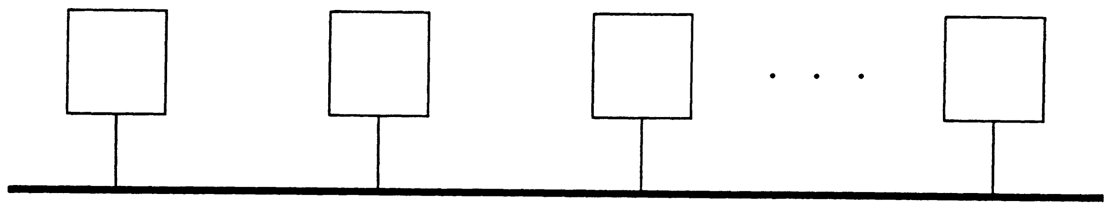

FIG. 1. *Broadcast network.*

The central component of a broadcast network is the *channel.* Typically, this is a passive transmission medium such as a coaxial cable, fiber-optic cable, or simply space (in the case of a radio broadcast network). Processors are connected to the channel through *links.* Links contain active components such as signal transceivers, network interface units, and buffer memory. Access to the channel by the links is controlled through a multiple-access transmission protocol [22]. We note that collision-free transmission protocols exist for multiple-access channels that can guarantee the Network Assumption [22]. Even with transmission protocols where collisions may occur (such as in Ethernet), the message delivery time can be bounded with high probability by keeping the channel lightly loaded.

The fundamental communication operation implemented by a broadcast network is **broadcast**(v), where v is the message being communicated. Through the Authentication Assumption, a processor receiving a message is able to reliably identify the broadcaster under any possible processor failure mode. We abstract the operation of the channel and the links as follows:

Non-faulty Channel Semantics. In the absence of failures, the channel arbitrarily selects one of the links to listen to and simply echoes the incoming message (if any) to all of the other links.

Faulty Channel Semantics. A fault in the channel causes some of the incoming messages to be lost (i.e., echoed to none of the other links).

Non-faulty Link Semantics. In the absence of failures, the link simply transfers messages between the channel and its associated processor.

Faulty Link Semantics. A fault in the link causes some arbitrary incoming messages, either from the processor or the channel, to be lost.

Broadcast networks that exhibit these behaviors can be readily implemented. We further require that neither a faulty channel nor a faulty link can spontaneously generate (valid) messages. This requirement is usually achieved by standard transmission protocols.

As defined, our channel semantics ensure that either all (in case the channel is non-faulty) or none (in case the channel is faulty) of the processors connected to a given channel through non-faulty links receive the broadcast message. There are rare channel failures that result in the partitioning of the links (and thus processors) such that some of them receive a broadcast message and others do not. We consider these types of failures in Section 8.

To simplify our presentation, we will assume that message authentication and validation (using the appropriate mechanisms) are performed at a level below the RB protocol. Messages that are invalid are simply discarded and are never presented to the RB protocol level. If no message is delivered to the RB protocol level after a time interval equal to the network delivery time bound, we say that a null message (denoted as ϕ) is received.

Given that processors communicate through a network with the above properties, we note that there is only one externally visible system behavior for any possible failure scenario.

> *Broadcast Network Failure Property:* In response to a broadcast, those processors that receive a (non-null) message receive the same message.

This behavior is ensured even if processor failures are malicious, since it is physically impossible for a processor to send conflicting messages to different processors in the same broadcast. In other words, we have succeeded in restricting the visible affects of processor failures.

We now discuss the power of this property when applied to solve the RB problem in the presence of processor, link and channel failures.

5. Redundant Broadcast Networks

Even with the Broadcast Network Failure Property, the **broadcast**(v) operation alone does not achieve the unanimity requirement for RB since a single broadcast network cannot guarantee 1-connectivity† among processors in the presence of communication faults. To increase connectivity, we propose an *R-Redundant Broadcast Network* generated by replicating each communication component R-fold. As shown in Figure 2, the construction requires R independent broadcast networks, where each of the processors is connected to each of the R channels through an independent link. In such a network we account separately

† Since no message forwarding ever takes place in the networks we consider, we are only concerned with *edge connectivity*. We say that a network has *k-connectivity* if there exist at least k disjoint paths consisting of links and channels between every pair of processors.

