A Comprehensive Study of Reliable Multicast Protocols over Ethernet–Connected Networks

Ryan G. Lane
Department of Computer Science
Florida State University
Tallahassee, FL 32306
ryanlane@cs.fsu.edu

A project submitted to the

Department of Computer Science

in partial fulfillment of the

requirements for the degree of

Master of Science

November 27, 2000

Abstract

Recent advances in multicasting over the Internet present new opportunities for improving communication performance for clusters of workstations. Realizing collective communication over reliable multicast can achieve higher performance than over reliable unicast primitives. The standard IP multicast, however, only supports unreliable multicast, which is difficult to use for building high level message passing routines. Thus, reliable multicast primitives must be implemented over the standard IP multicast to facilitate the use of multicast for high performance communication on clusters of workstations. Although many reliable multicast protocols have been proposed for the wide area Internet environment, the impact of special features of local area networks (LANs) on the performance of the reliable multicast protocols has not been thoroughly studied. Efficient reliable multicast protocols for LANs must exploit these features to achieve the best performance. In this paper, we present our implementation of four types of reliable multicast protocols: the ACK–based protocols, the NAK–based protocols with polling, the ring–based protocols, and the tree–based protocols. The protocols are implemented over IP multicast using the standard UDP interface. We evaluate the performance of the protocols over Ethernet–connected networks, study the impact of some special features of the Ethernet on the performance of the protocols, and investigate the methods to exploit these features to improve communication performance.

1 Introduction

As microprocessors become more and more powerful, clusters of workstations have become one of the most common high performance computing environments. Many institutions have Ethernet–connected clusters of workstations that can be used to perform high performance computing. One of the key building blocks for such systems is the message passing library. Standard message passing libraries, including MPI [15] and PVM [20], have been implemented for such systems. Current implementations, such as MPICH[8] and LAM/MPI[23], focus on providing the functionality, that is, moving data across processors, and addressing the portability issues. To achieve interoperability, these implementations are built over point-to-point communication primitives supported by the TCP/IP protocol suite. However, studies have shown that the current implementations of message passing libraries for networks of workstations are not tailored to achieve high communication performance over clusters of workstations [2].

Recent advances in multicasting over the Internet present new opportunities for improving communication performance for clusters of workstations without sacrificing the functionality and the portability of the message passing libraries. Specifically, coupling local area network (LAN) hardware, which supports broadcast communication, with IP multicast [6], unreliable multicast communication is now supported by the TCP/IP protocol suite and is currently available in many systems. For LANs, although there is no mature reliable multicast software package, an efficient reliable multicast communication layer can be built over the unreliable IP multicast. Using reliable multicast primitives to realize collective communication routines can greatly improve the communication performance since multicast reduces both the message traffic over the networks and the CPU requirements at the end hosts.

While many reliable multicast protocols have been proposed [1, 3, 4, 7, 9, 10, 14, 16, 24, 26, 27], mostly for multicasting in the general wide area Internet environment, the impacts of special features of LANs on the performance of the protocols have not been thoroughly studied. The features may affect, among others, the flow control, buffer management and error control mechanisms in the reliable multicast protocols. In addition, these existing reliable multicast protocols generally assume their applications to be Internet applications, such as video conferencing, which have different traffic patterns than parallel programs. Thus, the existing protocols may not be efficient for high performance computing clusters. To develop efficient reliable multicast protocols for such systems, the implication of running parallel applications over LAN clusters must be examined and the special features of LANs must be taken into consideration.

This paper focuses on studying the impacts of LAN features on the reliable multicast protocols. Four types of reliable multicast protocols are used in the study, the *ACK–based protocols*, the *NAK–based protocols* with polling, the *ring–based protocols*, and the *tree–based protocols*. The ACK–based protocols are an extension of the reliable unicast protocols. In the ACK–based protocols, each packet sent by the sender is acknowledged by each receiver to ensure reliability. As a result, the ACK–based protocols suffer from the *ACK implosion* problem since the sender must process all acknowledgments for each packet sent. In the NAK–based protocols, the receivers send non-acknowledgments (NAKs) only when a retransmission is required. By reducing the number of transmissions from the receivers, the ACK implosion problem can be overcome. However, the NAK–based protocols require large buffers and must use additional techniques, such as polling [21], to guarantee reliability. The ring–based protocols combine the features of the ACK–based and NAK–based protocols. In the ring–based protocols, a designated site is responsible for acknowledging packets to the source. Receivers send NAKs to the sender when a retransmission is required. These protocols also require large buffers. In the tree–

based protocols, receivers are grouped into sets with each set containing a leader. The group members send acknowledgments to the group leader, while each group leader summarizes the acknowledgments within its group and sends the summary to the sender. Hence, the tree–based protocols overcome the ACK–implosion problem. Furthermore, by imposing a logical structure (group) on the receivers, the tree–based protocols control the number of simultaneous transmissions at the protocol level. Limiting the number of simultaneous transmissions can be significant in LANs since the media access protocols in LANs may or may not be able to efficiently handle many simultaneous transmissions.

In this paper, we present our implementation of the protocols, evaluate the performance of the protocols over Ethernet–connected networks, study the impact of special features of the Ethernet on these protocols, and investigate the methods to achieve the best multicast performance on Ethernet using these protocols. The protocols are implemented using the standard UDP interface over IP multicast. The main contributions of this paper are the following. First, we study the impacts of Ethernet features on each of the four types of protocols and identify the conditions to achieve the best performance using each of the protocols in the Ethernet-connected network environment. Second, we compare the four types of protocols and determine the most efficient multicast schemes for different situations.

The rest of the paper is structured as follows. Section 2 describes the related work. Section 3 discusses the differences between multicasting in the general wide area Internet environment and multicasting on LANs for parallel applications. This section then presents the four reliable multicasting protocols used in the study. Section 4 details the implementation of the protocols. Section 5 reports the performance study and Section 6 concludes the paper.

