Reliable Broadcast Protocols

JO-MEI CHANG and N. F. MAXEMCHUK
AT&T Bell Laboratories

A reliable broadcast protocol for an unreliable broadcast network is described. The protocol operates between the application programs and the broadcast network. It isolates the application programs from the unreliable characteristics of the communication network. The protocol guarantees that all of the broadcast messages are received at all of the operational receivers in a broadcast group. In addition, the sequence of messages is the same at each of the receivers and a total ordering exists among all broadcast messages. This unique message sequencing can be used to simplify distributed database systems and distributed processing algorithms.

The protocol can operate with as few as one acknowledgment message per broadcast message, instead of one acknowledgment from each receiver per broadcast message. The protocol continues to operate when sites in the broadcast group fail.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*distributed networks*; *network communications*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*protocol architecture*; C.2.5 [**Computer-Communication Networks**]: Local Networks—*access schemes*; C.4 [**Computer Systems Organization**]: Performance of Systems—*performance attributes*

General Terms: Algorithms, Design, Performance, Reliability

Additional Key Words and Phrases: Broadcasting, multicasting, message sequencing, failure recovery

1. INTRODUCTION

Broadcast networks are common both in local area communication systems [4, 6, 7, 9] and in long haul satellite systems [8]. Messages transmitted in these networks are available to all receivers; however, some or all of the receivers may lose a message because of transmission errors or because a buffer overflows within a receiver. On point-to-point communication links, protocols are implemented to recover lost messages. Without a similar protocol, designed for a broadcast link, the capabilities of this type of network cannot be fully utilized by applications broadcast or multicast.

In this paper, a protocol is presented that allows groups of sites on unreliable broadcast networks to reliably broadcast messages [1, 2]. The protocol guarantees that all of the receivers in a group receive the broadcast messages and that each of the receivers order the messages in the same sequence. It will be shown that for each broadcast message much less than one acknowledgment per receiver is required.

Authors' address: AT&T Bell Laboratories, Murray Hill, NJ 07974.

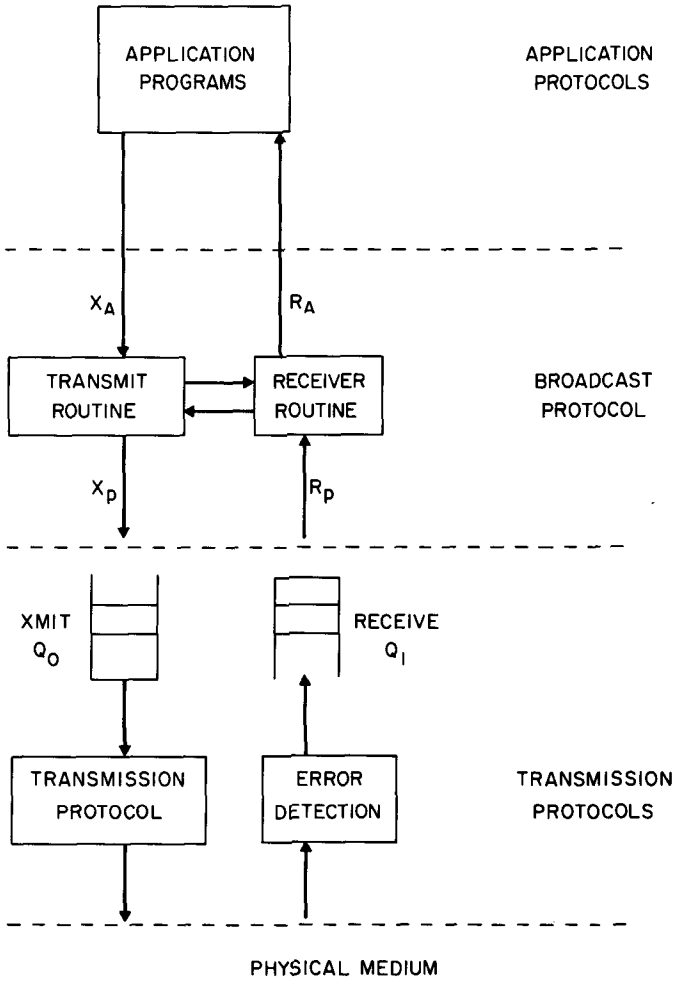


Fig. 1. Protocol description.

Requiring all of the receivers to order the messages in the same sequence is stricter than just requiring them to obtain all of the messages. However, this property is useful in distributed processing systems. If processes residing at different sites receive messages in the same sequence, they can arrive at the same conclusion without additional communications. Unique message sequencing can be used to simplify the design of concurrency control and crash recovery procedures in distributed database systems [3].

2. SYSTEM DESCRIPTION AND FAILURE ASSUMPTIONS

2.1 System Description

The broadcast protocol operates between the application level protocols and the transmission protocols, as illustrated in Fig. 1. At each site, application level programs transmit messages across the interface X_A and receive messages across

the interface R_A. For any transmitter, all of the messages transferred across interface X_A are delivered to the receivers in the same sequence. This sequence is preserved by transmitting one message at a time until the message is acknowledged. It is uncertain how these messages will be sequenced with respect to messages from other transmitters; however, the broadcast protocol guarantees that the sequence will be the same at every interface R_A.

The transmission protocol provides a datagram service [8] to the broadcast protocol. All messages from the broadcast protocol are transmitted as isolated units. The transmission protocol is responsible for acquiring the communication channel, transmitting messages, and accepting messages destined for the broadcast group. The receiver in the transmission protocol discards messages with transmission errors. Messages may also be lost because Q_i overflows. It is the responsibility of the broadcast protocol to retransmit lost messages and to sequence the received messages. When a broadcast message is first received in Q_i, it is not *committed* and cannot be read by the application programs. It remains uncommitted until the broadcast protocol can guarantee that this message will be received by all of the operational receivers. The broadcast protocol will then commit the message by passing it across the interface R_A to the application programs.

Messages transmitted on the broadcast network have a destination address. A broadcast group consists of N sites; each of the N sites can transmit and receive broadcast messages addressed to this group. Other addresses on the network can be used for point-to-point communications or for other broadcast groups. A receiver that is capable of recognizing several addresses can belong to several broadcast groups as well as conducting point-to-point communications.

2.2 Failure Assumptions

The following failure assumptions are made:

(1) We assume that when a site fails, the site simply stops processing. It does not send malicious messages or perform incorrect actions.

(2) Messages may be lost because of buffer overflow or they may be discarded because of transmission error. An arbitrary number of messages may be lost; however, all of the messages processed at a site are free of transmission errors.

(3) A failure may be due to a communication failure or a site failure. A *failure* occurs when a site in the broadcast group fails to communicate with another site after R attempts. A site that fails to respond is assumed to be *nonoperational.* A site recovers and becomes operational when it reestablishes communication with sites in the broadcast group. It is possible that a site is mistakenly assumed to have failed. The parameter R should be large enough that this will happen infrequently and small enough that failures can be detected in a reasonable amount of time.

3. PHILOSOPHY

Consider a system with multiple transmitters and receivers in a broadcast group. The most straightforward way to implement a reliable protocol is to retransmit the broadcast message until an acknowledgment is received from each of the

R_i: **Receive** M_1, **Receive** M_2.

R_j: (miss M_1), **Receive** M_2, **Receive** M_1.