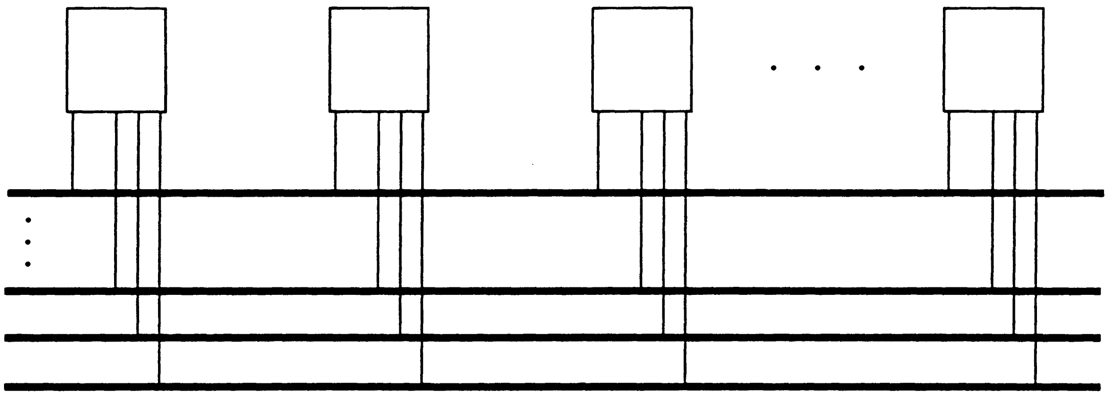

FIG. 2. *R-redundant broadcast network.*

the various components that exhibit faults. During some execution of a RB protocol, let

π be the actual number of faulty processors,
λ be the actual number of faulty links, and
γ be the actual number of faulty channels.

We say that a processor is ***properly connected*** to a channel if its link to that channel is non-faulty. By construction, it is trivial to show the following property of R-redundant broadcast networks.

PROPERTY 1. *In an R-redundant broadcast network, the condition $R > \lambda + \gamma$ is sufficient to guarantee that any pair of processors will remain at least 1-connected for the duration of the protocol.*

In the subsequent discussions, we will assume that this property holds without explicitly repeating it.

Since every processor in an R-redundant broadcast network is connected to R channels, we augment the broadcast primitive syntax as follows: **broadcast**(v, i); where v is the message to be communicated, and $i \in \{1,2,...,R\}$

selects a channel. We will use the following shorthand for notational convenience:

$$\begin{aligned}
\textbf{broadcast}\ (v, *) \equiv\ & \textbf{foreach}\ i\ \textbf{in}\ \{1,2,...,R\} \\
& \textbf{do} \\
& \qquad \textbf{broadcast}\ (v,\ i) \\
& \textbf{od;}
\end{aligned}$$

Note that we do not require this operation to be atomic nor do we make any assumptions about the relative order or timing of the broadcasts over the various channels. Furthermore, no assumptions are made about the behavior of faulty processors with respect to messages broadcast over different channels. In particular, a faulty processor can broadcast conflicting messages over different channels. All that we rely on is the Broadcast Network Failure Property to hold for each of the broadcast networks.

6. Reliable Broadcast Protocol for Omission Fault Processors

While Crash Faults are more restrictive failures than Omission Faults, it is interesting to note that restricting processors and network components to exhibit Crash Faults does not render the RB problem any easier. In fact, Hadzilacos [11] has shown that there exists a transformation procedure for converting any solution to the RB for Crash Fault components into an equivalent solution (in the sense of having the same time and information complexities) for Omission Fault components. Therefore, we will initially consider processors that fail through Omission Faults.

Round 1: For p_1 (the transmitter with local value $v \in \mathbf{V}$)

```
        broadcast(v, *);
        decide on v ;
```

Round 2: For all p_i, $i \neq 1$

```
        if |C_i^1| ≠ 0 then
                m_i is the (unique) message received in round 1
                foreach k_i in {1,2,...,R} - C_i^1 do
                        broadcast(m_i, k_i);
                od;
                decide on m_i ;
        fi;
```

At the conclusion of round 2: For all p_i

```
        if not yet decided then
                if |C_i^2| ≠ 0 then
                        m_i is the (unique) message received in round 2
                        decide on m_i ;
                else
                        decide on δ;
                fi;
        fi;
```

FIG. 3. *Protocol P1—Reliable broadcast for omission fault processors.*

Let $p_1, p_2, \ldots, p_N$ denote processors and assume p_1 is the transmitter. Let δ denote a default value known *a priori* to all of the processors such that all of the non-faulty ones will decide on δ in case the transmitter is deemed faulty. Let $\mathbf{V}$ denote the domain of all possible valid message values not including ϕ. We say that a RB protocol is *symmetric* if the computation carried out by each processor is independent of its name, with the exception of the transmitter which is distinguished from the other processors. We do not consider the message authentication steps, which can be processor name dependent, to be part of the RB protocol. Each processor executes the protocol of Figure 3. Note that all variables are subscripted with the processor name only to denote that they are local variables. Consequently, the protocol is symmetric according to our definition. Recall that the values broadcast in a given round cannot depend on messages received in that round. In the protocol, the notation $\mathbf{C}_i^r$ denotes the set of channels from which processor p_i received the (non-null) message $m_i \in \mathbf{V}$ at round r. Note that message m_i is unique since processor failures are restricted to Omission Faults.