2 Related work

Extensive research has been conducted in the area of reliable multicast over the Internet [1, 3, 4, 7, 9, 10, 14, 16, 24, 26, 27]. A summary of recent development of reliable multicast protocols can be found in [13]. These protocols in general focus on a number of issues including group management, flow control and the scalability issues. The protocols can be classified into four types, the ACK–based protocols [24, 16] where all receivers send positive acknowledgments for each packet that they receive, the NAK–based protocols [4, 7, 9, 10, 27] where the receivers monitor the sequence of packets they receive and send negative acknowledgments on detection of a gap in the sequence number, the ring-based protocols [3, 26] where the receivers are organized as a logical ring and the receivers take turns to acknowledge the packets received to ensure reliability, and the tree–based protocols [16, 27] where the receivers are organized as subgroups to release the sender from processing all control messages from all receivers. Each has its own advantages and disadvantages. ACK–based protocols are extensions of traditional reliable unicast protocols. They suffer from the ACK–implosion problem and are, in general, not scalable. NAK–based protocols overcome the ACK–implosion problem. However, such protocols require large buffers and need other techniques, such as polling [21], to ensure reliability. Ring–based protocols combine the reliability of the ACK–based protocols and the efficiency of the NAK–based protocols. These protocols also need large buffers and may not be efficient for small messages. Tree–based protocols release the sender from the burden of processing acknowledgments from all of the receivers and are more scalable than other protocols. Such protocols may incur longer communication delays since the acknowledgment packets may pass multiple hops to reach the sender.

Our work does not intend to invent new reliable multicast protocols. Instead, we investigate the impact of architectural features of the most popular LAN, the Ethernet, on the performance of the reliable multicast protocols, study how these features affect flow control, buffer management and retransmission mechanisms in the reliable multicast protocols, and attempt to determine the most efficient methods to perform reliable multicast over the Ethernet. This paper presents a comprehensive study of all the major reliable multicast protocols proposed. Part of the study has been summarized in [12]. Notice that most of the existing protocols were proposed for wide-area networks and targeted Internet applications and, therefore, these protocols may not be efficient in the LAN environment for running parallel applications.

3 Reliable multicast over the Ethernet

In the general Internet environment, reliable multicast for general Internet applications is difficult for three main reasons. First, providing reliability requires feedback from each of the receivers, which may overwhelm the sender [5]. Second, maintaining the membership and enforcing the semantics of reliable multicast over dynamic multicast groups, where members can join and leave a group dynamically, is difficult. Third, implementing flow control that is acceptable to all receivers is complicated by the presence of heterogeneous receivers.

The situation is different when developing reliable multicast protocols in the LAN environment for parallel applications. First, communication patterns in parallel applications exhibit locality, that is, the communication pattern changes very slowly during the execution of a program [22]. In most cases, the communication patterns are predictable at compiler time [11]. This indicates that for most communication patterns in parallel programs, multicast groups are static, that is, group members do not join and leave a group dynamically. In this paper, we will ignore the membership management issue and assume that a multicast group, once created, will not change. Second, parallel applications usually communicate with small message sizes. Thus, to achieve high performance, the protocols must be able to handle small messages efficiently.

This paper considers homogeneous clusters, that is, all workstations have the same network connection speed and processing power. In homogeneous clusters, efficient flow control is possible. Multicasting on heterogeneous clusters may require different techniques and is beyond the scope of this paper. As discussed earlier, the main challenge in designing efficient reliable multicast protocols for high performance LAN clusters is to exploit special features of LANs. Next, we will discuss some LAN features that may affect the performance of reliable multicast.

- LAN hardware typically supports broadcast, which means that sending a packet to one receiver costs almost the same bandwidth as sending a packet to the whole group. This might affect the retransmission and acknowledgment strategy in multicast protocols. However, it must be noted that using multicast for retransmission and acknowledgment may introduce extra CPU overhead for unintended receivers (to process the unintended packets).

- The use of shared media for communication in LANs implies that all transmitters compete for the shared resource. The media access protocol used in LANs, such as the CSMA/CD protocol with binary exponential backoff in the Ethernet, may or may not resolve contentions effectively. When the media access protocol cannot resolve contentions effectively, it may be desirable for the reliable multicast protocol to impose a limit on the number of simultaneous transmissions at a given time.

- The sender and the receivers are close to each other in LANs and the communication propagation delay is small. This might affect the choice of flow control and buffer management schemes. For example, since the sender can receive the acknowledgments from the receivers quickly in such systems, large buffers may not be necessary to achieve good performance.

- In a wired LAN, such as the Ethernet-connected networks, the transmission error rate is very low. In such systems, packets are lost mainly due to the overflow of buffers at the end host. With a proper transmission pacing scheme, the retransmission mechanism may not be as critical as that in the general Internet environment.

This paper studies the impacts of these features on the reliable multicast protocols. Four types of protocols will be used in the study, the ACK–based protocols, the NAK–based protocols with polling, the ring–based protocols, and the tree–based protocols. Next, we will describe the four types of reliable multicast protocols. The implementation details will be presented in the next section.

ACK–based protocols

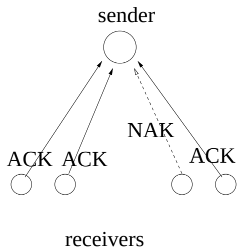


Figure 1: The ACK–based protocol

The ACK–based protocols [7, 16, 24] require the sender to maintain the state of all receivers to whom it has to send information and from whom it has to receive acknowledgments (ACKs). The sender uses multicast to transmit data to the receivers and the receivers unicast ACKs or non–acknowledgments (NAKs) to the sender. Note that like reliable unicast protocols, NAK packets may speedup the packet retransmission when transmission errors occur, but are not necessary for the correctness of the ACK–based protocols. The sender releases the buffer for a data packet only after positive acknowledgments from all receivers for that packet are received. Figure 1 illustrates the ACK–based protocols. Variations in transmission pacing and retransmission exist. For example, retransmission can be either sender-driven, where the retransmission timer is managed at the sender, or receiver-driven, where the retransmission timer is managed at the receivers [7]. The flow control can either be rate–based or window–based. The main limitation of the ACK–based protocols is that the sender must process all acknowledgment packets from all receivers. This ACK implosion problem limits the scalability of the ACK–based protocols.

