An Ordered and Reliable Broadcast Protocol for Distributed Systems [1]

Ye-In Chang and Ming-Hon Hwang
Dept. of Applied Mathematics
National Sun Yat-Sen University
Kaohsiung, Taiwan
R.O.C.
{E-mail: changyi@math.nsysu.edu.tw}
{Tel: 886-7-5316171 (ext. 3710)}
{Fax: 886-7-5319475}

Abstract

The purpose of a reliable broadcast protocol is to allow groups of nodes on unreliable broadcast networks to reliably broadcast messages. A reliable broadcast protocol must guarantee two properties: 1) all of the receivers in a group receive the broadcast messages, and 2) each of the receivers orders the messages in the same sequence. In an optimistic approach to reliable broadcast protocol, a batch acknowledgement is employed for a sequence of broadcast messages, instead of one or more acknowledgement per broadcast message used in the pessimistic approach. In this paper, based on the optimistic approach, we have proposed a counter-based reliable broadcast protocol. In this protocol, the unique token ownership is circulated among all the nodes in an order specified by a token-passing-list. The *system state* which records related information about messages broadcast by each node is included in the token message. By appropriately updating the counter information recorded in the *system state* included in the token message, instead of using explicit acknowledgement messages, the proposed protocol needs fewer control messages to commit a broadcast message than other protocols no matter the rate of transmission errors is high or low. Moreover, we show how to handle the flow control problem, and the token update technique.

(Key Words: agreement problems, broadcast communication, distributed database systems, distributed operating systems, fault tolerance, multicast communication)

[1] This research was supported by the National Science Council of the Republic of China, Grant No. NSC-83-0404-E-110-007

1 Introduction

Broadcast communication is the delivery of copies of messages to all the nodes in a network and is used in many applications, for examples, updating on replicated data, broadcasting of real-time news, and computer conferencing [2, 3, 7, 15, 18]. For many applications, it is important to receive all broadcast messages for all receiving nodes and accept those messages in the same order, for example, updating on replicated data. However, it is possible to occur that some broadcast messages are not received by all the nodes and receiving nodes receive those messages in a different order because of the transmission delay (or transmission errors) or because a buffer overflow within a receiver. Therefore, how to design a broadcast protocol which is reliable is an important topic in distributed systems.

Over the last decade, many reliable broadcast protocols for different kinds of networks have been proposed. Segall et al. [27] have proposed a protocol for point-to-point networks, in which they extend adaptive routing [24] to be reliable. Chang et al. [6] have proposed a protocol for Ethernets or satellite networks, in which all messages pass through an intermediary node (called the token node); the token node determines the global order of messages. Melliar-Smith et al. [22, 23] have proposed a protocol for local area networks (LANs), such as Ethernet or a token ring; based on the property of LANs, the protocol piggybacks (positive or negative) acknowledgements to old broadcast messages on new ones and guarantees that all nodes construct the same partial order of broadcast eventually. Birman et al. [4, 5] have proposed a protocol (called ABCAST) for both local- and wide-area networks, which is similar to the two-phase commitment. Luan et al. [20, 21] have proposed a protocol for an arbitrary network of "fail-stop" nodes which is based on the three-phase majority-consensus decision to commit on a unique order of received broadcast messages. Chow et al. [9] have proposed a two-phase protocol which consists of the broadcast phase and the reply phase; the protocol needs only bounded message buffer space if the network is eventually connected in time that is finite and bounded.

The above protocols can all be classified into one approach: the pessimistic approach. In the pessimistic approach, a broadcasting node must receive an explicit acknowledgement from every receiving node. Therefore, this approach needs many explicit acknowledgement

messages. In [10], Chung et al. have proposed another approach to reliable broadcast protocol: the optimistic approach. In general, broadcast protocols based on the optimistic approach employ a batched acknowledgement, instead of using an explicit acknowledgement for each broadcast message. In Chung et al.'s protocol, a receiving node acknowledges all received messages when it becomes the token owner rather than one message at a time. Therefore, the optimistic approach needs fewer control messages than the pessimistic approach and is a preferred approach when the rate of transmission errors is low. However, when the rate of transmission errors is high, a lot of rollback operations must be executed (and many rollback messages will be sent out), and the system performance will be degraded in Chung et al.'s protocol. Based on the optimistic approach, Amir et al. [1] have also proposed a token-based reliable broadcast protocol. In this protocol, the total order is achieved by including a sequence number in a token circulated around a logical ring, and message loss is also detected by using the sequence number. However, in this protocol, only the node in possession of the token can broadcast a message to the other nodes on the logical ring.

In this paper, based on the optimistic approach, we propose a counter-based reliable broadcast protocol for an unreliable network (which can be a broadcast network, or a point-to-point communication network). The protocol operates between the application programs and the network. It isolates the application programs form the unreliable characteristics of the communication network. In this proposed protocol, the unique token ownership is circulated among all the nodes in an order specified by a token-passing-list. The *system state* which records related information about messages broadcast by each node is included in the token message. By appropriately updating the counter information recorded in the *system state*, instead of using explicit acknowledgement messages, the proposed protocol needs few control messages to commit a broadcast message averagely no matter the rate of transmission errors is high or low. (Note that a broadcast message can be committed only if the order of the message is determined by all the nodes in the system.) The proposed protocol can detect a message loss in at most two-cycles time, where a cycle is defined as the time interval in which a node becomes the token owner again. Moreover, we show how to handle the flow control problem, and the token update technique.

The rest of the paper is organized as follows. Section 2 describes the basic idea of the proposed protocol. Section 3 presents the proposed protocol formally, describes the way to handle the flow control problem and describes the token update technique. Section 4 discusses the performance of the proposed protocol. Section 5 discusses the level of reliability that the proposed protocol can achieve. Finally, Section 6 gives the conclusion.

2 Basic Idea

In this section, we first describe assumptions made in our protocol. Next, we define the data structure of the *system state* which is included in the token message. We then describe the basic idea of the proposed protocol by an example.

2.1 Assumptions

Our discussion on reliable broadcast protocols are based on the following assumptions [8, 11, 25, 26].

1. Each node has a unique ID and a complete list of the IDs of the other nodes in the network, but has no information about the complete network topology.

2. Message delay is finite but unpredictable. Moreover, between any pair of nodes, messages may not be received in the order in which they are sent.

3. An error checking mechanism is used to detect corrupt messages. A corrupt message is treated as a lost message, as if it has never been received. That is, a message is always regarded as a correct one once it is received.