Fig. 2. Positive acknowledgment system.

receivers. *This is a positive acknowledgment system.* If N sites must receive the message, at least N acknowledgments must be transmitted. This protocol does not guarantee that each of the receivers obtain the broadcast messages in the same sequence. For instance, consider the following example with two receivers. Source S_A broadcasts message M_1, which is received by R_i but missed by R_j. Meanwhile, another source S_B broadcasts message M_2 which is received by both R_i and R_j. Finally, S_A retransmits M_1 which is received by R_j. R_i receives M_1 before M_2 and R_j receives M_2 before M_1, as illustrated by Fig. 2.

In a system with a single receiver or a single transmitter, reliable protocols can be implemented which sequence the received messages and may require fewer acknowledgments than the above protocol. In the system described above, if there is a single receiver and many sources, there is only one sequence of received messages, and one acknowledgment per broadcast message. When there is a single source and many receivers, sequencing messages at all of the receivers is trivial. The source assigns a sequence number to each message. It is not necessary for receivers to explicitly acknowledge messages. Messages that are lost are detected when a higher sequence number than expected is received, and retransmissions of the lost messages can be requested. Eventually, the proper sequence of messages is obtained by all of the receivers. *This is a negative acknowledgment system.*

In general, a broadcast protocol must operate between many sources and many receivers. The philosophy of the proposed protocol is to make the general system appear to be a combination of two simple systems, one with a single receiver and the other with a single transmitter. A system with many transmitters can be made to look like a system with a single transmitter by passing all of the messages through a primary receiver, which will be called the *token site.* This receiver appears to be a funnel through which all messages must pass, as shown in Fig. 3. The system operates as a positive acknowledgment system between the sources and the token site. The token site transmits one acknowledgment for each message from the sources. The acknowledgment contains a sequence number, called a timestamp. The system operates as a negative acknowledgment system between the token site and the remaining receivers. The remaining receivers use the timestamp to detect missing messages; they then request the missing acknowledgments and the acknowledged messages. The receivers place the messages in the sequence in which the token site acknowledged them. They do not transmit any messages if no messages are lost. Therefore, this protocol uses one acknowledgment per broadcast message.

There are two deficiencies with this protocol:

(1) In a negative acknowledgment system, there is no way to know when the receivers obtain the messages. Broadcast messages must thus be retained indefinitely for possible retransmission.

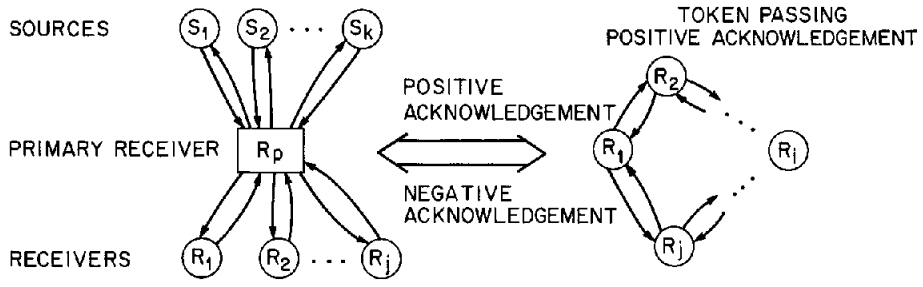


Fig. 3. Reliable broadcast protocol.

(2) When the token site fails, messages that it has acknowledged, but that have been missed by all of the other receivers, may be lost.

These deficiencies are eliminated by rotating the token site responsibility among all of the operational sites, requiring a receiver to have all of the timestamped messages before accepting the token, and requiring at least L additional receivers to accept the token before committing a message. The list of operational sites is called the token list. The token is transferred as part of an acknowledgment message. When a site accepts the token, all of the other sites in the token list have accepted the token since last time this site did. Therefore, all of the messages acknowledged the last time this site had the token have been received by all of the operational sites and will no longer have to be retransmitted. Only a finite number of broadcast messages must be retained for possible retransmission. Furthermore, if L sites accept the token after a message is acknowledged, at least $L + 1$ sites have the message and at least $L + 1$ sites would have to fail for the message to be lost.

The protocol provides mechanisms to detect site failures. The token site must acknowledge broadcast messages and is responsible for servicing retransmission requests from the other receivers. Token site failures are detected when a source cannot obtain an acknowledgment for a broadcast message or when one of the receivers cannot recover lost messages. A message which transfers the token is retransmitted until an indication is received that the next site has either accepted the token or failed. If there are N sites in the token list, any failure is detected within N token transfers.

The system alternates between two phases, a normal phase and a reformation phase. During the normal phase, messages are acknowledged and the token is passed. When a failure is detected or a site recovers, the system enters a reformation phase. During the reformation phase, a new token list is generated. The normal phase is described in Section 4, and the reformation phase is described in Section 5.

4. NORMAL PHASE

During normal operation, sources broadcast messages. Some receivers may lose messages because of transmission errors or buffer overflows. However, the protocol guarantees that each operational receiver will acquire the broadcast

messages that have been lost and that all of the receivers will place the messages in the same sequence.

If sites in the broadcast group are using different token lists, a reformation will occur. In addition, a reformation occurs when certain messages do not receive a response. The following messages solicit responses:

(1) When a broadcast message is transmitted, an acknowledgment is expected from the token site.
(2) When a retransmission is requested, the retransmitted message is expected from the token site.
(3) When a token is transferred, a confirmation from the next token site is expected.

If the site responsible for responding to one of these messages does not transmit any messages during R successive retransmission attempts, it is assumed that that site has failed.

4.1 Operation of the Protocol

Each site i maintains the following information:

—tl_i, the version number of the token list it is using;
—$M_i[s]$, the number of the next broadcast message it expects from site s;
—nts_i, the next timestamp it expects to receive.

During the normal phase, a site is committed to one token list tl_i. All of the messages transmitted identify this token list, and only sites using the same token list communicate. $M_i[s]$ is used to differentiate between new messages from source s and messages which have been acknowledged. It prevents a broadcast message from being acknowledged more than once. The purpose of nts_i is to ensure that site i obtains all of the acknowledgments and acknowledged messages in the proper sequence. When the normal phase starts, all of the sites in the token list tl_i have the same values for nts_i and $M_i[s]$. One site is given the token and can transmit an acknowledgment with this timestamp.

There are three phases in broadcasting a message: *transmitting*, *timestamping*, and *committing*.

Transmitting. A source retransmits a broadcast message until it receives an acknowledgment for the message. This is a positive acknowledgment system. If the token site does not transmit any messages during R successive retransmission attempts, the source assumes that the token site has failed.

Each broadcast message contains an identifier $B(s, n)$ which identifies the source s and the message number $n = M_s[s]$ from the source. The next broadcast message from a source is not transmitted, and the message number at the source is not incremented, until an acknowledgment is received. Therefore, successive messages from source s have incrementally increasing sequence numbers. This allows broadcast messages to be uniquely identified.

Timestamping. The token site acknowledges broadcast messages. When the token site, at i, receives a broadcast message $B(s, n)$ for which $M_i[s] = n$, it assumes that this message has not been acknowledged. The token site transmits

an acknowledgment, $ACK(nts_i, B(s, n))$. Each acknowledgment message indicates which site will transmit the next acknowledgment. Because of the resiliency requirements, described below, a site occasionally transfer the token when there are no broadcast messages to be acknowledged. It does this by transmitting $ACK(nts_i, NULL)$, which is an acknowledgment to a NULL message.