NAK–based protocols

The NAK–based protocols [4, 7, 9, 10, 27] avoid the ACK implosion problem as receivers only send non-acknowledgments (NAKs) to the sender when a retransmission is necessary. A retransmission is required when there is a transmission error, a skip in the sequence numbers of the packets received, or a timeout. No acknowledgments are used. Since the sender only receives feedback from the receivers when packets are lost, and not when they are delivered, the sender is unable to ascertain when data can safely be released from memory. In order to ensure reliability, an infinite buffer space would be

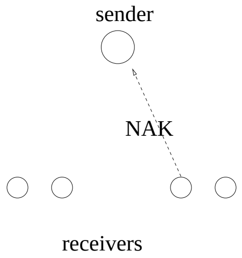


Figure 2: The NAK–based protocol

required. Thus, in practice when the buffer space is limited, some mechanism, such as polling [21], must be incorporated into the NAK–based protocols to guarantee reliability. Polling requires the sender to periodically flag packets to be acknowledged by the receivers, thereby allowing the sender to determine when data can safely be released from memory, without having to process ACKs for every packet.

NAK-based protocols can potentially suffer from the NAK implosion problem if many receivers detect errors. To remedy this problem, a NAK suppression scheme can be developed [17]. In the NAK suppression scheme, when a receiver detects an error, it waits for a random period of time and then multicasts a NAK to the sender and other receivers. When a receiver obtains a NAK for a packet which it intends to send a NAK for, the receiver behaves as if it had sent the NAK. The advantage that NAK–based protocols have over ACK–based protocols is that the sender does not process ACKs from the receivers. Since transmission errors rarely occur, the number of NAKs the sender must process is significantly less than the number of ACKs. Thus, the NAK–based protocols provide more scalability. The limitation, however, is that other mechanisms must be incorporated into a NAK–based protocol to offer reliable multicast service. Figure 2 illustrates the NAK–based protocols.

In our implementation, the NAK-based protocols with polling utilize a different NAK suppression scheme than the one in [17]. Our NAK suppression scheme is implemented at the sender side. A retransmission will happen only after a designated period of time has passed since the previous transmission. Thus, the receivers may send multiple NAKs to the sender while the sender performs retransmission only once.

Ring-based protocols

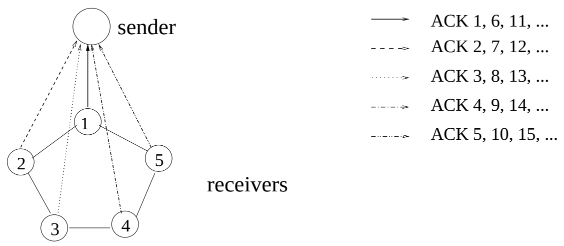


Figure 3: A typical ring structure

The ring-based protocols [3, 26] are a combination of the NAK–based protocols and the ACK–based protocols. These protocols aim at combining the throughput advantage of the NAK–based protocols with the reliability of the ACK–based protocols. Figure 3 shows a ring–based protocol. In ring–based protocols, among all receivers, only one token site is responsible for sending acknowledgment packets to the source. The source maintains timers and retransmits packets if an ACK is not received from the token site within the timeout period. Receivers send NAKs to the token site for lost packets which were originally multicast from the source. The ACK sent to the source (and multicast to the other receivers) also serves as a token passing mechanism. Each receiver will serve as the token site in a round-robin fashion. The rotating token passing scheme enables the source to know the status of each of the receivers once all receivers send ACKs back. For example, assuming that there are N receivers organized by node numbers 0, 1, ..., $N-1$, using the ring–based scheme, receiver 0 will ACK packet 0, receiver 1 will ACK packet 1 (and, implicitly, packet 0), and so on. After the sender sends packet N and receives the acknowledgment for packet N from receiver 0, it can safely release packet 0 from memory (if it has also received acknowledgments for all packets up to packet N) since an ACK from receiver 0 for packet N implies that receiver 0 has received all packets 0-N. The receipt of ACKs from the other receivers would also imply the receipt of packet 0 by all receivers. Note that a receiver will only send the ACK it is responsible for if it has received that packet and all packets in order up to that packet. Therefore when receiver 1 ACKs packet 1, it is also implicitly acknowledging receipt of packet 0. This scheme also implies that the number of buffers must be greater than the number of receivers since receipt of an ACK for packet X (and all packets preceding it) only releases packet $X-N$ from memory. Hence, there will always be at least N packets buffered in case a retransmission is required.

Three modifications were made to this scheme to adapt the protocol to the LAN environment. First, the receivers unicast the ACK packets to the sender, rather than using multicast. This modification reduces the computational load at the receiving nodes and decreases the number of out-of-order packets that are received. Second, since the last N packets can not be acknowledged by an ACK N greater, the last packet of the message is acknowledged by all receivers. Third, NAKs are sent directly to the source and not to a token site. This modification has a minimal effect on the source since retransmissions in Ethernet-connected LANs are rare due to the low transmission error rate.

Tree-based protocols

The tree–based protocols were designed to solve the scalability problem. The tree–based protocols are essentially ACK–based protocols in the sense that the sender releases the buffer for a packet only after it is certain that all receivers have received the packet. The difference is that the sender only processes the aggregated acknowledgments instead of individual acknowledgments from each receiver as in the ACK–based protocols.

The tree–based protocols are characterized by dividing the receiver set into groups with each group having a group leader. In the existing tree–based protocols [16, 27], which were designed for the Internet environment, the sender only interacts with the group leaders while the group leaders interact with their group members. In these protocols, the group leaders not only aggregate the acknowledgments in the group, but also are responsible for packet retransmissions within their groups. We adapt tree–based protocols for the LAN environment for two reasons. First, tree–based protocols overcome the ACK implosion problem and reduce the computational requirements at the sender in comparison to the ACK–based protocols. Second, tree–based protocols can control the number of simultaneous transmissions at the protocol level, which may be essential to achieve good communication performance in the LAN environment.