4. Communication connections (i.e., links) may fail causing messages to be lost or delayed but will not cause network partition. A communication failure can be recovered in finite time. When a communication link failure occurs, it can be detected by the nodes on both ends.

5. A failed node simply stops execution (i.e., a fail-stop system). In particular, a node may fail to broadcast or forward some messages but it will not create or transmit

Figure 1: The data structure of each element of the system state in the broadcast protocol

erroneous messages. That is, no Byzantine failure occurs. A node failure can be detected and recovered in finite time.

6. Reliable broadcast protocols make use of unreliable broadcast primitives. An unreliable broadcast primitive delivers broadcast messages to all receiving nodes but does not guarantee that each message is received by every receiving node. A reliable broadcast primitive, on the other hand, ensures that all broadcast messages are correctly received by all receiving nodes.

7. The broadcast protocol does not involve programming the network switches. As a consequence, the protocol has to be implemented at the host level.

2.2 Definitions

Basically, the proposed protocol makes use a token message to reduce the number of acknowledgement messages. The token ownership is circulated among all the nodes in an order specified by a *token-passing-list*. The data structure of each element of the *system state* which is included in the token message is shown in Figure 1. There are three fields in the system state: the message index field, the cycle field and the counter field. The messages index field records a sequence of message identifiers and the node identifiers x from which the messages are sent out. The cycle field records which cycle the related messages are sent out, where a cycle is the time interval that the same node x becomes the token owner again from the last time. The counter field records the number of nodes which have received those messages specified in the messages index field and the cycle field. (Note that the value of those three fields can be updated only when a node becomes the token owner.) Suppose the message index field contains "a1a2a3", the cycle field contains p and the counter field contains 3. It means that there are three nodes which have received all the messages with identifiers a1, a2 and a3 that are broadcast by node A in cycle p.

Figure 2: An Example

2.3 An Example

In this example, we assume that there is no message loss, no delayed message and no node failure. Suppose there are three nodes A, B, C in the system, the current token owner is node A, the token-passing-list is (A, B, C) and the current cycle of node A is (p+1). Figure 2 shows the change of the system state during cycle p of node A.

In Figure 2-(a), the token owner A broadcasts a token (to transfer the token) message to node B, and the current system state is included in the token message in which the current message index field contains "a1a2a3", the cycle field contains p and the counter field contains 0. It means that node A has broadcast messages with identifiers "a1a2a3" in cycle p and no node has acknowledged the messages. (Note that a node acknowledges a message by increasing the value of related counter by one when it has received the message and holds the token.) In Figure 2-(b), node B has become the token owner. Since during cycle p, node B has broadcast messages with identifiers "b1b2b3", node B adds such information into the system state. Moreover, node B increases the value of node A's counter field by one since node B has already received such messages. Node B then broadcasts the token message including the new system state to all the nodes.

Similarly, Figure 2-(c) shows the system state included in the token message broadcast by node C. From this figure, we know that node C has received messages with identifiers

"a1a2a3" and identifiers "b1b2b3" in cycle p that are broadcast by node A and B, respectively, and has sent out messages with identifiers "c1c2" in cycle p. At this time, every node receiving the token message can ensure that the messages broadcast by node A in cycle p have been received by every node in the system, since the value of the node A's counter field is equal to 2.

Next, node A becomes the token owner again. At this moment, node A increases the value of current cycle by one to (p+2), adds information about messages identifier with "a4a5" sent out in cycle (p+1) to the system state and increases the value of the counter field of every other node by one since it has received all the messages. Moreover, since the value of node A's counter field in cycle p is equal to the total number of all other nodes in the system, i.e., every node has received such messages broadcast by node A in cycle p, node A deletes the related information from the system state.

In the same way, Figures 2-(d) and 2-(e) show the system state included in the token message broadcast by node A and node B, respectively. Particularly, at this moment, since every node has received all the messages broadcast in cycle p, every node can commit all the messages (a1a2a3, b1b2b3, c1c2) sent out in cycle p when it receives the token message broadcast by node B.

Note that a node *commits* a message when it can ensure that the order of the message is determined by all the nodes in the system, i.e., at this point, it can send this message to the application in the upper layer of the system. In the proposed protocol, when the messages broadcast in cycle p have been committed, it means that every node has received the same set of broadcast messages. At this time, the order of those messages can be determined locally in the same sequence by using some predefined rules. For example, for a sequence of messages sent out from the same sender, their order can be determined by the sequence number used locally at the sender. For the messages sent out from different nodes, their order can be determined by a predefined rule. One simple way to do so is to let the order of messages broadcast by different nodes always follow the same order of the token-passing-list [10], that is, in the example, the order of messages broadcast in cycle p is determined as (a1a2a3, b1b2b3, c1c2). The other possible way to determine the order of messages broadcast by different nodes is to let this order also follow the same circular order

of the token-passing-list but starting from a different node at each time. For example, suppose the order of broadcast messages in cycle (p-1) follows the circular order of the token-passing-list starting from node B. Then in the example, the order of the messages broadcast in cycle p is determined as (c1c2, a1a2a3, b1b2b3) since at this time, the order should start from node C.

In the case that a message loss is detected by a node y, it broadcasts a request-for-retransmission message which specifies the identifier of the original message sender, message identifiers, the cycle number and node y's own identifier to ask for retransmission. (Note that since a node x sending messages out in cycle p will include such information in cycle (p+1) when it becomes the token holder, other nodes y can detect a message loss from node x when y receives the broadcast token message. This is done by comparing the message identifiers which have been received with those recorded in the system state included in the token message. Therefore, at the time when node y detects such a message loss, it also knows from which node and in what cycle it should ask for retransmission.) A node receiving the request-for-retransmission message and with the identifier of the original message sender equal to itself, will broadcast the lost message again. (Note that at this time, node y does not increase the value of node x's counter field by one when node y becomes the token owner; therefore, node x will not commit the messages until node y receives the messages and increases the value of node x's counter field by one.) Moreover, the information about lost messages broadcast by node x is not removed from the system state until node x becomes the token owner again and the value of node x's counter field is equal to what it expects.

Basically, there are two situations that a node can commit the messages broadcast in cycle p. First, if the system state included in the token message shows that every node has received all the messages broadcast in cycle p, and the messages broadcast before cycle p have been committed, then a node can commit all the messages broadcast in cycle p when it receives the token message. Second, in the case that a node does not receive some broadcast token messages, then it still can commit the messages broadcast in cycle p when the system state included in the token message does not contain the information about cycle p (i.e., the information about cycle p has been deleted from the system state by the

senders) and all the messages broadcast before cycle p have been committed.