The receivers store broadcast messages in a queue Q_B and process the acknowledgments in the order they are received. At site i, $ACK(ts, B(s, n))$ is only processed when $nts_i = ts$ and $B(s, n)$ is the NULL message *or* $B(s, n)$ is in Q_B. When acknowledgment ts is procesed, nts_i is incremented and if a message has been acknowledged, the next message from the source is changed to $M_i[s] = n + 1$. If $ts < nts_i$, this acknowledgment has been previously processed and is not processed again. If $ts > nts_i$, acknowledgment messages have been lost. Before processing acknowledgment ts, the missing acknowledgments are requested and processed. If $B(s, n)$ the broadcast message being acknowledged, is not in Q_B, it must be obtained before the acknowledgment is processed. Retransmission requests are retransmitted until the requested message is received, or until a failure occurs. Any acknowledgments that are received while waiting for a retransmitted message are stored in a queue Q_c and are processed in the order they arrive.

Committing. After the message is timestamped and the token is transferred L times, it is certain that $L + 1$ sites have obtained the broadcast message. At this time the message is committed and transferred to the application program. As long as L or fewer sites in the token list fail, all committed messages can be recovered during the reformation phase. This is referred to as an *L-resilient* system.

In order to guarantee that an acknowledged message can be committed, the token must be transferred L times after the message is acknowledged. A token site with an uncommitted message waits a period T for a new broadcast message. If there are no new messages, it transmits $ACK(nts_i, NULL)$. This procedure guarantees that the system cannot become deadlocked because a critical message cannot be committed. There is a tradeoff between the commit delay and the number of messages transmitted by the protocol. Resiliency is obtained by introducing commit delay or additional messages. This tradeoff is examined in Section 6. This procedure does not guarantee that every receiver has committed the message. By continuing to pass the token after the last message is committed, the probability that the message is committed everywhere increases. When the token is transferred around the entire token list back to the site which acknowledged it, it is certain that every site has received the message. If the token is transferred further, until it reaches the site which caused the message to be committed, it is certain that every site has committed the message.

4.2 Token Site

In addition to acknowledging messages, the token site is responsible for transferring the token to the next token site and for responding to retransmission requests.

The token transfer is a positive acknowledgment system. A site continues to transmit an acknowledgment which transfers the token until it receives a message

from the next token site which indicates that it has accepted the token. If the next site receives a broadcast message and transmits an acknowledgment, or transfers the token to commit a message, this constitutes accepting the token. If there are no new messages or uncommitted messages, the next site transmits a confirmation message and retains the token until a new broadcast message is received. This strategy guarantees that the token will be transferred between two sites that can communicate. Since the token is transferred as part of the acknowledgment message and since the next site can accept the token by transmitting an acknowledgment, the token transfer operation does not generate any additional messages when there are broadcast messages to be acknowledged.

The next token site accepts the token when it can process the acknowledgment that transfers the token to it. Therefore, when a site accepts the token, it has received all of the acknowledgments and acknowledged broadcast messages that may be requested. This site responds to all retransmission requests. It continues to answer retransmission requests until it is certain that the next site has accepted the token. If it did not, the next site would not be able to recover lost messages. If the token site misses the message which indicates that the next site has accepted the token, retransmission requests may be answered by more than one site. All sites which have accepted the token have the same set of messages, and the requesting site can simply discard the redundant response. This strategy guarantees that at least one site which is responsible for answering retransmission requests has all of the messages up to the last transmitted timestamp. Since the retransmission requests are retransmitted until a response is obtained, a missed message will be recovered as long as a communication failure does not occur.

Messages that require a response are retransmitted if the response is missed. The token site treats these messages as if they were retransmission requests for the response. If a previously acknowledged broadcast message is received, the site servicing retransmission requests assumes that the source missed the acknowledgment, and the acknowledgment is retransmitted. If an old acknowledgment is received, this site assumes that a previous token site missed the message that assumed token site responsibility, and this message is retransmitted.

The interaction between the sites is summarized in Fig. 4. The processing strategy at the receivers guarantees that all of the sites in the broadcast group place broadcast messages in the same sequence. The positive acknowledgment strategy guarantees that a broadcast message will be acknowledged as long as all sites in the broadcast group can communicate. The token passing strategy forces all sites to recover missing messages. The retransmission strategy guarantees that sites can recover missing messages. Therefore, during the normal phase of the protocol all of the operational sites in the broadcast group will receive, and thus commit, broadcast messages in the same sequence.

5. REFORMATION PHASE

The protocol enters the *reformation phase* when a failure or recovery is detected. Initially, the token list consists of all of the sites in the broadcast group. When a failure or recovery is detected, the reformation process redefines the token list and elects a new token site. The process is complete when a new token list is

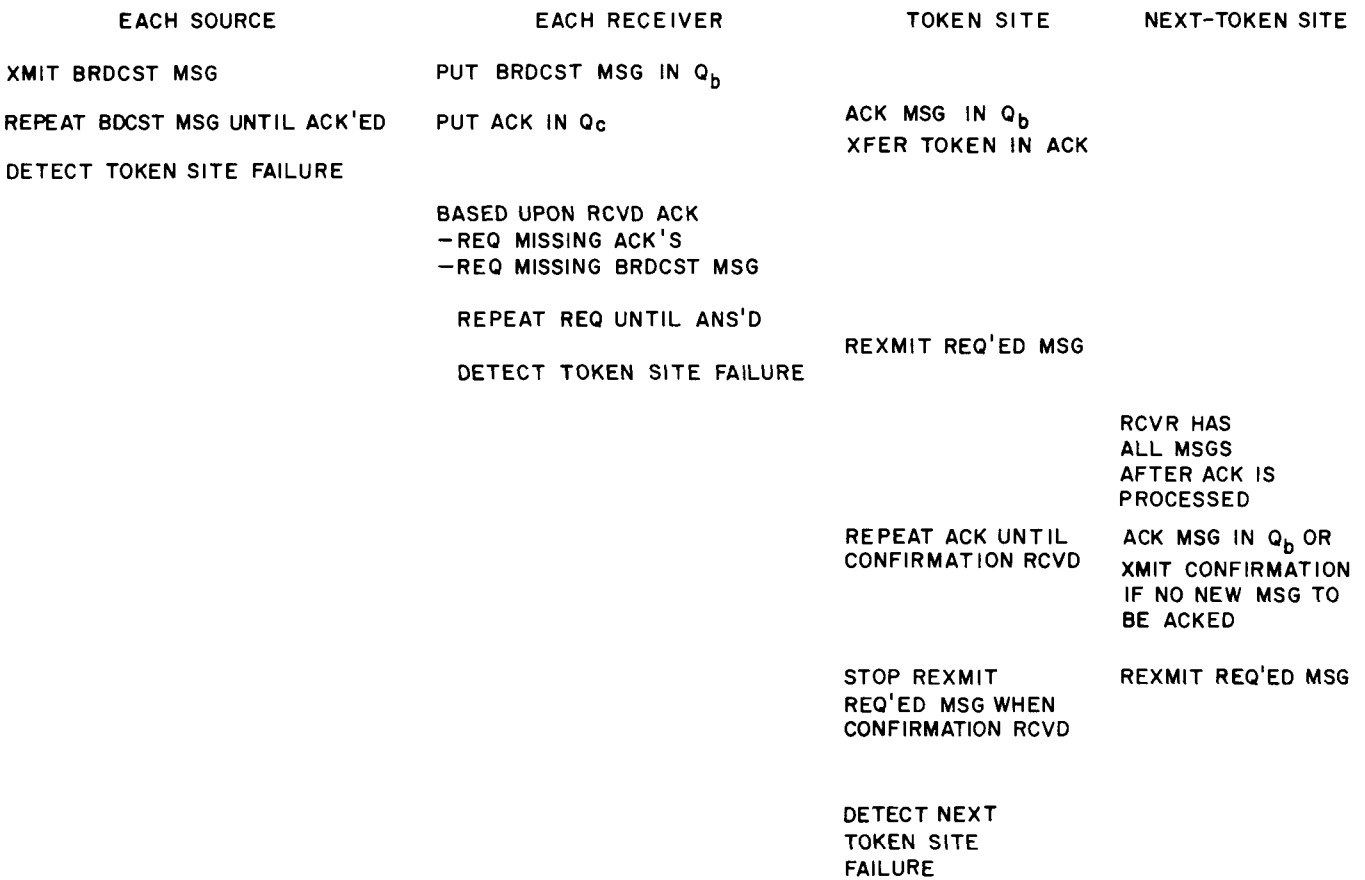


Fig. 4. Interaction among sources, receivers, token site, and next token site.

formed and a new token is generated and accepted by the new token site. The protocol then resumes normal operation.