We made two modifications to the existing tree–based protocols to adapt such protocols for the LAN environment. First, the group leaders in our protocol only aggregate the acknowledgments within their groups, but do not perform packet retransmission. The sender is responsible for all data packet transmissions and retransmissions. Since in the LAN environment, all nodes are close to each other, one node is sufficient for all retransmissions. This modification does not increase the computational load at the sender since the transmission error rate is very low in Ethernet–connected LANs and retransmission

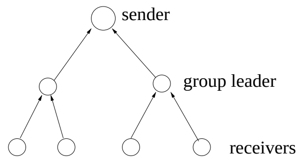


Figure 4: A binary tree structure

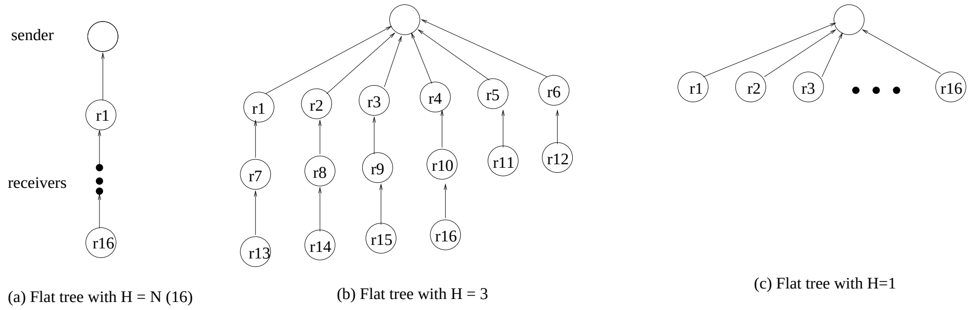


Figure 5: The flat tree structure

is rarely needed. This modification greatly reduces the implementation complexity. Second, we use a logical structure called a *flat tree* to organize the receivers. The existing tree–based protocols impose a logical tree that "grows". One example is the binary tree structure as shown in Figure 4. Such a logical structure is not effective in controlling the number of simultaneous transmissions. For example, using the binary tree structure, the maximum number of simultaneous transmissions is roughly $\frac{N}{2}$ with the tree height equal to $lg(N)$, where N is the number of receivers. The flat tree structure controls the simultaneous transmissions more effectively. The logical structure of a flat tree is determined by two parameters, the number of receivers, N, and tree height, H. Examples of flat trees of different heights for $N = 16$ are shown in Figure 5. In a flat tree, a node will wait for the acknowledgment for a packet from its successor before it sends the acknowledgment to its predecessor. Thus, each subtree can at most have one active transmission at a given time and the maximum number of simultaneous transmissions at a given time is $\frac{N}{H}$. Notice that the ACK–based protocol is a special case of the tree–based protocols, a flat tree with $H = 1$.

Comparison of the four protocols

Each type of protocol has its own advantages and limitations. ACK–based protocols provide reliability, but suffer from the ACK implosion problem. NAK–based protocols alleviate the ACK implosion problem but require an infinite buffer size or other mechanisms to ensure reliability. Ring–based protocols are efficient, but require a large buffer size when the number of receivers is large and are complicated to implement. Tree-based protocols are scalable but more complicated compared to the other protocols. Different types of protocols may have different requirements for system resources. Table 1 shows a summary of the resource requirements of the protocols.

4 Implementation issues

We implement the four protocols over IP multicast using the UDP interface in the Linux operating system. The protocols are executed as user processes. All protocols use a window based flow con-

Table 1: Resource requirements of the protocols

	bandwidth requirement	memory requirement	CPU requirement	implementation complexity
ACK-based	high	low	high	low
NAK-based	low	high	low	low
Ring-based	low	high	low	high
Tree-based	high	low	low	high

trol scheme and maintain a timer at the sender side to implement the sender–driven error control mechanism. Some implementation issues are discussed next.

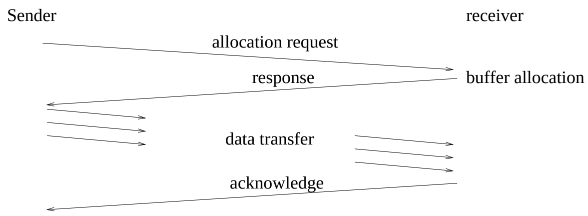


Figure 6: Buffer management

- **Buffer management**. The implementation assumes limited buffer space on the receiver side. The buffer allocation is done by sending the size of the message to the receivers first before the actual message is transmitted so that the receiver can allocate the necessary buffer to receive the message. The messaging protocol is depicted in Figure 6. At least two round–trips of messaging are necessary for each data transmission.

- **Packet Header**. The reliable multicasting protocol is implemented over the UDP interface. The sender/receiver identity information in the UDP and IP header, such as the machine IP address and the port number, are used as the identity information for the reliable multicast protocols. In addition to the identity information, the packet header has another two fields, a one-byte *packet type* and a four-byte *sequence number* which is used to implement the window based flow control mechanism. There are three types of packets used in the protocol, the data packet, the ACK packet and the NAK packet.

- **Timer management**. A timer must be maintained in the protocols. Since we implement the protocol in the user domain, the system call *gettimeofday()* is called everytime when we need to know the time. The time precision of this system call is in the order of microseconds (10^{-6} second), which is more than sufficient for the protocol. However, since the system call is an expensive operation, we only maintain the approximate time by making the system call only at (1) the beginning of the main iteration (to check whether a timeout has occurred), and (2) every time a packet is sent (to set the retransmission timer).

- **Flow control**. The reliable multicast protocols use a window based flow control scheme. The sender and the receivers maintain a window of buffers. The Go–back-N sliding window protocol [25] is implemented, which requires the sender to retransmit all packets once an error occurs.

Theoretically, this protocol may perform worse than the selective–repeat sliding window protocol [25], which only requires the retransmission of the packets that incur transmission errors. In practice, transmission errors almost never occur in the Ethernet connected networks and thus, the Go–Back–N protocol performs as well as the selective–repeat protocol. Since the logic in the Go–Back–N protocol is much simpler than that in the selective–repeat protocol, the implementation of the Go–back–N protocol is more efficient.

- **Error control**. Sender–driven error control is implemented in all the protocols. A timer is associated with each packet in the sender window. If the sender does not receive acknowledgments (which may be aggregated) from all receivers, the timer goes off and the packet is retransmitted by multicasting the packet to all receivers. The retransmission may also be triggered when a NAK packet is received. A retransmission suppression scheme is implemented to avoid retransmitting too eagerly. A retransmission will happen only after a designated period of time has passed since the previous transmission. The sender may process a number of NAK packets and perform retransmission only once. It must be noted that the efficiency of error recovery schemes makes little difference in the communication performance in the wired LAN environment since errors almost never happen in such environment.