3 The Protocol

In this section, we first give a formal description of the protocol. Next, we consider the problem of flow control, and then the token update technique.

3.1 The Formal Description

In this section, we give a formal description of the protocol. There are three types of messages in the system, the broadcast message Msg, the token message T and the request-for-retransmission message Req. The token message $T(c, n, t, S)$ contains the current token owner identifier c, the last token owner identifier n, the current cycle t, and the system state S. A broadcast message $Msg(i, rec, p, k, msg)$ contains the sender identifier i, the list rec of the destination nodes' identifiers in the broadcast group, the cycle number p, the message identifier k and the message body msg. If there is a message to be broadcast at any time then broadcast it. The request-for-retransmission message $Req(j, i, p, k, n)$ contains the sender identifier j, the receiver identifier i, the cycle number p, the message identifier k and the request number n. Assume that there are N nodes in the system. There are four possible events which can occur during each cycle at a node x:

1. The arrival of a token message.

2. The arrival of a broadcast message.

3. The arrival of a request-for-retransmission message.

4. The expiration of the token-owner timer.

In the event of the arrival of a token message, when a token message arrives at a node x, node x checks whether it is the new token owner. This can be done by comparing its own identifier with the next token owner identifier included in the token message. If it is, node x resets its token-owner timer and increases the cycle number. (Note that the token-owner-timer is used to avoid the case that a node holds the token ownership for too long. When a node owns the token, it can update the system state included in the token

message.) Moreover, no matter node x is the new token owner or not, it can compare its local information (about what messages it have received) recorded in its $DataList$ with the system state when it receives the broadcast token message. If an inconsistency is detected, node x sends a request-for-retransmission message; otherwise, node x records those node identifiers from which node x has received all the broadcast messages in $IdList$. If all the messages broadcast in cycle p have been received by every node and all the messages broadcast before cycle p have been committed, then node x commits all the messages broadcast in cycle p.

In the event of the arrival of a broadcast message at a node x, if there is no such a message in its $DataList$, then node x inserts it to its $DataList$; otherwise, node x discards the message. In the event of the arrival of a request-for-retransmission message, the node retransmits the required message if the message receiver is itself and this is a new request which can be detected by checking the identifier of the request-for-retransmission message.

In the event of the expiration of the token-owner timer, the node must update the system state S. It contains two operations: One is to update the counter field that the related messages have been received. The other is to attach the messages that this node has sent during the previous cycle to the system state S. Then the node passes the token ownership and the new system state to the next token owner.

The protocol is described as follows:

```
/* for node x */ /* there are N nodes in the system */
/* the cycle number for node x is Px */
loop forever
;***********************************
if there are messages to send then send them;
;***********************************
case T(c, n, t, S):
if ((n = x) and (t = Px + 1)) or
   ((x is the start node) and (n = x) and (t = Px)) then
                       /* c is the current token owner id */
                       /* n is the next token owner id    */
                       /* t is the current cycle number   */
                       /* S is the system state           */
                       /* start node is the first node of the token-passing-list */
                       /* Px is the current cycle number of node x */
   {
     reset the token-owner timer;
     if counter in S equals to what node x expects in cycle Px
        {
```

```
         remove the related information from S;
        }
     Px = Px+1;
   }
if node x has received all the messages sent by node i in cycle Pi then
  {
  IdList = IdList + (i,Pi);
  }
commit the messages sent in a certain cycle if it is possible;
if the local information of the DataList is not consistent with the
   system state S in cycle (t-1) then
   {
loop:set request timer;
     send Req(x, i, t-1, k, n);
                         /* x is the sender, t is the cycle number */
                         /* k is the lost message identifier */
                         /* i is the receiver, n is the request number */

  if request timer is out and node x has not received the requested data then
        {
        n = n + 1;
        goto loop;
        }
   }
;************************************
case Msg(i, rec, p, k, msg):
                         /* i is the sender,
                            rec is the list of the
                            destination nodes' identifiers in the broadcast group,
                            p is the cycle number,
                            k is the message identifier,
                            msg is the message body */
  if there is no such a msg in the  DataList then insert it
     else discard it;
;************************************
case Req(j, i, p, k, n):
if (i = x) and (this is a new request) then
                          /* we use 'n' to recognize whether
                             the request is the new request or not. */
   {
    send Msg(x, rec, p, k, msg);
                         /* x is the sender, p is the cycle number,
                            k is the message identifier, msg is a message body */
                         /* rec is the list of the
                            destination nodes' identifiers in the broadcast group */
   }
;************************************
case token-owner timer has expired:
{
/* update system state S: */
  {
     for every node i for any message in IdList do
```

```
       {
       increase the counter field of node i of message Mi of cycle Pi by 1;
       IdList = IdList - (i, Pi);
       }
    S = S + messages sent out in the previous cycle by node x;
  }
broadcast token message T(x, y, Px, S)
                         /* y is the one next to x in the token-passing-list */
}
;*************************************
endloop;
```

3.2 Flow Control

A basic characteristic of reliable broadcast is that the rate of broadcasting messages cannot exceed the rate at which the slowest processor can receive messages [1]. At faster rates of broadcasting, the input buffer of the slowest receiver will become full and messages will be lost. Retransmission of these messages will result in an increase in message traffic and a reduction in the useful transmission rate. Therefore, a broadcast protocol without an effective flow control mechanism will easily overwhelm the slowest receiver. The flow-control mechanism should allow a high transmission rate when all nodes are able to process messages at that rate. However, to prevent buffers from overflowing and messages from being lost, the mechanism must dynamically restrict the flow of messages to the rate at which messages can be processed. The response time of the flow control mechanism is determined by the size of the message buffer in each node. To ensure that buffer overflow does not occur, during the interval from the time at which a node's buffer becomes empty and the next flow control message transmitted by that node, the other node must not broadcast more messages than that node's buffer can contain. Consequently, each node must transmit flow control information at least once during any sequence of messages that is sufficient to fill its buffer. To satisfy this requirement, in our flow control algorithm, we include the following four fields into the token message:

1. The local field: it is an array with N entries; each entry x indicates that the message buffer at node x is nearly overflow.

2. The localC field: it is an array with N entries; each entry x records the cycle number that the related local field is set.