The following theorem establishes the correctness of the protocol.

THEOREM 1. *If* $N \geq \lambda + \pi$, *then protocol P1 implements RB for processors that may fail according to Omission Faults.*

Proof: We proceed by case analysis.

If the transmitter is non-faulty, all processors receive the broadcast in round 1 (and know that it is from the transmitter) and decide on v at round 2. Thus, non-triviality and unanimity are assured. The following assure unanimity in cases of transmitter failure.

If the transmitter is faulty and broadcasts no messages over non-faulty channels to which it is properly connected in the first round, by the end of the second round only ϕ messages have been received and all non-faulty processors decide on δ.

Next consider the case where the transmitter is faulty and some non-faulty processor p_i receives the initial broadcast. Since processor failures are restricted to Omission Faults, the non-null messages m_i received by p_i must all be equal to v. The protocol requires p_i to echo m_i through all channels not in $\mathbf{C}_i^1$. Therefore, by the end of the second round, v-messages will have reached all non-faulty channels to which p_i is properly connected. By the 1-connectivity property, it is guaranteed that all processors receive v in one of the two rounds. As receiving a non-null message in either of the two rounds implies a decision, all processors decide on v.

Finally, we consider what happens if the transmitter is faulty and only other faulty processors receive the initial broadcast. This can only occur when all non-faulty processors have faulty links to the non-faulty channels p_1 broadcast over in the first round. The minimum number of link faults necessary by the end of round 1 for this scenario is $N-\pi$ and comes about when p_1 broadcasts over a single non-faulty channel to which it is properly connected and all of the non-faulty processors have faulty links to this channel. Since $N \geq \lambda + \pi$ and $\lambda = N-\pi$, there can be no further faulty links. If one of the faulty processors echoes v during round 2 over a channel other than the one p_1 broadcast over, then all non-faulty processors are guaranteed to receive it and decide on it. Otherwise, all non-faulty processors receive only ϕ messages by the end of round 2 and decide on δ. $\square$

From the protocol description, it is obvious that P1 results in $N \cdot R$ messages to be exchanged and requires two rounds to implement RB. The next result shows the optimality of P1 with respect to time complexity within the class of redundant broadcast networks.

THEOREM 2. *For the class of R-redundant broadcast networks, there exists no symmetric one-round protocol to implement RB in the presence of even non-faulty channels, a single faulty processor, and a single faulty link.*

Proof: Without loss of generality, assume that the transmitter is the faulty processor and fails according to Crash Faults. Under any (one-round) protocol, there must be a step where p_1 broadcasts its local value. Assume that p_1 fails after having broadcast v over k channels $(0<k\leq R)$. As before, $\mathbf{C}_i^1$ denotes the set of channels over which processor p_i received non-null messages from the transmitter by the end of round 1. Note that during the single round, any two processors can also exchange information through broadcasts. However, this information cannot depend on what each one received from the transmitter since this occurs in the same (single) round. Furthermore, since we require that the protocol be deterministic and symmetric, the information content of this communication between the processors is zero. Note that each channel could have a different name with respect to different processors. This fact, along with the requirement that the protocol be symmetric, prevents the decision to be based on the actual elements of the set $\mathbf{C}_i^1$. This leaves the number of non-null messages received by each processor from the transmitter by the end of the round as the only relevant information for deciding. Therefore, for any symmetric one-round protocol, there must exist a common function ψ such that each processor p_i

decides on $\psi(|\mathbf{C}_i^1|)$ at the end of the round. From the non-triviality requirement of RB, we must have $\psi(R)=v$. Furthermore, $\psi(0)=\delta$ simply due to complete lack of global information. Consequently, there must exist some threshold value $z>0$ such that $\psi(z-1)=\delta$ and $\psi(z)=v$. Now, in the above failure scenario, let $k=z$ whereby all proper processors decide on v except for the processor with the single faulty link to one of the k channels which has to decide on δ. We conclude that the unanimity requirement for RB cannot be assured under any symmetric one-round protocol. □

We next consider the RB problem when processor faults are unconstrained.