5 Performance

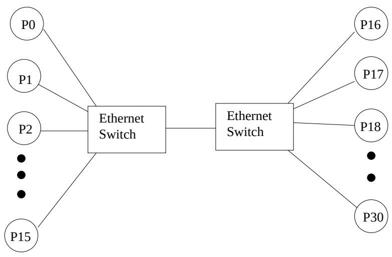


Figure 7: Network configuration

The performance of the protocols is evaluated in a cluster with 31 Pentium III–650MHz processors. Each machine has 128MB memory and 100Mbps Ethernet connection via a 3Com 3C905 PCI EtherLink Card. All machines run RedHat Linux version 6.2, with 2.2.16 kernel. The machines are connected by two 3Com SuperStack II baseline 10/100 Ethernet switches as shown in Figure 7. In the figure, $P0$ is used as the sender and all other machines are receivers. Communication in Ethernet can sometimes be quite random. To deal with this problem and obtain more accurate experimental results, we measure the communication time three times for each experiment and report the average of the three measurements.

Figure 8 compares the communication time for transferring a 426502 byte file to different numbers of receivers using TCP, the standard reliable unicast protocol, and using the ACK–based reliable multicast protocol. As can be seen in the figure, using TCP, the communication time grows almost linearly with the number of receivers, while using multicast, the communication time grows only slightly as the number of receivers increases. In this particular experiment, using multicast, transferring the file to 30 receivers only needs about 6% more time than transferring the file to one receiver (from 0.060 second to 0.064 second). Even though our protocol is implemented in the user domain and introduces other overheads, the protocol outperforms TCP when the number of receivers is larger than 1.

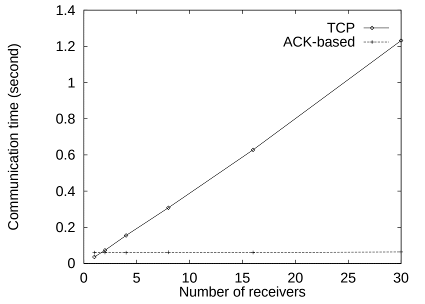


Figure 8: ACK–based protocol .vs. TCP

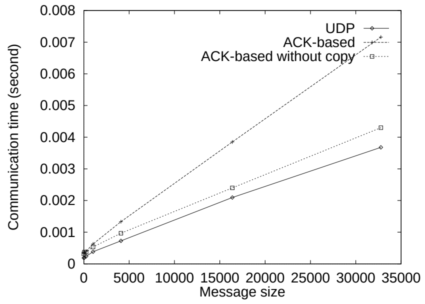


Figure 9: ACK–based protocol .vs. raw UDP

Figure 9 compares our protocol with raw UDP. The raw UDP performance is measured by using UDP with IP multicast to send all of the data and having the receivers reply upon receipt of the last packet. This comparison shows the protocol overhead we introduce in our implementation of the ACK–based protocol. As can be seen in the figure, our protocol introduces substantial overhead compared to raw UDP. When the message size is small, our protocol requires two round–trips for buffer management. When the message size is large, our protocol introduces an additional copy of the data from the user space to the protocol buffer, and this copy incurs most of the overhead when the message size is large. For comparison purposes, the figure also shows the protocol without the copy (which is an incorrect protocol).

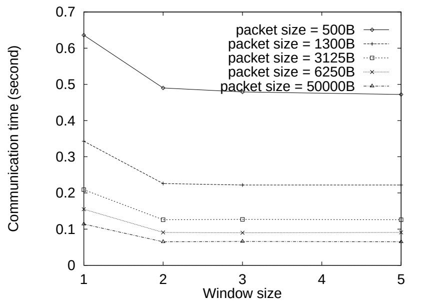
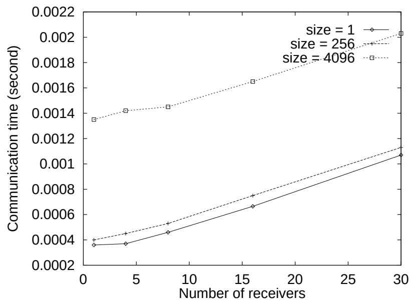
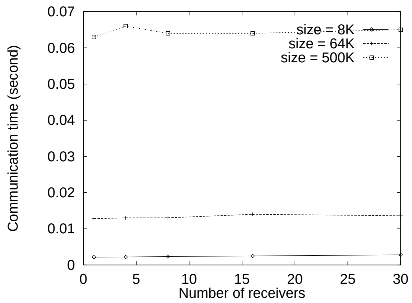


Figure 10: Performance of ACK–based protocols with different packet sizes and window sizes

Figure 10 shows the performance of the ACK–based protocols with different packet sizes and different window sizes. This experiment is done with 30 receivers receiving 500KB data from the sender. Notice that the total amount of buffer memory is equal to packet size times window size. As can be seen from



(a) Small message sizes



(b) Large message sizes

Figure 11: The scalability of ACK–based protocols

the figure, the performance of the ACK–based protocol is very sensitive to the packet size, the larger the packet size, the better the performance. Using UDP, the maximum packet size is 64K including the UDP and IP header. Two reasons contribute to this result. First, a large packet size reduces the number of acknowledgments that the receivers send, which reduces both the network load and the CPU processing in the sender. Second, the transmission error rate is very low in such systems, therefore, increasing the packet size does not really increase the chance of retransmission. Another interesting observation from this experiment is that for any given packet size, the best performance can be achieved with a window size of 2, which indicates that the ACK–based protocol is not sensitive to the amount of buffer space. The short propagation time in the networks does not give much opportunity for pipeline communication using the ACK–based protocol.

Figure 11 shows the scalability of the protocol. The experiment is done with a maximum packet size of $50KB$ and thus, when the message size is less than $50KB$, only one data packet is sent. When the message size is small ($< 4KB$), the protocol is not scalable as the communication time increases almost linearly with the number of receivers as shown in Figure 11 (a). This is because the total number of messages increases linearly as the number of receivers increases since each receiver will send acknowledgement packets. In order for the protocol to be scalable, the message size must be large enough so that the data transmission time dominates the time for acknowledgments. Figure 11 (b) shows this case when the message size is large ($> 8KB$). In this case, the acknowledgment overhead is negligible and the protocol is scalable.