3. The global field: it indicates that the current network is nearly overloaded.

4. The globalC field: it records the cycle number that the global field is set.

When a node becomes the token owner and finds that its message buffer is nearly overflow, it will set its local field included in the token message to 1 and also record its cycle number in the related localC field at the expiration of the token-owner timer. If the node holding the token detects that the network is nearly overload, which can be detected by counting the number of messages recorded in the system state, it will set the global field to 1 and update the globalC field to the current cycle number at the expiration of the token-owner timer.

When a node receives the token message and finds that the entry x of the local field is set to 1, it will not send any message to node x. A node can start to send messages to node x when it receives a token messages in which the current cycle number is larger than the cycle number recorded in entry x of the localC field and entry x of the local field is set to 0. When a node receives a token message with the global field equal to 1, it stops to send any message. A node can start to send messages when it receives a token message in which the current cycle number is larger than the cycle number recorded in the globalC field and the global field is set to 0.

When a node x becomes the token owner and is not suffering the buffer overflow problem, it resets entry x of the local field and the localC field to 0. If the node holding the token detects that the network is not overload now, it resets the global field and the globalC field to 0 at the expiration of the token-owner timer.

3.3 Token-Passing-List and Token Update

The token-passing-list specifies the order in which the token ownership is circulated among nodes. There are three cases which will change the contents of the token-passing-list [19]: (1) insertion of a node, (2) deletion of a node, and (3) a token failure.

In case one, when a node x wishes to participate the broadcast system, it listens for a

token message. When a token message is received, it then sends a participation request message to the current token owner. When the current token owner receives the participation request message, it creates a new token-passing-list including the new node identifier x and sends the new list to all nodes in the token message. When a node receives this token message, it will add the new node identifier x to its token-passing-list and update the information about the number of the nodes in the system.

In case two, when a node x wishes to drop out of the broadcast system, it waits until its turn to be the token owner. Node x then broadcasts a token message including a new token-passing-list with its own node identifier deleted from the list. It leaves the broadcast system after this token period. Each node receiving the token message updates its own token-passing-list to delete the old node identifier x and update the information about the number of the nodes in the system.

A token message may be delayed and a token owner may fail. These failures will cause the faulty conditions where no node is the token owner (i.e., the lost token condition) or two or more nodes are current token owners (i.e., the duplicate token condition). To handle these failures, in case three, after the token owner x broadcasts the token message, it listens for a predetermined period (which is longer than the token period) to make sure that the next token owner broadcasts a token message [19]. When node x receives no token message within this predetermined period, it concludes that the token is lost. Then node x reissues a token message after a randomly chosen delay. When the node x does not receive a token message from the next token owner after executing the above action for L times ($L > 0$), it concludes that the next token owner has failed. Node x then generates a new token-passing-list with the failed node deleted and sends the list with the token message to the node which is next to node x in the current token-passing-list. In order to recover from the duplicate token condition, the token owner x checks whether the token ownership is returned to itself after exactly N token periods. If the token ownership is returned before N token periods, node x concludes the presence of duplicate tokens. Node x gives up its token ownership immediately. This operation may cause the lost token condition. In that case, the lost token recovery action mentioned before is carried out to regenerate the token.

4 The Performance

In this section, we first show the performance analysis and the simulation results of the proposed protocol, then we compare the proposed protocol with Chang et al.'s, Birman et al.'s, Chung et al.'s and Amir et al.'s protocols [1, 4, 6, 10], where the first two protocols [4, 6] belong to the pessimistic approach and the last two protocols [1, 10] belong to the optimistic approach.

4.1 Performance Analysis

In this performance study, two models are considered: a broadcast network and a point-to-point network. For each model, we look specially at two performance measures: (1) C, the average number of control messages required to commit a broadcast message which includes token messages and request-for-retransmission messages; (2) D, the time elapsed between the beginning of the broadcast message and the time when all the nodes in the system can make the message ready for local delivery.

In our performance model, we have the following assumptions. There are N nodes in the system. The token-owner time is denoted as T. Message propagation delay between any two nodes is denoted as P. Moreover, we assume that the request to broadcast a message arrive at a node according to Poisson distribution with a parameter A. That is, 1/exp(A, N) is the number of broadcast messages per second, where *exp* is an exponential distribution function.

In a broadcast network, when no failure occurs, C in our protocol is equal to

$$1/(N^*(T+P)/\exp(A, N)) \; (= \exp(A, N)/(N^*(T+P))),$$

that is, during one token owner time T plus one message propagation time P, each of N nodes with a broadcast request rate A can broadcast (T+P)/exp(A, N) messages, for a total of N*(T+P)/exp(A, N) messages, and those messages are batch acknowledged by one token message. From this formula, we find that as T is increased, C is decreased. The reason is that when the token-owner time is increased, more messages can be broadcast during that time, which are also acknowledged by one batched token message. Moreover, as N is increased, C is decreased. Furthermore, as A is increased, C is decreased. The

reason is the same as above.

D in our protocol is equal to

$$(2*N*(T+P))/(N*N*(T+P)/\exp(A, N)) \ (= 2*\exp(A, N)/N),$$

that is, during two-cycles time (i.e., $2*N*(T+P)$), up to $(N*N*(T+P)/\exp(A, N))$ messages can be broadcast by N nodes in the first cycle and committed at the end of the second cycle. From this formula, we find that D will be the same no matter how T is changed. Moreover, the system with a higher arrival rate (A) will need a smaller D than a system with a lower arrival rate. Furthermore, as N is increased, D is decreased.

The above performance analysis is based on the assumption that no error occurs. Let E be the error rate of communication failures which will result in a message loss. In this case, message retransmission is required. When E is considered, in a broadcast network, C in our protocol is equal to

$$(1/((N*(T+P))/\exp(A, N)))*(1+E+E^2+...)+(N-1)*(E+E^2+E^3+...)$$
$$(= \exp(A, N)/(N*(T+P)*(1-E))+(N-1)*E/(1-E)),$$

where the first item is the number of the token messages and the second item is the number of request-for-retransmission messages. Note that those retransmitted messages may be lost (with an error rate E) again; therefore, the formula has an infinite equation. However, since $E < 0$, $1+E+E^2+... = 1/(1-E)$. From this formula, obviously, as E is increased, C is increased. Moreover, as N is increased, C is increased. The reason is that when E is considered, the number of request-for-retransmission messages will dominate the performance.