7. Reliable Broadcast Protocol for Malicious Fault Processors

By far the most important property of a broadcast network is that it ensures unanimity among the receivers of a broadcast in the absence of network failures. It is impossible for a (maliciously) faulty processor to send conflicting values to different processors in a single broadcast over the same channel. Our motivation for the construction of R-redundant broadcast networks was to increase connectivity so that network failures could be tolerated in an effective manner. However, in so doing, we have left the system vulnerable to malicious failures whereby conflicting messages can be sent by a faulty processor over each of the R components of the redundant broadcast network during a single **broadcast**(v,*) operation.

In this section, we present an efficient protocol that implements RB in R-redundant broadcast networks in the presence of Malicious Fault processors. Our assumptions about the behavior of the communication system continue to apply. In terms of fault-tolerant system design, Malicious Fault processors represents a "worst case" design strategy. Even in this case, our protocol requires only two rounds to implement RB. The only additional cost with respect to protocol P1 is an increase in the total number of processors that must participate in the protocol for a given level of fault tolerance.

Before discussing the protocol, we introduce some notation and definitions. Let $\mathbf{W} = \{\phi, \perp\} \cup \mathbf{V}$ denote the domain of message values including the null message and the special error symbol $\perp$. The set of all possible bags (multisets) of size k constructed from the elements of a set $\mathbf{S}$ is denoted as $\mathbf{S}^k$. In the protocol, every non-faulty processor broadcasts at most one message over a channel in a single round. Let $B_i^r(j) \in \mathbf{W}^R$ be the *in-bag* of R messages (one from each channel) processor p_i receives from processor p_j during round r. If a processor p_i receives more than one message over the same channel from some (faulty) processor p_j during some round r, it sets the in-bag $B_i^r(j) = \{\perp\}^R$ immediately regardless of what messages it received (or will receive) from p_j over other channels in the same round.

We say that a bag B is *consistent in v* if and only if $B = \{\phi, v\}^R$ for some unique $v \in \mathbf{W}$ and $v \neq \perp$. In other words, the bag contains copies of either a single non-error message or two messages one of which has to be the null message. The *filter function*

$$\sigma : \mathbf{W}^R \rightarrow \mathbf{W}$$

maps bags of size R to the valid message domain is defined as follows:

$$\sigma(B) = \begin{cases} v & \text{if } B \text{ is consistent in } v \\ \delta & \text{otherwise.} \end{cases}$$

Finally, we define the *decision function* with threshold z

$$\psi^z : \mathbf{W}^N \rightarrow \mathbf{V}$$

that maps bag $M \in \mathbf{W}^N$ of N messages such that

$$\psi^z(M) = \begin{cases} v & \text{if } \rho(M)=v \text{ and } \theta(M,v) \geq z \text{ and } \theta(M,\delta) < z \\ \delta & \text{otherwise} \end{cases}$$

where $\rho(M)$ selects the most frequent non-ϕ element in the bag M and $\theta(M,x)$ counts the number of times element x appears in M.

The protocol for Malicious Fault processors is shown in Figure 4. Let us consider the result of a processor failure during the **broadcast**(v,*) operation in this system. Note that if the failure is such that a processor broadcasts conflicting messages over faulty channels or through faulty links to non-faulty channels, the

---

Round 1: For p_1 (the transmitter with local value $v \in \mathbf{V}$)

  broadcast(v, *);

Round 2: For all p_i

  $m_i \leftarrow \sigma(B_i^1(1))$;
  broadcast(m_i, *);

At the conclusion of round 2: For all p_i

  $M^i \leftarrow \{j=1,2,\ldots,N: \sigma(B_i^2(j))\}$;
  decide on $\psi^z(M^i)$;

FIG. 4. *Protocol P2 with threshold z: Reliable broadcast for malicious fault processors.*

---

failure is hidden from all other processors and the transmitter appears to be non-faulty. It is only when the conflicting messages are visible to other processors (because they reach non-faulty channels) that unanimity can be compromised. With this in mind, we say that a sending processor is *overtly malicious* if the bag of message values as observed on the channels due to a broadcast is not consistent. A receiving processor p_i is called a *witness* to this type of failure of processor p_j in round r if and only if $B_i^r(j)$ is not consistent.

Similarly, a processor exhibits *overt omission* failure if it is not overtly malicious but broadcasts a null message over at least one non-faulty channel to which it is properly connected. In an R-redundant broadcast network, processor p_i *witnessing* an overt omission failure of processor p_j in round r requires that $B_i^r(j) = \{\phi\}^R$.