During the reformation, failures can occur. The reformation protocol must be robust against an arbitrary number of lost messages and must not be blocked if sites fail during the reformation process. Furthermore, a site may fail to communicate with the token site or the next token site because of repeated communication failures rather than because the site has failed. The reformation process ensures that

(1) only one valid token list can exist at any time;
(2) none of the committed messages from the old token list are lost.

5.1 A Valid List

Any site that detects a failure or recovery initiates a reformation and is called an *originator.* It invites other sites in the broadcast group, the *slaves*, to form a new list. The reformation process can be described in terms of the activities of sites *joining* and *committing* a *valid list.* A valid list satisfies a set of specific requirements, as explained below. When the reformation starts, a site is invited to join a new list and eventually commits to a valid list. When all of the sites in a valid list are committed to this list, the list will be authorized with a token and the reformation terminates. This list becomes the *new token list.*

Multiple originators can exist if more than one site discovers the failure or recovery. During the reformation, it is possible that acknowledged messages from the old token list have been missed by all sites that join a new list. To guarantee that there is only one new list and that this list has all of the committed messages, the list must be tested before it can be considered a valid list. Specifically, a list becomes valid if it passes the *majority test*, the *sequence test*, and the *resiliency test.*

Majority Test. The majority test requires that a valid list has a majority of the sites in the broadcast group. During the reformation, a site can join only one list. The majority test is necessary to ensure that only one valid list can be formed.

Sequence Test. The sequence test requires that a site only join a list with a higher version number than the list it previously belonged to. The version number of a token list is in the form of (version #, site number). Each site has a unique site number. When a new list is formed, the originator chooses the new version # to be the version # of the last list it has joined plus one. Therefore, token lists have unique version numbers. The originator always passes the sequence test. If any of the slaves fail the sequence test, it tells the originator its version number. The originator increments the higher version # the next time it tries to form a new list.

The combination of the majority and the sequence test ensures that all valid lists have increasing version numbers. This is true because any two valid lists must have at least one site in common, and a site can join a second list only if the second list has a higher version number. Therefore, the version numbers indicate the sequence in which token lists were formed.

Resiliency Test. The resiliency test ensures that none of the messages committed by the old token list are lost. It also ensures that the old token list can no longer be effectively used by any site that is not aware of the reformation. In an L-resilient system, a committed message must be received by the L sites following the site that acknowledges the message; therefore, at least $L + 1$ sites have received it. If the new list consists of one of the $L + 1$ sites following the site that assigned the largest known timestamp in the old token list, none of the committed messages can be lost. This also ensures that the old token list does not have all of the sites needed to commit additional messages.

To perform the resiliency test, the old token list and the last timestamp issued by this list must be determined. When a site i agrees to join a list, it tells the originator the timestamp of the next message to be processed, nts_i, and its token list version number. The list with the largest known version number is considered to be the *old list.* This is the list that the resiliency test of the new list is based on. The resiliency test is successful if the new list consists of either the site that is expected to issue the largest known next timestamp of the old list or one of the L sites following it in the old list. The resiliency test is explained further in Section 5.2.

When a valid list is formed, a new token site has to be elected and its starting timestamp has to be determined. The site that nominates the largest known next timestamp has the most complete information regarding the old list. This site is elected as the new token site. The starting timestamp of the new list equals the largest known next timestamp of the old list. The new token site has all the messages up to this timestamp. Before resuming normal operation, each site in the new list recovers any missing message from the new token site. This guarantees that the system is still L-resilient when the new list is used. Note that once a site joins a new list, even if the new list cannot be successfully formed, the site will not use the old list to process messages. Therefore, once a new list passes the resiliency test, the list that it bases its test on is, and remains, inactive.

When a site recovers from failure, it discards all of its uncommitted messages and requests retransmissions from the new token site. Note that this site may have received messages before its failure which were missed by the new token site. These messages are discarded. Therefore, the recovered site has the same message sequence as the new token site. A recovered site must also obtain (from the new token site) $M_{NTS}[s]$, the number of the next broadcast message from each site in the broadcast group. Consequently, each site in the new list, including a recovered site, can differentiate between a new broadcast message and a previously acknowledged message.

5.2 A Reformation Protocol

The reformation protocol is described in Fig. 5. It is a three-phase protocol for the originator and the new token site and a two-phase protocol for the slaves. In Phase I, the originator forms a new list. In Phase II, if the new list passes the majority and resiliency tests, it is announced to the slaves, and a new token site is elected. In Phase III, the new token site is authorized: a new token is generated by the originator and passed to the new token site. After accepting the token,

```
Phase I:    When a failure of recovery is detected, start Reformation;
            Broadcast an invitation to all sites in the broadcast group.
Phase II:   Wait for either all responses received or TIMEOUT_i*;
            If (all responses = "yes" and pass Majority and Resiliency Tests)
              New TL = {all sites responded};
              Announce New TL to all sites in the New TL;
            Else
              Announce "Reformation Abort" to all sites in the New TL*;
              Modify TL version number, Wait and Restart;
Phase III:  Wait for either all responses received or TIMEOUT_i;
            If (All responses from sites in New TL = "Yes")
              Generate "a New Token" and pass to New Token Site;
              Commit New TL;
              End of Reformation Phase;
            Else
              Announce "Reformation Abort" to New Token site;
              Wait and Restart;
```

(a)

```
Phase I:    Wait for "Reformation Invitation" is received;
            If (Sequence Test passes and j does not belong to any reformation list)
              Vote "Yes";
            Else  Vote "No";
Phase II:   Wait for either "New TL", "Abort" is received, or TIMEOUT_j;
            If ("New TL" is received)
              If (j still belongs to this list)
                Recover all missing and then Vote "Yes"; messages
                Commit the New TL;
                End of Reformation Phase (except the new token site);
              Else  Vote "No";
            If ("Abort" is received or TIMEOUT_j)
              Leave the list that previously joined.
              Wait and Restart;
```

(b)

```
Phase III:  Wait for either "New Token," "Abort," or TIMEOUT_k;
            If ("a New Token" is received and j still belongs to this list)
              Accepts the token and Starts Acknowledging Messages;
              End of Reformation Phase;
            If ("Abort" is received or TIMEOUT_k)
              Wait and Restart;
```

(c)

Fig. 5. (a) Reformation protocol for originator site i. (b) Reformation protocol for slave j including new token site. (c) Authorization phase for new token site k. (The asterisk indicates that each slave has been given R opportunities to respond.)

the new token site resumes normal operation and acknowledges broadcast messages.