In a point-to-point network, when no failure occurs, the control messages C in our protocol is equal to

$$(N-1)/((N*(T+P))/\exp(A, N)) \ (= (N-1)*\exp(A, N)/(N*(T+P))).$$

Since in this case, (N-1) token messages must be sent to the other (N-1) nodes. Moreover, since the time ε spent in processing an incoming or outgoing message is negligible as compared to the message propagation delay P (i.e., $\varepsilon << P$), the delay D of our protocol in a point-to-point network, is the same as that in a broadcast network. When E is considered, in a point-to-point network, C in our protocol is equal to

$$((N\text{-}1)/((N^*(T+P))/\exp(A, N)))^*(1+E+E^2+...)+(N\text{-}1)^*(E+E^2+E^3+...)$$
$$(= (N\text{-}1)^*\exp(A, N)/(N^*(T+P)^*(1\text{-}E))+(N\text{-}1)^*E/(1\text{-}E)),$$

where the first item is the number of the token messages and the second item is the number of request-for-retransmission messages.

4.2 Simulation Results

In this simulation study, message propagation delay (P) between any two nodes is a constant (PP) times a random number (between 0 and 1) with a uniform distribution to simulate the unpredictable message delay. The token-owner time (T) is a constant. There are N nodes in the system. The events of broadcasting messages at every node are created according to the Poisson distribution with a parameter A. That is, 1/exp(A, N) is the number of broadcast messages per second. Moreover, we only consider the case of communication failures resulting in a message loss or delayed messages; we do not consider node failures. To simulate the case of a message loss, we use a random function with a uniform distribution, and then compare the value v (between 0 and 1) returned from the random number function call with the chosen error rate E (also between 0 and 1) for each node x. If the value v is smaller than the chosen error rate E, then we let the broadcast message to a certain node x be lost, i.e.; we do not deliver it to node x.

This simulation (written in SIMPAS) runs on a simulated broadcast network and a point-to-point network on a single machine. Simulation experiments were carried out for a homogeneous system of 10 nodes (N=10) and 20 nodes (N=20) for various values of the error rate (E). Message propagation delay (PP) between the nodes was taken as 0.1, the arrival rate (A) to broadcast a message was taken as 10 and 20 and the token owner time (T) was taken as 1 and 5, respectively. For the performance measure, we collect the values of these variables for 30000 broadcast messages.

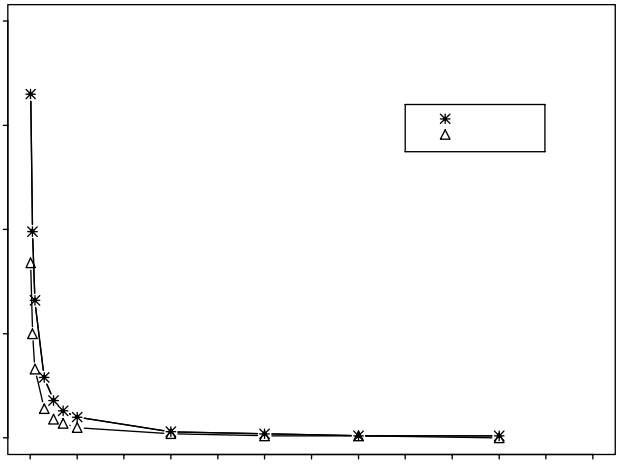
Figure 3 shows the average number of control messages (C) as a function of the token owner time (T) with different values of the arrival rate (A) in a broadcast network when no error occurs. As T increases, C decreases, which is as what we expected in the performance analysis. Moreover, the control messages C in Figure 3-(a) is about twice of those in Figure 3-(b) because the number of nodes in Figure 3-(a) is half of that in Figure 3-(b). When in

the normal condition (i.e., E = 0), the cost is smaller than 0.1 in all the cases.

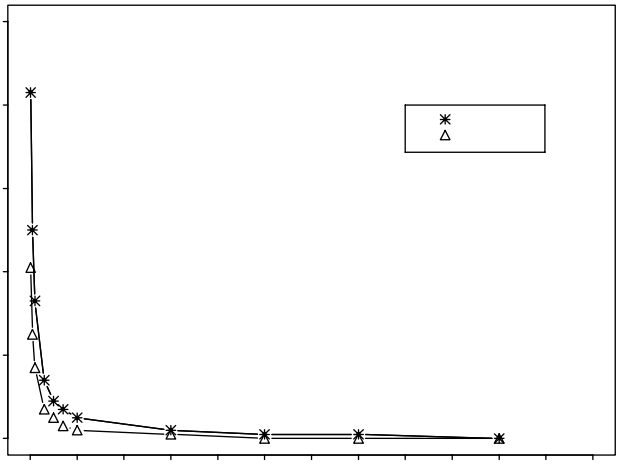
Figure 4 shows the average number of control messages per broadcast message as a function of the probability of errors given a fixed value of the arrival rate to broadcast a message and a fixed value of token owner time. As we can see, the cost increases as the error rate increases since the number of request-for-retransmission messages is proportional to the error rate. When N is increased, the cost is also increased based on the same reason. Moreover, given the same N, the cost in a broadcast network is also almost the same as that in a point-to-point network based on the above reason.

4.3 A Comparison

Table 1 shows a comparison of performance of our protocol with Chang et al.'s (noted as centralized), Birman et al.'s (noted as two-phase), Chung et al.'s (noted as optimistic) and Amir et al.'s (noted as fast message) protocols [1, 4, 6, 10] in a broadcast network, where Num equals to the sum of one broadcast message and the average control messages C per broadcast message. Chang et al.'s centralized protocol [6] requires simply two messages: one broadcast message from the source to the central node and one acknowledgement message from the central node to the rest of the members of the destination group. Birman et al.'s two-phase commit protocol [4] requires one broadcast message from the source to (N-1) destinations, (N-1) messages containing the local highest sequence number from the destinations to the source, and one message containing the global highest sequence number back from the source to the destinations, for a total of (N+1) messages. Chung et al.'s optimistic protocol [10] requires (1+1/N) messages to commit a message averagely, since a batch acknowledgement message (i.e., the token message) is used and a node can send out one broadcast message during one token owner time. Amir et al.'s fast message ordering protocol [1] requires 2 messages to commit a message averagely, since a batch acknowledgement message is used and a node can send out a broadcast message only when it holds the token. Figure 5 shows a comparison of Num among the centralized, optimistic, fast message protocols and two cases of our counter-based protocol (with an arrival rate A = 10 and 20) under different values of N in a broadcast network, where $N \leq 50$. (Note that since Num in the two-phase protocol is much larger than that in all other protocols,