As we have defined it, the filter function σ is designed to detect exactly these failures. It results in ϕ for the witnesses of overt omission failures and in δ for the witnesses of overt commission failures. We will next present results that bound the number of processors that can be witnesses to the failure of the transmitter and show that under no circumstance can other faulty processors collude to prevent unanimity. This is accomplished by ensuring that a sufficiently large number of non-faulty processors are biased towards the same decision by the end of the first round.

LEMMA 1. *In an R-redundant broadcast network, the number of witnesses to an overt omission failure of the transmitter by the end of the first round in Protocol*

P2 is either $N-1$ or in the range $[0,\lambda]$.

Proof: Let x be the number of non-faulty channels to which p_1 is properly connected and sends non-null messages. Clearly $0 \leq x < R$ since $x = R$ would imply that the transmitter is non-faulty. Since 1-connectivity is ensured, when $x = 0$ all $N-1$ remaining processors witness the failure. If $x > 0$, then for a processor to witness the failure it must have faulty links to all x channels. As there are at most λ actual faulty links, the number of such witness processors is $\lfloor \lambda / x \rfloor$. This results in the range $[0,\lambda]$. □

LEMMA 2. *In an R-redundant broadcast network, the number of witnesses to an overtly malicious transmitter by the end of the first round in Protocol P2 is in the range* $[N-\lambda-1, N-1]$.

Proof: Let p_1 fail by broadcasting two different (non-null) messages over two non-faulty channels to which it is properly connected. For a processor not to witness this failure, it must not be properly connected to one of these channels. Since there are λ possible faulty links, there can be at most λ such processors. Consequently, the number of witnesses (not counting the faulty transmitter) is at least $N-\lambda-1$ and at most $N-1$. □

Recall that t is the upper bound for the number of actual faulty processors, π.

THEOREM 3. *If $N > t + \pi + \lambda$, then protocol P2 with threshold $z = t + 1$ implements RB for processors that may fail according to Malicious Faults.*

Proof: We proceed by case analysis.

If the transmitter is non-faulty, all of the in-bags will be consistent in v by the end of the first round. For all non-faulty processors, the filter function yields v, which is echoed in round 2. The $N-\pi$ such echoes will be filtered by the end of the second round and added as v values to the M^i bag for each processor. For all non-faulty p_i, the bags M^i will contain at least $N-\pi$ occurrences of v and at most π occurrences of some other value. Even if all of the π faulty processors collude and try to force a decision on this other value, it will fail since $\pi \leq t$ and the threshold of the decision function will not be exceeded. The condition $N > t + \pi + \lambda$ can be written as $N-\pi \geq t + \lambda + 1$ and since $\lambda \geq 0$, we have $N-\pi \geq t+1$ so that v will be selected by all of the decision functions of the non-faulty processors. Thus we have shown non-triviality for Protocol P2. We next argue that it also ensures unanimity if the transmitter is faulty.

Assume that the transmitter fails by overt omission. By Lemma 1, there will be $k \in \{0,1,\ldots,\lambda,N-1\}$ witnesses. If $k=N-1$, then at least $N-\pi$ processors will broadcast ϕ in the second round. By an argument identical to that of the previous paragraph, all non-faulty processors decide on δ. If $k \leq \lambda$, then $N-\pi-k$ non-faulty processors do not witness the failure, and for each, the filter function yields the value v since each has an in-bag consistent in v. In the second round, $N-\pi-k$ consistent v messages are broadcast such that for each non-faulty processor p_i, the bag M^i will contain $N-\pi-k$ occurrences of v. Since $k \leq \lambda$, the inequality $N-k-\pi \geq t+1$ is assured by the constraint $N > t + \pi + \lambda$ and the decision function yields v for all non-faulty processors.

Finally, assume that the transmitter is overtly malicious. By Lemma 2, $N-\lambda-1 \leq k \leq N-1$ processors witness the failure. In the second round, at least $k-(\pi-1)$ processors will echo δ messages. By the end of the second round, M^i for each non-faulty p_i will contain at least $k-\pi+1$ occurrences of δ. Since for all possible values of k we have $k-\pi+1 \geq t+1$, the decision function yields δ. □

8. Coping with Channel Partitions

Consider the failure mode of an individual broadcast network that partitions the set of links into two disjoint groups—those that receive a message and those that do not. For example, in an Ethernet, the coaxial cable physically breaking, the failure of the repeater unit between two channel segments and an interface unit causing collisions shorter in duration than the end-to-end propagation delay of the channel are rare instances of partition failures. We now relax the faulty channel semantics to allow these partition failures.