Specifically, in Phase I, an originator sends out an invitation to all of the sites in the broadcast group to form a new list. The invitation carries the new token list version number. A slave site can only join one list and it can only join a list that passes the sequence test.

The originator enters Phase II when any negative response is received or when every site in the broadcast group has either responded or has failed to respond after R attempts. If all responses are positive, the new list consists of all of the sites that have responded. The majority and resiliency tests are applied to the new list. A new token site and a starting timestamp are also determined. A valid list is announced to all of the sites in the new list. If the list is not valid or if a negative response is received, the originator aborts the reformation process and releases its members.

To prevent the reformation process from being blocked when the originator fails, a site leaves a list if no messages are received from the originator during a specific *timeout* period. Therefore, a site may join one list, release itself after a timeout, and join another list. Because a site can leave a list, in Phase II a slave is required to acknowledge receiving the valid list. If a slave has already left this list, it responds negatively. A slave can respond positively only if it still belongs to this list and has recovered all missing messages. A site missing any message will first request retransmissions from the new token site. After recovering all missing messages, the slave commits to the new list and resumes normal operation.

The new token site has all of the messages up to the starting timestamp of the new list. When the new list is announced, if the new token site still belongs to the new list, it answers retransmission requests from the other slaves. If the new token site has already left this list, it votes negatively and ignores retransmission requests.

In Phase III, if a unanimous vote is obtained, the originator authorizes the creation of a new token at the new token site. When the new token site receives this token, and if the new token site has not yet left the list, it accepts the token and starts acknowledging broadcast messages. The reformation phase is now complete. If the originator cannot obtain a unanimous vote from the slaves, it notifies the new token site to abort the reformation process.

The reformation process may be aborted because there are multiple originators or because a failure occurs. Any time a site leaves a list due to an abort message or a timeout, it waits for a random period of time and then restarts the reformation process. The random wait period reduces the possibility that two sites initiate the reformation process simultaneously. This process is repeated until a new token site successfully accepts the token. Eventually, if enough sites remain operational in the system, a valid list will be formed and a new token site will be authorized.

The existence of aborted lists creates the following problems:

(1) A site could be committed to an aborted list while other sites have left the list due to a timeout. Therefore, operational sites could temporarily be committed to different lists.

(2) If a site in a new list was last committed to an aborted list, this list, having a higher version number than the old token list, will be used for the resiliency test of the new list.

These two problems are handled by the reformation protocol.

First, sites could be committed to different lists. However, if a new token site is not authorized, this token list is never used to acknowledge broadcast messages.

Sites committed to this list will eventually assume that the new token site has failed and start a reformation. Meanwhile, sites that have not committed to this list, because of a timeout or an abort message, will also restart the reformation process. In either case, another reformation will start. Because a unanimous vote is required, when a token site is successfully authorized all sites in the list are committed to the list.

Second, the old list determined by the resiliency test is not the old token list. The function of the resiliency test is to ensure that the old token list is inactive and that the new token site elected has all of the committed messages from the old token list.

Old Token List Inactive. The combinations of the tests guarantee that if the new list is valid, the tested old list has a version number greater than or equal to the old token list. As discussed in Section 5.1, once a new list has passed the resiliency test, the old list that it is based on remains inactive. Therefore, a valid list formed after the old token list ensures that the old token list is inactive. Although the resiliency test is based on an aborted list, because the aborted list must have a version number greater than the old token list, the old token list must be inactive.

New Token Site Has All Committed Messages. Since a site must recover all messages from the new token site before it is committed to a list, and the new token site of an aborted list must have all of the committed messages from the old token list, any site that has committed to the aborted list has all of the committed messages from the old token list. When one of these sites is elected as the new token site, it is capable of answering all retransmission requests and none of the committed messages from the old token list can be lost.

6. PROTOCOL PERFORMANCE

In the protocol defined, the token can be passed after several broadcast messages are timestamped, each time a broadcast message is timestamped, or several times for each broadcast message timestamped. The broadcast message can be committed to the list of sequenced messages as soon as it is acknowledged or after the token has been passed to L additional sites. The token passing frequency and the commit delay define a family of broadcast protocols with different storage requirements, resiliency to failures, message delays, and numbers of control messages.

In Section 6.1, the number of control messages transmitted per broadcast message and the storage required in an error-free environment is analyzed. The total number of messages transmitted per broadcast message increases when messages are lost. In Section 6.2, the performance of the protocol under different error rates is analyzed and compared with simulation results.

6.1 Message Analysis

The control messages used in the protocol consist of acknowledgments, token passing messages, and confirmation messages. The protocol transmits one acknowledgment per broadcast message. The number of token passing messages depends upon the token transfer rate. A confirmation message is transmitted to

acknowledge receiving the token when no token transfer or acknowledgment messages are to be sent.

Initially, the number of control messages is calculated without considering the effect of resiliency. When the token is transferred once for each message acknowledged, the number of confirmation messages required per broadcast message is $P_{\bar{Q}}$, where $P_{\bar{Q}}$ is the probability that the queue of messages waiting to be acknowledged is empty. Therefore, the average number of control messages transmitted per broadcast message is $1 + P_{\bar{Q}}$. At most, two control messages per broadcast message are required. When a token is accepted, a site must have all of the previously acknowledged messages. Therefore, when a message is acknowledged and the token has passed around all sites in the token list, all sites in the token list must have received the message. If there are N sites in the token list, the token site only needs to retain the last $N - 1$ broadcast messages and acknowledgments in order to answer retransmission requests. Since every site becomes the token site, this storage is required at every site.

The number of confirmation messages transmitted can be reduced by transferring the token after acknowledging K_w messages. This, however, increases the storage requirement by a factor of K_w. As K_w increases to ∞, the average number of confirmation messages per broadcast message decreases to zero. The average number of control messages decreases to 1, and the storage required increases to ∞. This becomes a system in which a single site is responsible for acknowledging all of the messages.

The storage requirement can be reduced by transferring the token K_r times for each message acknowledged, but this increases the number of control messages transmitted. When the token must be transferred K_r times before the next broadcast message can be acknowledged, the number of control messages per broadcast message is $P_{\bar{Q}} + K_r$. However, the number of messages that must be stored for possible retransmission is $(N - 1)/K_r$ rounded up to the next largest integer. Since the retransmission buffer cannot be reduced below 1, there is no reason to make K_r greater than $N - 1$. As K_r increases to $N - 1$, the maximum number of control messages per broadcast message increases to N, and the retransmission buffer decreases to 1 message. $K_r = N - 1$ corresponds to a system in which every receiver must acknowledge the broadcast message. It has the same number of control messages as a positive acknowledgment system, which requires individual acknowledgments; however, all receivers are guaranteed to have the same message sequence.

The token transfer rate describes a family of protocols. A trade-off exists between the number of control messages transmitted per broadcast message and the storage requirements as summarized in Table I. As the number of control messages per broadcast message increases from 1 to N, the storage for retransmitting messages decreases from ∞ to 1. The system in which a single receiver acknowledges all of the messages, and assigns timestamps to the messages, lies at one extreme of this family, and a system in which every receiver must acknowledge every message lies at the other extreme. In general, a single token transfer per broadcast message provides a satisfactory compromise between the number of control messages transmitted and the storage required. When $K_w = K_r = 1$, 1 to 2 control messages are transmitted depending on the system

Table I. Trade-off between Number of Control Messages and Storage

Rate of token transfer	Number of control messages	Storage
1 per acknowledgment	$1 + P_{\bar{q}}$, max 2	$N - 1$
1 per K_w acknowledgment	$1 + P_{\bar{q}} \rightarrow 1$, as $K_w \rightarrow \infty$	$K_w*(N-1) \rightarrow \infty$, as $K_w \rightarrow \infty$
K_r per acknowledgment	$P_{\bar{q}} + K_r \rightarrow N$, as $K_r \rightarrow N - 1$	$(N-1)/K_r \rightarrow 1$, as $K_r \rightarrow N - 1$

utilization. When the system is highly utilized, the number of control messages per broadcast message is 1, because no confirmation messages are transmitted.