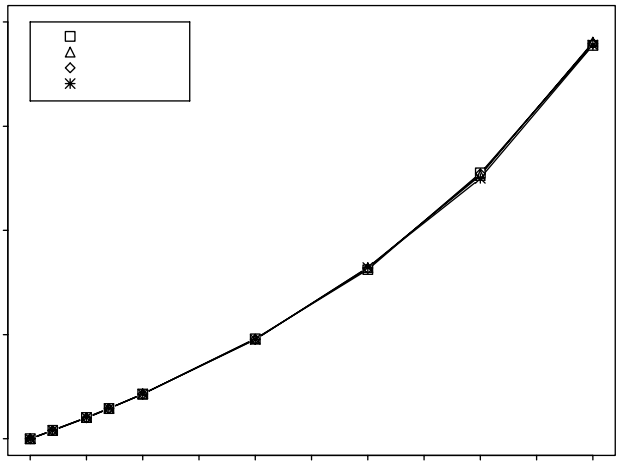


(a)

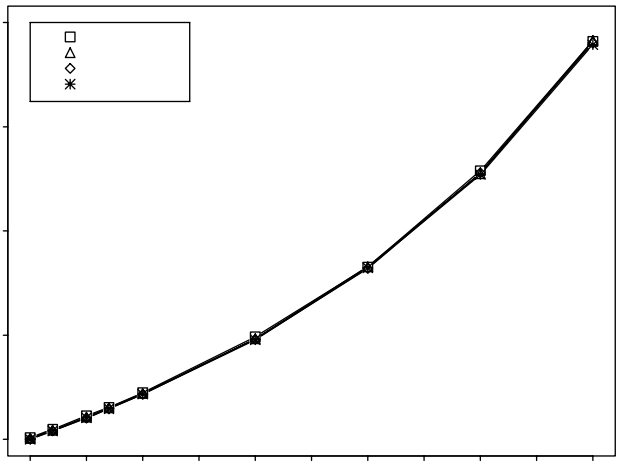


(b)

Figure 3: The average number of control messages of the simulation results in a broadcast network: (a) N=10; (b) N=20.



(a)



(b)

Legend: arrival rate (A) - token owner time (T)

Figure 4: The average number of control messages of the simulation results when E >= 0: (a) N = 20, a broadcast network; (b) N = 20, a point-to-point network.

approach	pessimistic		optimistic		
algorithm	centralized [6]	two-phase [4]	optimistic [10]	fast message [1]	counter-based
Num	2	$N+1$	$1+1/N$	2	$1+exp(A,N)/((T+P)*N)$
D	2P	$3P$	$T/N+P/N$	T+P	$2*exp(A,N)/N$

Table 1: A comparison of performance of reliable broadcast protocols in a broadcast network

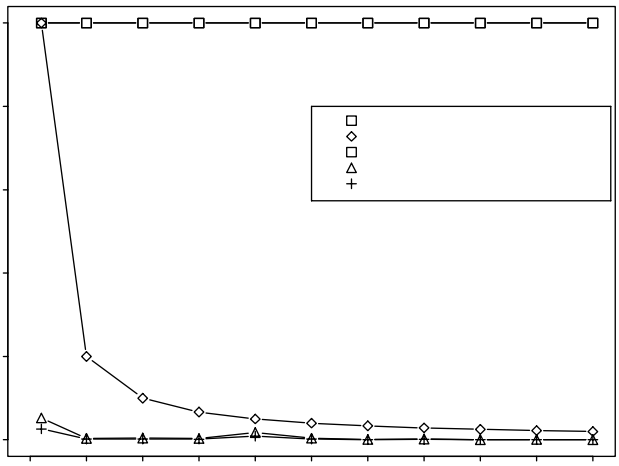


Figure 5: A comparison of Num in a broadcast network

we do not include it in this comparison.) From this figure, we show that the counter-based protocol requires the fewest number of Num among those protocols. Moreover, as N is increased, Num is decreased in the optimistic protocol.

Due to the characteristics of broadcast networks, each message has the same delay. This delay consists of the time for the sender to put the message on the network, and the time for the message to get to the other nodes (i.e., P). We assume that the first item is much shorter than the second item; therefore, we let P be this total delay. The delay in Chang et al.'s protocol [6] is 2*P and Birman et al.'s protocol [4] is 3*P. Chung et al.'s protocol [10] requires (T/N+P/N) time to commit a message averagely. Amir et al.'s protocol [1] requires (T+P) time to commit a message averagely. Figure 4.3 shows a comparison of D

Figure 6: A comparison of D in a broadcast network

approach	pessimistic		optimistic		
algorithm	centralized [6]	two-phase [4]	optimistic [10]	fast message [1]	counter-based
Num	N	$3*(N-1)$	$(N-1)+(N-1)/N$	$2*(N-1)$	$(N-1)+(N-1)*exp(A,N)/((T+P)*N)$

Table 2: A comparison of performance of reliable broadcast protocols in a point-to-point network

among the centralized, two-phase, optimistic, fast message protocols and two cases of our counter-based protocol (with an arrival rate A = 10 and 20) under different values of N in a broadcast network, where N $\leq$ 50. From this figure, we show that the counter-based protocol requires the shortest delay among those protocols. Moreover, as N is increased, D is decreased in the optimistic protocol. Furthermore, the fast protocol, the two-phase protocol and the centralized protocol have a constant value delay.

Moreover, when errors occur, Num in Chang et al.'s protocol [6] and Birman et al.'s protocol [4] will be larger than that in our protocol. Although all three protocols send out the same number of request-for-retransmission messages, but the number of positive acknowledgements needed in these two protocols [4, 6] is larger than the number of the token messages needed in our protocol. Furthermore, when errors occur, Chung et al.'s protocol [10] may need up to (N-2) rollback request messages per lost message and the whole system must discard a lot of previous received broadcast message (i.e., rollbacks to the previous committed state). Although no explicit request-for-retransmission message is used in Amir et al.'s protocol [1], their protocol takes a long time delay (i.e., a cycle time) for the message sender to know that message retransmission is required. Moreover, since only the node holding the token can send one broadcast message in [1], their protocol has a low system throughput as compared to our protocol.