> *Faulty Channel Semantics:* A faulty channel fails to deliver a broadcast message to a subset of the links connected to it. Those links that do receive a message receive the same message.

With respect to a given broadcast, the group of links receiving the message is called the *in-partition*, and the group not receiving it is called the *out-partition.* Note that our definition allows the composition of these partitions to change dynamically from one message to another and from one broadcaster to another. Let us observe the behavior of the communication system from a processor's point of view for a given broadcast. The case where the processor's link is non-faulty but in an out-partion and the case where its link has failed and is in an in-partition are indistinguishable—they both result in the processor receiving a ϕ message. This observation suggests a duality between link and channel failures. We can either assume that channels are failure free and simulate a partition failure by pretending that all of the links in the out-partition have failed regardless of their actual state, or we can assume that links are failure free and simulate their failure by constructing the appropriate out-partitions for channel failures. We will pursue the first alternative.

Now, links can be "faulty" either because they have failed or because they belong to the out-partition of a faulty channel. With this new interpretation, all of our previous results continue to hold for the more general faulty channel semantics. All that is required is that we be able to account properly for the total number of faulty links. Next we characterize each channel according to the configuration of link failures to that channel. For channel i, let λ_i^m denote the number of processors that do not receive a given broadcast message m over that channel. If $\lambda_i^m = 0$ then channel i has not failed. If $\lambda_i^m = N$ then channel i is said to have *failed totally*. For any other value, channel i is said to have *failed partially*. Throughout the execution of a protocol, a channel is non-faulty if it has not failed for any broadcast message. If all of the channel failures are total, then it is said to exhibit a *total fault*†. Any other channel behavior is called a *partition fault*. We let γ' count the number of faulty channels (either total or partition).

† Note that this behavior corresponds to the earlier semantics for faulty channels whereby some messages are lost completely.

We will first examine the implications of the new semantics for faulty channels on the replication factor of our R-redundant broadcast networks. Recall that in such networks, $R > \lambda + \gamma$ is a sufficient condition for 1-connectivity. Clearly, the network can sustain more than $R - \gamma$ faulty links and still preserve 1-connectivity if there are multiple faulty links to the same channel. Since we do not allow any message forwarding by processors, 1-connectivity can be assured if there is at least one non-faulty channel with all non-faulty links. According to our new notation, 1-connectivity is guaranteed if $R > \gamma'$.

For each channel, we are interested in the maximum number of processors that do not receive some message after a broadcast. Let λ_i be the maximum of λ_i^m over all messages m broadcast over channel i during the execution of the protocol such that $\lambda_i^m \neq N$. If the channel exhibits a total fault, then we define λ_i to be zero for that channel. Note that this definition results in $\lambda_i = 0$ for both channels that fail totally and that are non-faulty. Under both circumstances, all processors receive the same message (either ϕ or what was broadcast) and the N "faulty" links need not be accounted (except for connectivity reasons). Let λ' be the sum of λ_i over all channels. With this new interpretation, all of our previous results can be extended to channel partition failures simply by replacing all occurrences of λ with λ'. In essence, λ' is an upper bound on the total number of link failures that can occur in each round. Note that the number of links that fail at least once during the execution of the protocol can be much larger than λ'.

For certain network configurations, we can tolerate even more link faults. These configurations are characterized as permitting a bound on the λ_i for each channel. For example, consider a broadcast network consisting of several Ethernet segments connected through repeaters. If repeater failures are the only sources of channel partitions, and the number of processors connected to each segment is known, we can easily express the minimum number of processors that will receive every message broadcast over that channel. Since λ_i denotes the maximum number of processors that do not receive some message over channel i, $N-\lambda_i$ denotes the minimum number of processors that receive every message over the same channel. If $N-\lambda_i > \pi$ can be ensured for each channel, we know that at least one non-faulty processor will receive each message broadcast over each channel. For Omission Fault processors, this condition is sufficient to guarantee that protocol P1 continues to implement RB in two rounds.

In summary, if the faulty links that simulate partitions are well-structured (as is the case with multiple Ethernet segments and repeaters), our protocols can sustain many more of them than what is allowed by the pessimistic accounting through λ'. As an example, Theorem 1 requires that the total number of faulty links not exceed $N-\pi$ for protocol P1 to work. However, if $\lambda_i < N-\pi$ for each channel, the system can tolerate almost $R-1$ times as many faulty links.

9. Malicious Communication Component Failures

The failure mode for the links and channels we have examined so far can be characterized as Omission Faults. In this section we examine the consequences of network component failures that can undetectably alter the contents of messages.