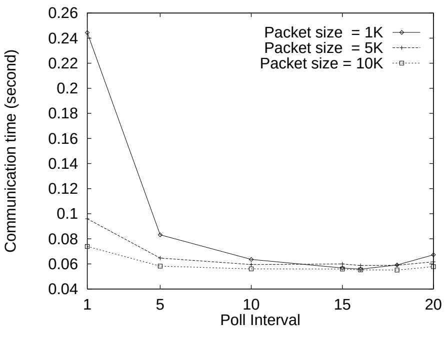


Figure 12: Performance of NAK–based protocols with polling at different poll intervals

The NAK–based protocols with polling provide better communication performance than the ACK–based protocols by avoiding the ACK implosion problem and the overhead of processing ACKs from all receivers for every packet. The cost of this advantage is the need for additional buffer space. Figure 12 shows the performance of the NAK–based protocols with polling at different poll intervals with different packet sizes. This experiment is done with 30 receivers receiving 500KB data from the sender and a window size of 20. Notice that the total amount of buffer memory is equal to the packet size times the window size of 20. The poll interval refers to the interval between packets that are flagged to be acknowledged by the receivers. As shown in the figure, the best performance is achieved when the poll interval is 16-18, or about 80-90% of the window size, regardless of the packet size. Although not shown here, other experiments with different window sizes demonstrate a similar trend. There are two reasons for this result. First, at a small poll interval, the protocol essentially becomes an ACK–based protocol and increases the network load and the CPU processing in the sender. As in the ACK–based protocols, this overhead is more pronounced at smaller packet sizes. Second, at poll intervals too close to the window size, the protocol again behaves like the ACK–based protocols. A large amount of data close to the buffer size is sent before any acknowledgments are received. The result is similar to using a very large packet size in the ACK–based protocols. The pipeline communication is affected due to the fact that the buffer becomes nearly full before ACKs are received to safely release the data from memory.

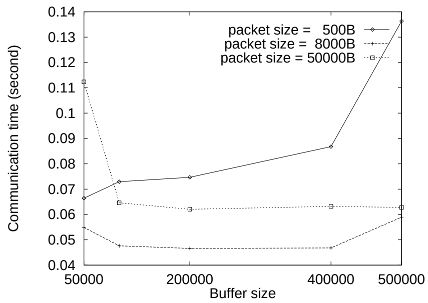


Figure 13: Performance of NAK–based protocols with polling for different buffer sizes

Figure 13 shows the performance of the NAK–based protocols with different packet sizes and different buffer sizes. This experiment is done with 30 receivers receiving 500KB data from the sender with the poll interval set to 80-85% of the window size. Notice that the window size can be determined by dividing the buffer size by the packet size. These results show that if the packet size is too small or too large the performance will suffer. Two reasons contribute to this result. If the packet size is too small, there is increased overhead due to the number of packets and ACKs that need to be transmitted. If the packet size is too large, the pipeline is affected by the increase in the amount of time required to copy the data from the user space to the protocol buffer. In general, unlike the ACK–based protocols, smaller packet sizes tend to result in better performance, as long as the packet size is not excessively small. This can be attributed to the fact that smaller packet sizes assist in creating pipeline communication. This figure also shows that when the window size is too small, the pipeline communication cannot be maintained efficiently and the communication performance suffers. A final observation is that the performance of the protocol is not strictly increasing or decreasing relative to the packet size. This would indicate that the performance depends on a number of factors, including the buffer size, the packet size, and the poll interval.

The scalability of the NAK–based protocols with polling is shown in Figure 14. The experiment is done with various packet sizes ranging from $500B$ to $50KB$ and a message size of 500KB. The window

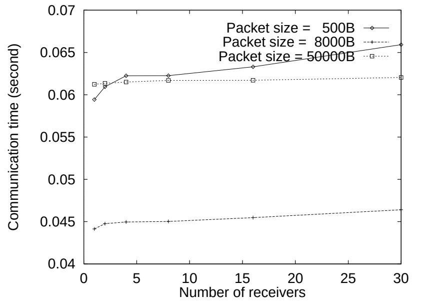


Figure 14: The scalability of NAK–based protocols with polling

size and the poll interval were set to provide the best performance based on the previous experiments. For example, when an $8KB$ packet size was used, the window size was set to 25 with a poll interval of 21. Overall the protocol is quite scalable with an average increase of only about 5.5% for sending to 30 receivers in comparison to sending to just one receiver. The larger the packet size, the more scalable the protocol becomes. As shown in the figure, the increase for a $500B$ packet size is about 11% (from 0.059 second to 0.066 second) while the increase for a $50KB$ packet size is only 1% (from 0.061 second to 0.062 second). The reason for this is that as the packet size increases, the number of packets that are sent decreases. As fewer packets are sent, the number of ACKs required also decreases. By having even fewer ACKs to process, the protocol becomes more scalable. In general, the NAK–based protocols with polling are scalable due to the fact that data transmission time is able to dominate the time required to process acknowledgments.

The ring–based protocols also limit the number of acknowledgments that the sender must process. By reducing the ACK processing overhead, the ring–based protocols seem to offer better performance than the ACK–based protocols, particularly for large message sizes. Another advantage is that by rotating the acknowledgments amongst the receivers, contention in the network is limited. The next part of the study will focus on the ring–based protocols.

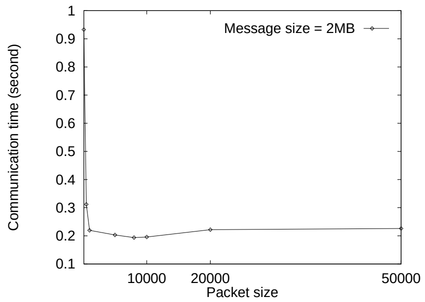


Figure 15: Performance of ring–based protocols with different packet sizes

Figure 15 demonstrates the performance of the ring–based protocols with different packet sizes. The experiment was conducted by sending a $2MB$ message to 30 receivers. The window size was set at 35. As can be seen from the figure, the best results were obtained when the packet size was between $5KB$ and $10KB$. Once again, this can be attributed to the fact that extra overhead is introduced at smaller packet sizes, while the pipeline communication is not efficient at larger packet sizes.