The number of control messages transmitted is also affected by the resiliency of the system. A token passing message is transmitted after time T if no new broadcast messages arrive and the last acknowledged message has not been committed. This approach has the desirable characteristic that the number of messages transmitted is low during high utilization intervals and higher during low utilization intervals. During a high utilization interval, when there are almost always broadcast messages waiting to be acknowledged, no token passing message is transmitted. During low utilization periods, when there are almost never additional broadcast messages waiting to be acknowledged, there are at most $L - 1$ token passing messages per broadcast message.

The maximum time delay between acknowledging and committing a message is $T(L - 1)$. The shorter T becomes, the less likely a broadcast message will be waiting to be acknowledged and the greater the number of control messages. Therefore, in an L-resiliency system, there is a tradeoff between the commit delay and the number of control messages. This trade-off is examined by Maxemchuk and Chang [5].

In summary, depending upon the utilization, the protocol adaptively switches between 1 and $L + 1$ control messages per broadcast message. The resiliency requirement only increases the number of messages transmitted when the system is not busy. This shows the general characteristics of this broadcast protocol. There are fewer messages transmitted per broadcast message when the system is busy than when it is idle. This is verified by simulation results in Section 6.2.

6.2 Simulation Results

The protocol has been implemented and is running on an operational Ethernet connecting over 20 VAXs and SUNs and on a simulated broadcast network implemented on a single machine. The simulator uses the same code for the protocol as the real network. However, the error rate of the network can be controlled, the sequence of events leading to a failure can be reproduced, and execution can be stopped to investigate the state of all of the sites in a broadcast group. This has made it easier to debug the protocol and to make measurements under different error rates.

In the appendix, an approximate analytical model of the broadcast protocol and a positive acknowledgment protocol are developed. The positive acknowledgment protocol guarantees that every site in the broadcast group receives every message, but cannot guarantee that the broadcast messages can be placed in the same order at every site, as the example in Section 3 indicated. The average number of messages transmitted using both protocols is calculated. This number

is also determined using the simulator. The closeness of the analytical calculation and the simulation result strongly indicates that the protocol is performing as expected.

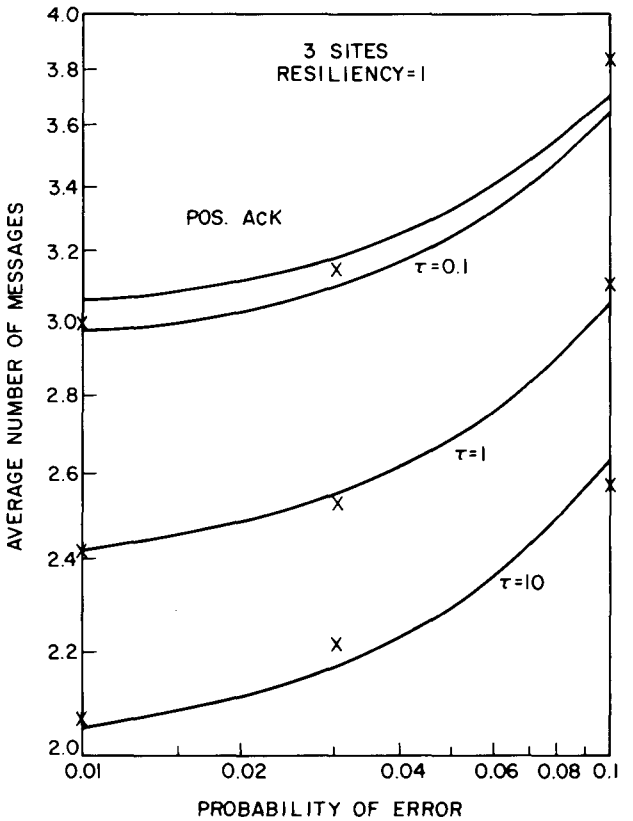
In Fig. 6, the average number of messages per broadcast message are plotted as a function of the probability of error for both the positive acknowledgment protocol and the reliable broadcast protocol. The lines correspond to the values predicted by the model, and the X's correspond to measurements taken on the simulator. Each simulation point was obtained by transmitting at least 1000 broadcast messages. In the simulation experiments reported here, over 50,000 messages were broadcast and received in the same order at every site in the broadcast group using the reliable broadcast protocol.

In Fig. 6a, b, and c, the broadcast group consists of 3, 10, and 30 sites, respectively, and the broadcast protocol has a resiliency of 1. The broadcast protocol is evaluated when the ratio of the token transfer period to the average message interarrival period τ is 0.1, 1, and 10. When τ is 10, the system is extremely busy, and there is almost always a broadcast message to be acknowledged. When τ is 0.1, the system is idle most of the time, and many confirmation messages must be transmitted. Fig. 6d shows the effect of resiliency on a system with 10 sites. Resiliencies of 1, 2, and 4 are considered. (Since the reformation cannot succeed unless a majority of sites can communicate, it does not make any sense to consider resiliencies greater than 4).

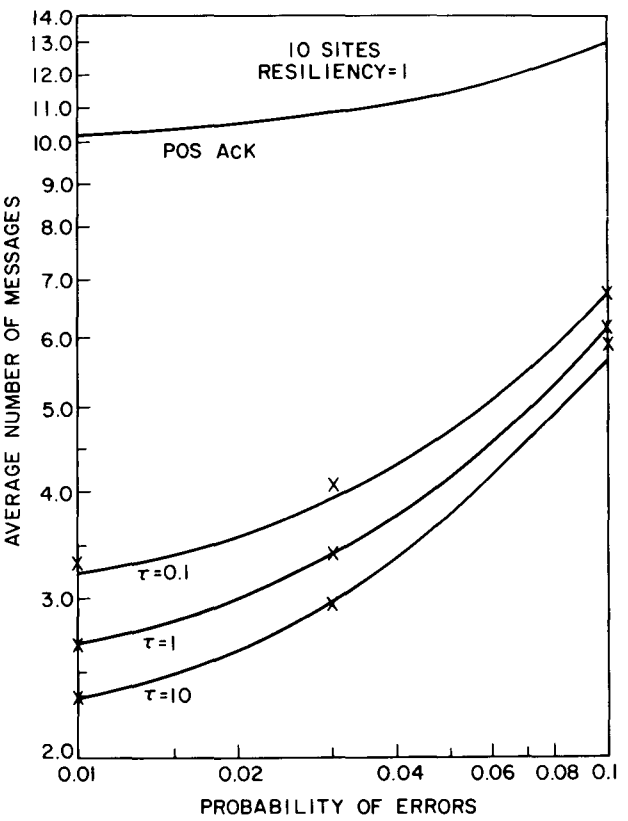
Fig. 6 shows that the reliable broadcast protocol, in general, transmits fewer messages than the positive acknowledgment protocol. Furthermore, in the reliable broadcast protocol, the number of messages transmitted per broadcast message decreases when the system is busy, increases when the error rate increases, and improves over the positive acknowledgment protocol as the number of sites in the broadcast group increases. Note that when there are only three sites in the broadcast group, only two sites must transmit acknowledgments in the positive acknowledgment protocol as opposed to one site transmitting an acknowledgment in the reliable broadcast protocol. Depending on the system load, the resiliency does not necessarily increase the average number of messages transmitted. Fig. 6(d) shows that when the system is busy, $\tau = 10$, the number of messages is independent of the resiliency. This is not surprising because there is almost always a broadcast message to be acknowledged, and very few token passing messages are generated. However, when the system is idle most of the time, $\tau = 0.1$, many token passing messages are required, and the average number of messages per broadcast message increases as the resiliency increases. Note that when error rate approaches 0, the average number of messages transmitted per broadcast message matches our prediction in Section 6.1. The total number of messages transmitted is approximately $L + 2$ when $\tau = 0.1$, and 2 when $\tau = 10$. (The total number of messages transmitted includes the broadcast message; hence the number of control messages estimated in Section 6.1 is one less.)