Given that the communication media can alter messages, receiving a message over a single channel does not permit the processor to infer the actual message broadcast over that channel. Let $G > \gamma'$ be an upper bound on the number of faulty channels. If the we ensure that $R > 2G$ and each processor communicates through **broadcast**(v, $*$), we can overcome the behavior of faulty communication components. If a non-faulty processor broadcasts the message v, then all processors will receive at least $G+1$ copies of v since at most G copies could have been altered by faulty paths. Since there are more non-faulty paths between processors than faulty ones, receivers can always determine the actual broadcast message by simply selecting the majority element from the bag of received messages. Having guaranteed effective communication through this mechanism, the subsequent discussion will assume channels that are failure free. We can do this without loss of generality since malicious channel failure behavior can be simulated through the failure of links.

We modify slightly our previous notion of consistency for bags. A bag B is said to be ***consistent in v*** if and only if v is a valid message and it is the majority element. A bag that is not consistent in any value is called ***inconsistent***.

Let B^* be the bag of messages as observed on the channels due to the execution of a broadcast by some processor over the R channels. We will now prove that protocol P2 achieves RB under malicious processor and link faults.

LEMMA 3. *If B^* is consistent in v, then* *(i)* *at most λ' processors may receive an inconsistent message,*

(ii) *at most λ' processors may receive a consistent message in some value y different from v,*

(iii) *at least $N-\lambda'$ processors receive a message consistent in v.*

Proof: For the first two cases, a faulty link to a processor could turn a bag consistent in v observed over the channels into an inconsistent bag or a bag consistent in some other value y as seen by the processor. In the worst case, there can be λ' such processors since there are at most λ' link failures in each round. As a consequence, at least $N-\lambda'$ processors have non-faulty links and receive the bag of messages consistent in v. □

LEMMA 4. *If B^* is inconsistent, then at most λ' processors receive a bag of messages consistent in some value.*

Proof: Trivially follows from the observation that a change to a single element of an inconsistent bag could make it consistent. □

Given these two Lemmata, we next show how protocol P2, with the new definition of consistency, can be adapted to support arbitrary communication failures. Let L denote the upper bound for λ' for all executions of the protocol.

THEOREM 4. *If $N > t + L + \pi + \lambda'$, then protocol P2 with threshold $z = t + L + 1$ implements RB when processors and communication components may fail according to Malicious Faults.*

Proof: If the transmitter is non-faulty, the bag of messages observed over the channels resulting from its initial broadcast is consistent in v. By Lemma 3 at least $N-\lambda'$ processors receive bags consistent in v. Among them, at least $N-\lambda'-\pi$ processors are non-faulty and echo v in the second round. Since by the statement of the theorem $N-\lambda'-\pi$ is greater than z and $\lambda'+\pi$ is less than z, non-triviality is ensured.

If the transmitter is faulty, the bag of messages observed over the channels resulting from its initial broadcast can be either consistent in v or inconsistent. If it is consistent, we have decision on v as above. Otherwise, by Lemma 4 at least $N-\lambda'-\pi$ non-faulty processors receive inconsistent bags. In the second round these processors echo δ. By the same argument as in the previous paragraph, decision on δ is ensured. □

Note that the modified protocol P2 with threshold $z=L+1$ implements RB for Omission Fault processors and arbitrary communication failures, provided $N>L+\pi+\lambda'$.

10. Discussion and Conclusions

In this paper, we have shown that the reliable broadcast operation in distributed systems need not be expensive (in time) if the correct model of communication is realized through the network architecture. Furthermore, additional degrees of fault tolerance can be "bought" simply by increasing the number of appropriate system components but without slowing the system down.

From a practical point of view, we must be careful in comparing two different protocols based on their time complexity as measured in rounds. The total communication time associated with a single round of protocols P1 and P2 can be structured to require exactly N single message transmission (end-to-end delay) times. If the system is composed of a very large number of processors, this communication step could account for an inordinate amount of time. However, in such cases, only the minimum number of processors required by the protocol for a given degree of fault tolerance need actively participate in the protocol execution. For example, in protocol P1 only $\lambda+\pi$ processors are required to implement RB for a given value of $\lambda+\pi$. If $N>>\lambda+\pi$, then we can simply arrange that only $\lambda+\pi$ of the total available processors actively participate in the protocol execution (i.e., by broadcasting messages) while the others participate as passive listeners. All of the processors execute the same protocol but the passive ones simply refrain from performing any of the broadcasts that are prescribed. In this manner, the decision they reach will be consistent with that of the the active processors. Obviously, the same observation applies to the other protocols.