Figure 16 shows the performance of the protocol with different window sizes. The experiment was performed by sending $2MB$ of data to 30 receivers. As can be seen in the figure, the window size that

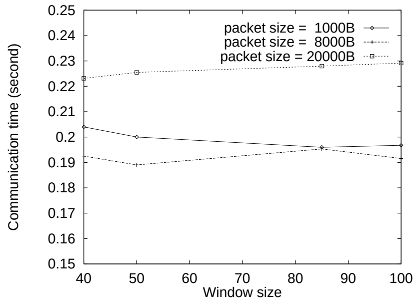


Figure 16: Performance of ring–based protocols with different window sizes

achieves the best performance is related to the size of the packets. In comparison to the other protocols, the ring–based protocols generally require more buffers in the window to provide better performance. The protocols require at least one more buffer than the number of receivers, however, the protocols can achieve better performance with an even larger buffer size. This results from the fact that the ring–based protocols allow more pipeline communication as the sender isn't required to process ACKs from every receiver for each packet.

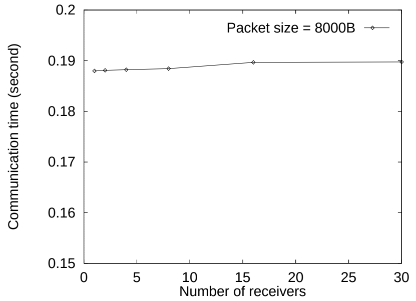


Figure 17: The scalability of ring–based protocols

The scalability of the ring–based protocols is shown in Figure 17. The results were obtained from transmitting $2MB$ of data with a window size of 50. However, at large message sizes, scalability is not really an issue as any protocol will appear scalable. As demonstrated in the figure, the difference between sending a large message to one receiver and 30 receivers is negligible. The communication time increases less than one percent, from 0.188 second to 0.190 second.

The tree–based protocols may offer better communication performance than the ACK–based protocols since (1) they reduce the processing at the sender and (2) they limit the contention in the networks at the protocol level. These advantages are obtained at the cost of maintaining a logical structure on the set of receivers, which can cost extra communication delay since communications are sequentialized at the protocol level. Whether the advantages can out–weigh the costs depends on many factors, including the number of receivers, the logical structure, the size of messages and the buffer sizes. Next, we will study the tree–based protocols and compare the tree–based protocols with the ACK–based protocols.

Figure 18 shows the performance of different trees when transferring 500KB data to 30 receivers with packet sizes of $50KB$ and $8KB$ and with sufficiently large window size (20). As can be seen from the figure, the two extreme cases ($Height = 1$ and $Height = 30$) do not produce the best communication performance for either packet size. Unlike the ACK–based protocols where a large packet size results in

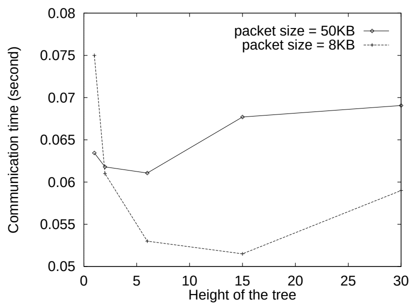


Figure 18: Performance of tree–based protocol for different logical structures

better communication performance, the performance of the flat tree depends on both the tree structure and the packet size. For example, when packet size = $8KB$, the tree structure with $Height$ = 15 results in the best performance. When packet size = $50KB$, the tree structure with $Height$ = 6 performs the best. In all cases except $Height$ = 1, using an 8KB packet size performs better than using a $50KB$ packet size. There are two reasons for this observation. First, small packets help in creating the pipeline communication, which can result in better communication performance. Second, using small packets requires more acknowledgments. However, this does not have a significant impact on the tree–based protocols since in these protocols, the sender only processes the aggregated acknowledgments. The accumulated effect is that using $8KB$ packets performs better than using $50KB$ packets for most cases.

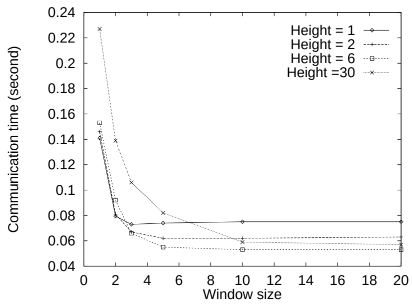


Figure 19: Performance of tree–based protocols for different window sizes

Figure 19 shows the relation between the logical tree structure and the memory requirement. This experiment transfers 500KB data to 30 receivers with a packet size of 8KB. Comparing Figure 19 and Figure 10, we can see that the tree–based protocol requires different window management than the ACK–based protocols. Essentially, when the tree grows higher, more buffers are necessary for the protocols to achieve the best performance. For example, when the tree height is 30, we need buffers for about 10 packets in the window for the protocol to achieve close to the best performance in comparison to buffers for just 2 packets in the ACK–based protocols. Another conclusion from this experiment is that the tree–based protocol can offer better communication performance than the simple ACK–based protocol when the message is large. As shown in the figure, the ACK–based protocol performs worse than other tree protocols with sufficient window size. Two factors contribute to the performance gain, first the tree–based protocol reduces the load on the sender which is the bottleneck in the multicast communication and second, tree–based protocols allow more pipeline communication as the round-trip delay becomes larger. While the second factor may allow the tree–based protocol to out–perform the

ACK–based protocol for large messages, it might hurt the performance when the message size is small. This is demonstrated in the next experiment.

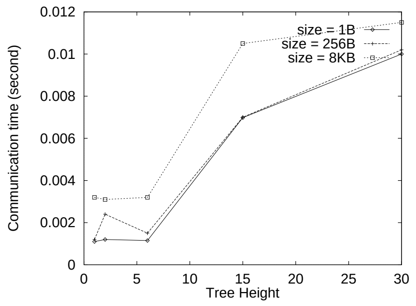


Figure 20: Performance of tree–based protocols for small messages

Figure 20 shows the performance of different trees when the message size is small. As shown in the figure, when the tree height is small (< 6), all the protocols have similar performance, however, when the tree height is larger (≥ 15) the delay for transferring the small message increases dramatically. This is mainly due to the relay of the messages at the user level. These results indicate that the tree-based protocols are not efficient for small messages compared to the ACK–based protocol.