7. CONCLUSION

A family of reliable broadcast protocols have been designed. The tradeoffs between the number of control messages per broadcast message, the internal storage required, and the resiliency of the system have been studied. The protocols


3 SITES
RESILIENCY=1
POS. ACK
τ=0.1
τ=1
τ=10
AVERAGE NUMBER OF MESSAGES
PROBABILITY OF ERROR
4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
2.2
2.0
0.01
0.02
0.04
0.06
0.08
0.1

(a)


10 SITES
RESILIENCY=1
POS ACK
τ=0.1
τ=1
τ=10
AVERAGE NUMBER OF MESSAGES
PROBABILITY OF ERRORS
14.0
13.0
12.0
11.0
10.0
9.0
8.0
7.0
6.0
5.0
4.0
3.0
2.0
0.01
0.02
0.04
0.06
0.08
0.1

(b)

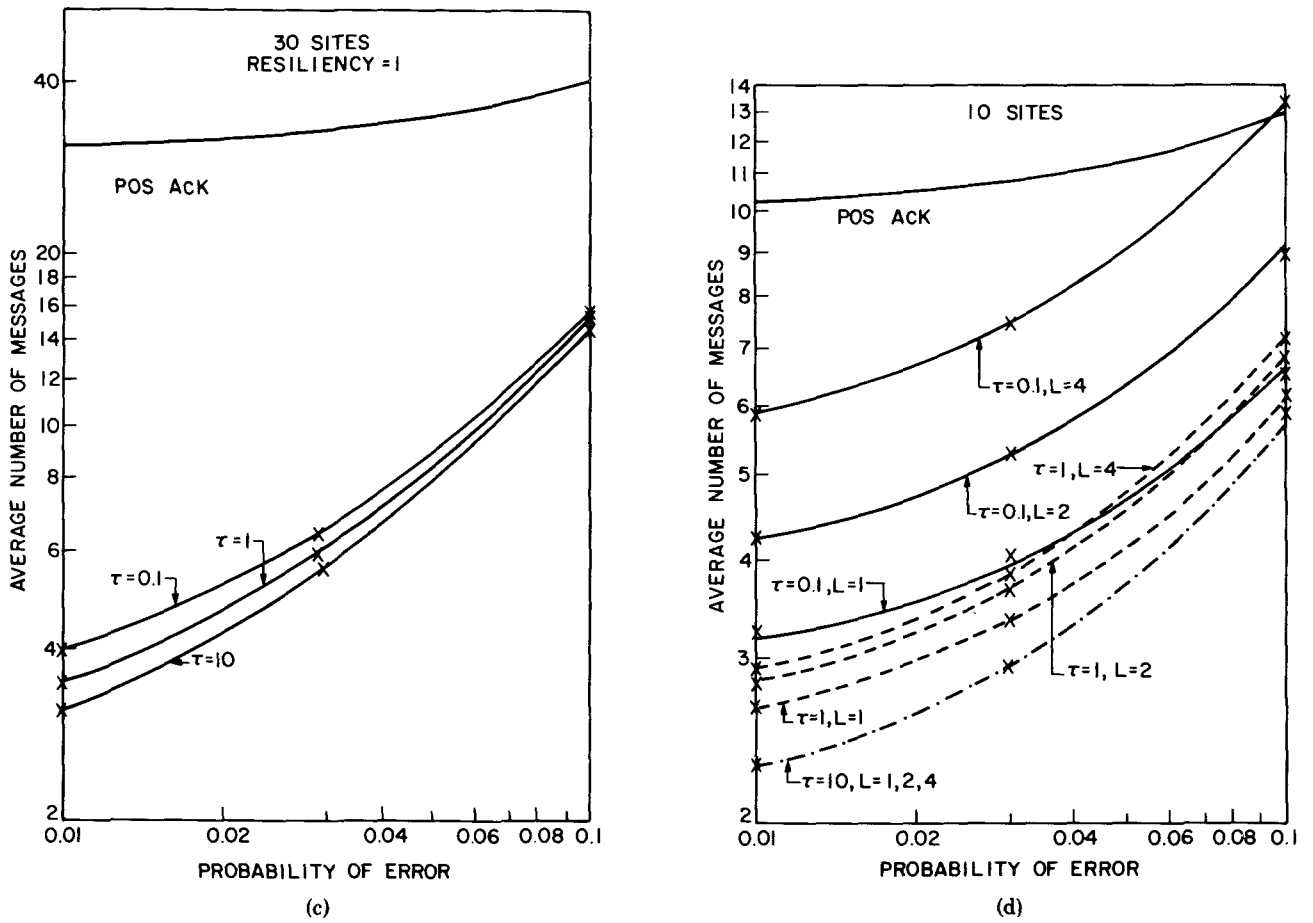


Fig. 6. Average number of messages transmitted per broadcast message as a function of probability of losing message for reliable broadcast protocol with $\tau = 0.1$, 1, and 10 and protocol which requires positive acknowledgment from each receiver. (a) System with 3 sites and resiliency of 1. (b) System with 10 sites and resiliency of 1. (c) System with 30 sites and resiliency of 1. (d) System with 10 sites and resiliencies of 1, 2, and 4.

have been implemented both on an operational ethernet and on a simulator. The simulation results closely match the prediction of an analytical model.

The protocols allow N receivers to reliably receive broadcast messages when less than N acknowledgments are sent, place the messages in the same sequence, and detect site failures within N token transfers. The proposed protocols thus provide a synchronized reliable broadcast communication environment with a failure detection facility. This environment can simplify the design of higher level distributed algorithms. The protocols have been used as the underlying communication mechanism for a distributed database system LAMBDA [3] which has been designed and is currently being implemented to provide a distributed database service on a local area network. The design of this system has been simplified by the reliable broadcasting of messages, the message sequencing, and the failure detection provided by these protocols.

APPENDIX. AN ANALYTICAL MODEL

In this appendix, an approximate analytical model of the broadcast protocol and a simple acknowledgment protocol are developed. This model is a refinement of an earlier model [5]. In the earlier approximation, many of the message requirements were calculated assuming that the token was not moving until responses were obtained. The earlier model is therefore referred to as the static model. The modified model assumes that the token is always being transferred and is referred to as the dynamic model.