The parallelism within the communication step of a round for a broadcast network architecture is limited by the number of physical media that exist within the channel implementation. If there is a single medium that has to be time multiplexed (as it usually is the case), the communication step duration grows proportionally to the number of processors in the system. On the other hand, a point-to-point communication architecture allows for parallelism of message transmissions over all of the available links. In particular, if the communication topology forms a fully connected graph, all processors can communicate with each other in a single end-to-end delay time since all message transmissions can proceed in parallel. However, we conjecture that for point-to-point communication networks with N processors and only $O(N)$ links (i.e., less than fully connected), the communication step duration of a round will require $O(N)$ end-to-end delay times just as in broadcast networks.

ACKNOWLEDGEMENTS

We are grateful to Fred Schneider and Gil Neiger for discussing this material with us and commenting on earlier versions of this manuscript. We would also like to thank Flaviu Cristian for his helpful comments.

REFERENCES

[1] ABRAMSON, N. The ALOHA system. In *Computer-Communication Networks*, N. Abramson and F. Kuo (eds.). Prentice Hall, Englewood Cliffs, N.J., 1973.

[2] CHANG, J.M. AND MAXEMCHUK, N.F. Reliable broadcast protocols. *ACM Trans. Compt. Sys. 2*, 3, (August 1984), 251-273.

[3] DIFFIE, W. AND HELLMAN, M. New directions in Cryptography. *IEEE Trans. on Information Theory IT-22* (1976), 644-654.

[4] DOLEV, D. AND STRONG, H.R. Authenticated algorithms for Byzantine agreement. *SIAM J. on Comput. 12*,4 (November 1983), 656-666.

[5] FARBER, D., FELDMAN, J., HEINRICH, F., HOPWOOD, M., LARSON, K., LOOMIS, D. AND ROWE, L. The distributed computing system. In *Proceedings of CompCon 73*, IEEE Computer Society (February 1973), 31-34.

[6] FISHER, M.J. The consensus problem in unreliable distributed systems (a brief survey). Tech. Rep. YALEU/DCS/RR-273, Computer Science Department, Yale University, June 1983.

[7] FISHER, M.J. AND LYNCH, N.A. A lower bound for the time to assure interactive consistency. *Inform. Proc. Letters 14*,4 (April 1982) 183-186.

[8] FISHER, M.J., LYNCH, N.A. AND PATERSON, M.S. Impossibility of distributed consensus with one faulty process. In *Proceedings of 2nd ACM Symposium on Principles of Database Systems* (March 1983).

[9] GARCIA-MOLINA, H., PITELLI, F., AND DAVIDSON, S. Applications of Byzantine Agreement in database systems. Technical Report TR-316, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey, June 1984.

[10] GEHANI, N.H. Broadcasting satellite processes (BSP). *IEEE Trans. on Software Engineering SE-10*, 4 (July 1984), 343-351.

[11] 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.

[12] HALPERN, J., SIMONS, B., STRONG, R. AND DOLEV, D. Fault-tolerant clock synchronization. In *Proceedings of 3rd ACM Symposium on Principles of Distributed Computing* (Vancouver, Canada, August 1984), 89-102.

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

[14] LAMPORT, L., SHOSTAK, R. AND PEASE, M. The Byzantine Generals Problem. *ACM Trans. Prog. Lang. Syst. 4*, 3 (July 1982), 382-401.

[15] LELANN, G., A distributed system for transaction processing. *IEEE Computer 14*, 2 (February 1981), 43-48.

[16] METCALFE, R. AND BOGGS, D.R. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM 19*, 7 (July 1976), 396-403.

[17] MOHAN, C., STRONG, H.R. AND FINKELSTEIN, S. Method for distributed transaction commit and recovery using Byzantine agreement within clusters of processors. In *Proceedings of 2nd ACM Symposium on Principles of Distributed Computing* (Montreal, Quebec, Canada, August 1983), 89-103.

[18] PEASE, M., SHOSTAK, R. AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (April 1980), 228-234.

[19] RIVEST, R.L., SHAMIR, A. AND ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 2 (February 1978), 120-126.

[20] SCHNEIDER, F.B. Synchronization in distributed programs. *Trans. Prog. Lang. Syst. 4*, 2 (April 1982), 125-148.

[21] SRIKANTH, T.K. AND TOUEG, S. Clock Synchronization. Tech. Rep. TR 84-656, Computer Science Department, Cornell University, Ithaca, N.Y. (December 1984).

[22] TANENBAUM, A. *Computer Networks.* Prentice Hall, Englewood Cliffs, N.J., 1981.