Table 2 shows a comparison of performance in a point-to-point network. Chang et al.'s centralized protocol [6] requires one message from the source node to the token node and (N-1) messages from the token node to the remaining nodes, for a total of N messages. In Birman et al.'s two-phase protocol [4], the source node sends a request message to each of

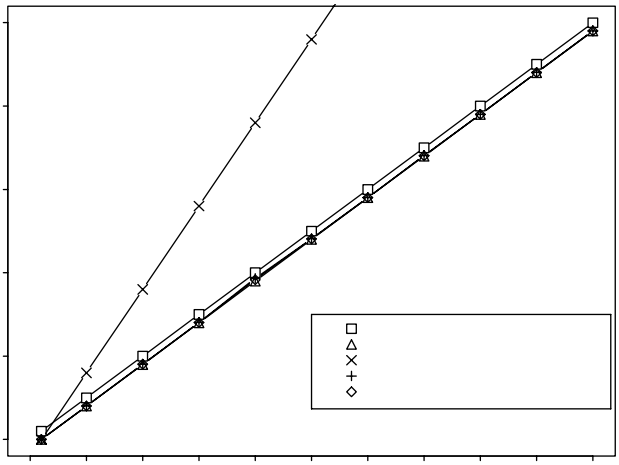


Figure 7: A comparison of Num in a point-to-point network

(N-1) nodes, then these (N-1) nodes send their local priority to the source node. Finally, the source node sends a message including the highest priority to those (N-1) nodes, for a total of 3*(N-1) messages. Chung et al.'s optimistic protocol [10] requires (N-1)+(N-1)/N messages to commit a message averagely. Amir et al.'s protocol [1] requires 2*(N-1) messages to commit a message averagely. The delay D of these protocols in a point-to-point network is almost the same as that in a broadcast network. Figure 7 shows a comparison of Num among the centralized, optimistic, fast message protocols and two cases of our counter-based protocol (with an arrival rate A = 10 and 20) under different values of N in a point-to-point network, where $N \leq 50$. (Note that since Num in the two-phase protocol is much larger than that in all other protocols, we do not include it in this comparison.) From this figure, we show that the counter-based protocol and the optimistic protocol require the fewest number of Num among those protocols. However, From Table 3, we show that, in fact, our counter-based protocol still requires fewer number of Num than the optimistic protocol.

Table 3: A comparison of Num between the counter-based protocol and the optimistic protocol in a point-to-point network

5 Reliability

In this section, we discuss the level of reliability that our protocol can achieve and compare it with those four algorithms [1, 4, 6, 10]. According to [13], there is a spectrum of reliability. On the low end of the spectrum, there is no guarantee made on the reliability of ordered deliveries, which is called $R0$. In the next level, it ensures that messages are delivered in a consistent order at operational nodes, which is called $R1$. That is, nodes that have failed and then came back up do not need to recover missed messages. Moreover, their message history before failure becomes irrelevant. This implies that there can be inconsistent deliveries of messages, but always involving failed nodes. Thus, there are never inconsistencies among operational nodes. Next, stronger than consistent delivery at operational nodes is forcing nodes that recover to roll back and re-deliver messages that they have delivered inconsistent, which is called $R2$. In this case, failed nodes may temporarily deliver messages in an incorrect order. However, after a failure, they must compensate for this and must also recover any missed messages. Strongest on the scale is a strict ordering guarantee that never allows inconsistent delivery, which is called $R3$.

Moreover, according to [1], reliable ordered delivery services can be categorized into

Figure 8: Five levels of reliable ordered delivery services

the following five levels of service: (1) basic, (2) FIFO, (3) casual, (4) agreed, and (5) safe. The *basic* level of service guarantees that messages are delivered to the application (without regard for ordering). Every node that receives a basic message can deliver that message immediately. The $FIFO$ level of service ensures that messages originated by a given node are delivered in the order in which they were originated. However, messages from different nodes can be arbitrarily interleaved. The *casual* level of service, which is taken from Lamport [17], is the reflexive transitive closure of the relation. Two properties must be satisfied: (a) Message m precedes message m' if node p delivers m before p sends m', and (b) Message m precedes message m' if node p sends m before p sends m'.

A message m is delivered by a node p in an *agreed* order in configuration C if and only if (a) p is a member of C, p has received m, and m was originated by a member of C or of a configuration that precedes C, (b) p does not deliver two different messages at the same position in an agreed order in C nor does it delivers one message at two different positions in an agreed order in C, (c) p has delivered all messages that precede m in an causal order in C, (d) p has delivered all messages that precede m in an agreed order in C, and (e) for any other node q in C, if p delivers m before n in an agreed order in C, then q does not deliver n before m in an agreed order in C. A message m is delivered by a node p in a $safe$ order in configuration C if and only if it is delivered by p in an agreed order in C and if p knows that all nodes in C have received and will deliver m.

The relationship among the five levels can be viewed in Figure 8. A comparison of levels of reliability for our protocol and these four protocols [1, 4, 6, 10] is shown in Table 4.

approach	pessimistic		optimistic		
algorithm	centralized [6]	two-phase [4]	optimistic [10]	fast message [1]	counter-based
[13]	R3	R3	R2	R3	R3
[1]	agreed	agreed	safe	safe	safe

Table 4: A comparison of levels of reliability for broadcast protocols

6 Conclusion

While the pessimistic approach to reliable broadcast protocols relies on the use of explicit acknowledgements from each receiving node to ensure the reliability of messages delivery, which results in a large number of control messages, the optimistic approach makes use of an implicit batch acknowledgement included in a token to discard the use of explicit acknowledgements. In this paper, based on the optimistic approach, we have proposed a counter-based reliable broadcast protocol. By appropriately updating the counter information recorded in the *system state* included in the token message, instead of using explicit acknowledgement messages, the proposed protocol needs fewer control messages and shorter delay to commit a broadcast message than other protocols no matter the rate of transmission errors is high or low. From our performance analysis, in the normal case (i.e., the error rate = 0, as the token-owner time (or the number of nodes, or the arrival rate of the events of broadcasting messages per second) is increased, the average control messages per broadcast message is decreased. When the error rate (E) is considered, as E (or the number of nodes) is increased, the average control messages per second is increased. The reason is that when E is considered, the number of request-for-retransmission messages will dominate the performance. However, the average control messages to commit a broadcast message in our protocol when errors occur is still less than all other protocols (as explained in Section 4.3).. The time delay to commit a message is the same no matter how the token-owner time is changed. As the number of nodes is increased, the time delay to commit a broadcast message is decreased. Moreover, from the simulation results, our protocol needs only no more than 0.1 control message per broadcast message in the normal condition. Furthermore, the level of reliability which the counter-based protocol can

Figure 9: The data structure of each element of the system state in the multicast protocol

achieve is no lower than all other protocols. We also have presented how to handle the flow control problem and discussed the token update technique.