In both the dynamic and static models, broadcast messages arrive with a Poisson process. The average number of token transfers and confirmation messages is calculated assuming that broadcast messages can be acknowledged as soon as they arrive. This results in an average number of acknowledgments and token transfers per broadcast message,

$$Y_a = \frac{1 - e^{-L\tau}}{1 - e^{-\tau}}$$

and an average number of confirmation messages,

$$Y_c = e^{-L\tau}$$

where L is the resiliency of the system and τ is the ratio of the period between token transfers to the average interarrival time of messages. These quantities are derived in [5]. In the simulations there are a finite number of sources which transmit a broadcast message until it is acknowledged. Therefore, the Poisson arrival process is distorted. In addition, a broadcast message is not acknowledged immediately if the site with the token does not receive the message, or if the token is being transferred and the next site misses the token transfer message or must recover missing messages. This occasionally results in a queue of waiting broadcast messages and a reduction in the number of times the token is transferred between acknowledgments.

In the static model it is assumed that a broadcast message is rebroadcast until an acknowledgment is received from one particular token site and that only sites which miss all of these transmissions must request a retransmission. However, when the system is busy, the token is transferred. If a site misses a broadcast

message and transfers the token, the next site may have received the message. In the implementation of the protocol, broadcast messages are not retransmitted while token transfers are received and a site never loses the messages it transmits. This results in very few broadcast messages being retransmitted. In the dynamic model, it is assumed that a broadcast message is only transmitted once ($n_b = 1$). The number of broadcast messages decreases, but the average number of sites which have missed the message increases. The average number of sites which must request the missing broadcast message is

$$n_{rb} = \left(N - 1 - \frac{N-1}{N}\right)P_e,$$

where P_e is the probability that a message is lost, and $1/N$ is the probability that a site acknowledges its own message. This approximation agrees more closely with the results of the simulations than does the static model.

In the static model, an acknowledgment is rebroadcast until a response is received from the next token site, and the previous token site receives the acknowledgment. Only sites which miss all of the retransmissions of the acknowledgment must request this message. In the implemented protocol, sites process acknowledgements in the order in which they are received. Therefore, if a site misses an acknowledgment, receives an acknowledgment with a higher timestamp, then receives a rebroadcast of the missing acknowledgment, it still requests the missing acknowledgment. This is reflected in the dynamic model, where it is assumed that any site which does not receive the acknowledgment when the next token site receives it will receive a higher order acknowledgment and will have to request this acknowledgment. The probability that the next token site receives the acknowledgment on the jth attempt is $(1 - P_e)P_e^{j-1}$. The average number of times the acknowledgment is transmitted before the next site receives it is

$$n_a = 1/(1 - P_e)$$

and the average number of sites which miss the acknowledgment is

$$n_{ra} = (N-2)\sum_{j=1}^{\infty}(1 - P_e)P_e^{j-1}P_e^j$$

$$= (N-2)\frac{P_e(1 - P_e)}{1 - P_e^2}.$$

A confirmation message is transmitted until the previous site receives it. Thus the average number of times a confirmation message is transmitted is $n_c = 1/(1 - P_e)$.

Retransmission requests are handled on a point-to-point transmission basis. When several sites have missed a message, more messages are required than if the retransmitted message were broadcast to all of the sites. However, in a local area network, messages not destined to a particular site are eliminated by the network interface. By using point-to-point transmissions for retransmitted messages, sites which have not missed a message do not have to process the

retransmitted message. The protocol was implemented this way because in a local area network, the processing power of sites on the network is a more valuable resource than transmission capacity. A retransmission request is transmitted until a response is received. The average number of times a retransmission request is transmitted is $n_{rq} = 1/(1 - P_e)^2$. The retransmitted message is transmitted until it is received by the requesting site. Therefore, the average number of times that a lost message is retransmitted is $n_{rx} = 1/(1 - P_e)$. The average number of messages transmitted each time a message is missed is

$$n_r = n_{rq} + n_{rx} = \frac{2 - P_e}{(1 - P_e)^2}.$$

The average number of messages transmitted for each broadcast message is

$$X_p = n_b + n_{rb}n_r + (n_a + n_{ra}n_r)Y_a + n_cY_c.$$

When $P_e \rightarrow 0$, the number of messages transmitted

$$X_p \rightarrow 1 + Y_a + Y_c$$

$$\rightarrow 1 + \frac{1 - e^{-(L+1)\tau}}{1 - e^{-\tau}}.$$

If, in addition, broadcast messages start arriving quickly with respect to the token transfer period, $\tau \rightarrow \infty$ and $X_p \rightarrow 2$. If, however, messages arrive infrequently, $\tau \rightarrow 0$ and $X_p \rightarrow 2 + L$. This shows the general characteristics of this broadcast protocol. When the system is operating properly, and very few messages are lost, there are very few messages transmitted per broadcast message. There are fewer messages per broadcast message transmitted when the system is busy than when it is idle.

The performance of the broadcast protocol is compared with a positive acknowledgment protocol. The positive acknowledgment protocol requires an acknowledgment from each of the receivers. It guarantees that every receiver obtains every broadcast message, but does not guarantee that the messages are obtained in the same sequence at every receiver. In an error-free environment, a positive acknowledgment system requires a broadcast message and $N - 1$ acknowledgments. Unlike the broadcast protocol, this number of messages is required no matter how busy the system is. How the number of messages increases when messages are lost depends upon the retransmission strategy. In this comparison, it is assumed that missed messages are handled on a point-to-point basis, as in the broadcast protocol.

The broadcast message is transmitted once. The average number of sites which receive the message and transmit an acknowledgment is $n_{a1} = (N - 1)(1 - P_e)$. Sites enter the point-to-point retransmission mode if they miss the broadcast message, or if the broadcast source misses their response. The average number of sites which enter this mode is $n_{rq} = (N - 1)(1 - (1 - P_e)^2)$. Once in this mode,

a site requires an average of n_r messages, as in the broadcast protocol. Therefore, the average number of messages per broadcast message is

$$X_b = 1 + n_{a1} + n_{rq}n_r$$

$$= 1 + (N - 1)\frac{1 + P_e - P_e^2}{(1 - P_e)^2}.$$

REFERENCES

1. CHANG, J.M., AND MAXEMCHUK, N.F. Reliable broadcast protocols. Tech. Rept. AT&T Bell Laboratories, Murray Hill, N.J., Jan. 1983.
2. CHANG, J.M., AND MAXEMCHUK, N.F. A broadcast protocol for broadcast networks. In *Proceedings of GLOBCOM* (Dec. 1983).
3. CHANG, J.M. Simplifying distributed database systems design by using a broadcast network. In *Proceedings of ACM SIGMOD* (Boston, Mass., June 1984). ACM, New York, pp. 223–233.
4. KONG, I. Cablenet: A local area network. In *Proceedings of the International Computer Conference* (June 1982), pp. 6C.2.1–6C.2.5.
5. MAXEMCHUK, N.F., AND CHANG, J.M. Analysis of the messages transmitted in a broadcast protocol. In *Proceedings of the International Computer Conference* (Amsterdam, May 1984). pp. 1263–1267.
6. METCALF, R.M., AND BOGGS, D.R. Ethernet: Distributed packet switching for local computer networks. *Commun. ACM 19*, 7 (July 1976), 395–404.
7. PENNEY, B.K., AND BAGHDADI, A.A. Survey of computer communications loop networks: Part I and Part II. *Comput. Commun. 2*, 4 (Aug. 79), 165–241.
8. TANNENBAUM, A.S. *Computer Networks.* Prentice-Hall, Englewood Cliffs, N.J., 1981.
9. TOBAGI, F.A. Multiaccess protocols in packet communication systems. *IEEE Trans. Commun. COM-28* (April 1980), 468–489.

Received March 1983; revised April 1984; accepted April 1984