Multicast communication is the delivery of copies of messages to a multicast group which is a collection of processes that are the destinations of the same sequence of messages [4, 5, 6, 12, 13, 14, 16]. We can extend the counter-based approach to ordered and reliable multicast communication for distributed systems. This can done by including an IDs field in the system state as shown in Figure 9, where the IDs field records the identifiers of the destination nodes for the message specified in the message index field. A token owner x can commit a message when the value of node x's counter field for the message sent out by node x equal to the expected number of the related IDs field. The extended protocol for multicast communication does not have the overhead in setting up logical patterns (for example, a tree) which are used in most of the other multicast protocols and can handle the case of the change of group membership easily [11, 13, 14].

References

[1] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella, "The Totem Single-Ring Ordering and Membership Protocol," ACM Trans. on Computer Systems, Vol. 13, No. 4, pp. 311-342, Nov. 1995.

[2] Amr El Abbadi, and Sam Toueg, "Maintaining Availability in Partitioned Replicated Databases," *ACM Trans. on Database Systems*, Vol. 14, No. 2, pp. 264-290, June 1989.

[3] Philip A. Bernstein, and Nathan Goodman, "Multiversion Concurrency Control-Theory and Algorithms," *ACM Trans. on Database Systems*, Vol. 8, No. 4, pp. 465-483, December 1983.

[4] Kenneth P. Birman, and Thomas A. Joseph, "Reliable Communication in the Presence of Failures," *ACM Trans. on Computer Systems*, Vol. 5, No. 1, pp. 47-76, February 1987.

[5] Kenneth Birman, Andre Schiper, and Pat Stephenson, "Lightweight Causal and Atomic Group Multicast," *ACM Trans. on Computer Systems*, Vol. 9, No. 3, pp. 272-314, August 1991.

[6] Jo-Mei Chang, and N. F. Maxemchuk, "Reliable Broadcast Protocols," *ACM Trans. on Computer Systems*, Vol. 2, No. 3, pp. 251-273, August 1984.

[7] J. Chang, "Simplifying Distributed Database Systems Design by Using a Broadcast Network," *Proc. of ACM SIGMOD*, Vol. 14, No. 2, pp. 223-233, 1984.

[8] Bo-Shoe Chen, and Chia-Lin Tan, "A Distributed Fail-Safe Broadcast Routing Protocol," *Proceeding of 1992 International Computer Symposium,*, pp. 241-248, 1992.

[9] Yuan-Chieh Chow, Kenneth C.K. Luo, and Richard Newman-Wolfe, "An Efficient Broadcast Protocol in Networks with Changing Topologies," *Proc. of IEEE International Conf. on Distributed Computing Systems*, pp. 88-93, 1990.

[10] Jen-Yao Chung, Jane W. S. Liu and Kwei-Jay Lin, "Optimistic Token-Driven Reliable Sequenced Broadcast Protocols," *Proc. of the 19th International Conference on Parallel Processing,* Vol. 3, pp. 303-310, 1990.

[11] Hector Garcia-Molina, and Boris Kogan, "An Implementation of Reliable Broadcast Using an Unreliable Multicast Facility," *Proc. of IEEE International Conf. on Distributed Computing Systems*, pp. 101-111, 1988.

[12] Hector Garcia-Molina, and Annemarie Spauster, "Message Ordering in a Multicast Environment," *Proc. of IEEE International Conf. on Distributed Computing Systems*, pp. 354-361, 1989.

[13] Hector Garcia-Molina, and Annemarie Spauster, "Ordered and Reliable Multicast Communication," *ACM Trans. on Computer Systems*, Vol. 9, No. 3, pp. 242-271, August 1991.

[14] Xiaohua Jia, and Shirley Y. SO, "A Multicast Mechanism with Ordering on Overlapping Groups," *Proc. of IEEE International Conf. on Network Protocols*, pp. 242-249, 1993.

[15] Thomas A. Joseph, and Kenneth P. Birman, "Low Cost Management of Replicated Data in Fault-Tolerant Distributed Systems," *ACM Trans. on Computer Systems*, Vol. 4, No. 1, pp. 54-70, February 1986.

[16] M. Frans Kaashoek, Andrew S. Tanenbaum, Susan Flynn Hummel and Henri E. Bal, "An Efficient Reliable Broadcast Protocol," *Operating Systems Review*, Vol. 23, No. 4, pp. 5-19, 1989.

[17] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. of the ACM*, Vol. 21, No. 7, pp. 558-565, July 1978.

[18] Luping Liang, Samuel T. Chanson, and agaerald W. Neufeld, "Process Groups and Groups Comm.: Classifications and Requirements," *IEEE The Computer Magazine*, Vol. 23, No. 2, pp. 56-66, 1990.

[19] Jane W.S. Liu, and Satoshi Hasegawa, "A Reliable Token-Driven Process Synchronization Algorithm," *Proc. of IEEE International Conf. on Distributed Computing Systems*, pp. 598-604, 1986.

[20] Shyh-Wei Luan, and Virgri D. Gligor, "A Fault-Tolerant Protocol for Atomic Broadcast," *Proc. of Annual IEEE Symposium on Reliable Distributed Systems*, pp. 112-126, 1988.

[21] Shyh-Wei Luan, and Virgil D. Gligor, "A Fault-Tolerant Protocol for Atomic Broadcast," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 1, No. 3, pp. 271-285, July 1990.

[22] P. M. Melliar-Smith, and L. E. Moser, "Fault-Tolerant Distributed Systems Based on Broadcast Communication," *Proc. of IEEE International Conf. on Distributed Computing Systems*, pp. 129-135, 1989.

[23] P. M. Melliar-Smith, Louise E. Moser, and Vivek Agrawala, "Broadcast Protocols for Distributed Systems," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 1, No. 1, pp. 17-25, January 1990.

[24] Philip M. Merlin and Adrian Segall, "A Failsafe Distributed Routing Protocol," *IEEE Trans. on Comm.,* Vol., COM-27, No. 9, pp. 1280-1287, Sept. 1979.

[25] Louise E. Moser, P. M. Melliar-Smith, and Vivek Agrawala, "Membership Algorithms for Asynchronous Distributed Systems," *Proc. of IEEE International Conf. on Distributed Computing Systems*, pp. 480-488, 1991.

[26] S.Navaratnam, S.Chanson, and G.Neufeld, "Reliable Group Communication in Distributed Systems," *Proc. of IEEE International Conf. on Distributed Computing Systems*, pp. 439-446, 1988.

[27] Adrian Segall and Baruch Awerbush, "A Reliable Broadcast Protocol," *IEEE Trans. on Comm.,* Vol. Com-31, No. 7, pp. 896-901, July 1983.