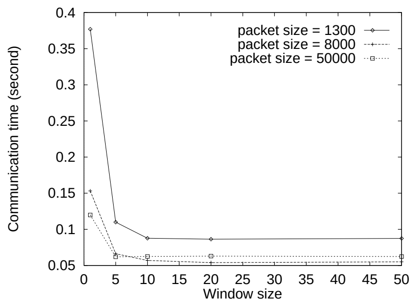


Figure 21: Performance with different window size and packet sizes

Figure 21 shows the performance of a tree with 30 receivers and $Height = 6$. As shown in the figure, the performance of the tree–based protocol not only depends on the packet size, but also the window size. Unlike the ACK–based protocol where a larger packet size always results in better performance, to achieve better performance, the packet size must be carefully selected. As shown in the figure, when the packet size is too large ($packet\_size = 50000$), the pipeline communication will be affected, while a small packet size ($packet\_size = 1300$) may result in excessive overhead.

Next, we will compare the performance of the four types of protocols for transferring different size messages. For small messages that can be transferred in one packet, the ACK–based protocols, the NAK–based protocols with polling, and the ring–based protocols have exactly the same behavior. All these protocols will perform better than the tree–based protocols as shown in Figure 20. This is because the tree–based protocols relay ACK packets at the user level while the other protocols do not.

For large message sizes, the performance of the protocols is ordered as follows:

NAK–based $\geq$ ring–based $\geq$ tree–based $\geq$ ACK–based

The results of transmitting a large message ($2MB$) are shown in Table 2. In this experiment, we try to compare the "best" protocol for each type of protocol. The data are obtained by probing the parameter space for each type of protocol and selecting the ones that can provide the best performance.

Table 2: Throughput achieved when sending 2MB of data

	Throughput
ACK-based	68.0Mbps
NAK-based	89.7Mbps
Ring-based	84.6Mbps
Tree-based ($Height = 6$)	77.3Mbps
Tree-based ($Height = 15$)	81.2Mbps

Specifically, the ACK–based protocol used a packet size of $50KB$ with a window size of 5; the NAK–based protocol with polling had a packet size of $8KB$, a window size of 50 (giving a buffer size of $400KB$), and a poll interval of 43; the ring–based protocol also had an $8KB$ packet size and a window size of 50; the tree–based protocols of different heights ($H = 6$ and $H = 15$) used a packet size of $8KB$ and a window size of 20. The NAK–based protocol with polling was able to achieve the best performance due to the fact that it does not require ACKs to be processed for every packet. Since retransmissions are rare in Ethernet-connected LANs, it was not necessary to process NAKs, either. Thus, the NAK based protocol could perform efficient pipeline communication without interruption. The ring–based protocol also achieved high performance due to the fact that it only had to process one ACK for each packet. This feature allows it to maintain pipeline communication and achieve better performance than the other two protocols. However, the NAK–based protocol still processed fewer ACKs and could attain a higher throughput. The tree–based protocol out–performed the ACK–based protocol for large messages. The factors that contributed to the performance gain include a reduced load of ACK processing at the sender, and more pipeline communication due to the longer round trip delay. The communication delay of relaying the messages at the user level prevented the protocol from achieving the throughput of the NAK–based with polling and ring–based protocols. The ACK–based protocol exhibited extremely poor performance due to the overhead of processing ACKs from every receiver for every packet. The results support the conclusion that the NAK–based protocols with polling are the most efficient protocol for large messages, although the ring–based protocols also show potential for high performance.

6 Conclusion

In this paper, we evaluated the performance of four types of reliable multicast protocols, the ACK–based protocols, the NAK–based protocols with polling, the ring–based protocols, and the tree–based protocols, over Ethernet–connected networks and studied the impact of some features of the Ethernet on the flow control, buffer management and error control schemes in the protocols. The main conclusions we obtained for reliable multicast over Ethernet connected networks are the following.

- **ACK–based protocols**. First, the traditional window–based flow control mechanism is not effective. In the studies, the ACK–based protocols only require buffers for two packets to achieve the best performance for any given packet size (and regardless of the packet size). A larger buffer size does not result in better performance. Second, the performance of ACK–based protocols is quite sensitive to the packet size. In general, the larger the packet size, the better the performance.

- **NAK–based protocols with polling** First, although the performance of the NAK–based protocols is also quite sensitive to the packet size, smaller packet sizes may perform better than larger packet sizes. Second, the window–based flow control mechanism is effective for the NAK–based protocols. A large buffer size is necessary for the NAK–based protocols to achieve good performance. Third, the poll interval plays an important role in the performance of the NAK–based protocols. The best performance can be achieved when the poll interval is about 80-90% of the window size, regardless of the packet size.

- **Ring–based protocols**. First, the performance of the ring–based protocols is also sensitive to the packet size. The packet size must be carefully selected as performance suffers if the packet size is too small or too large. Second, the window–based flow control mechanism is effective. The buffer size to obtain the best performance depends on the packet size.

- **Tree–based protocols**. First, the window–based flow control mechanism is effective for the tree–based protocols. A large window size is necessary for the tree–based protocols to achieve good performance. The window size depends on a number of factors including the packet size and the tree structure. Second, unlike the ACK–based protocols, the packet size that can achieve the best communication performance for the tree–based protocols depends on many factors including the logical tree structure.

- **Comparison of the protocols**. For small messages, the ACK–based, NAK–based with polling, and ring–based protocols have the same behavior and performance. All of these protocols perform better than the tree–based protocols. For large messages, the performance of the protocols is ordered as follows:

$$\text{NAK–based} \geq \text{ring–based} \geq \text{tree–based} \geq \text{ACK–based}$$

References

[1] M. P. Barcellos and P.D. Ezhilchelvan, "An End-to-End Reliable Multicast Protocol Using Polling for Scalability", *Proceedings of IEEE INFOCOM'98*, pages 1180–1187, March 1998.

[2] P.H. Carns, W.B. Ligon III, S. P. McMillan and R.B. Ross, "An Evaluation of Message Passing Implementations on Beowulf Workstations", *Proceedings of the 1999 IEEE Aerospace Conference*, March, 1999.

[3] J-M, Chang and N.F. Maxemchuk, "Reliable Broadcast Protocols", *ACM Transactions on Computing Systems*, 2(3):251-273, 1984.

[4] J. Crowcroft and K. Paliwoda, "A Multicast Transport Protocol", in Proceedings of ACM SIGCOMM'88, 1988.

[5] P. Danzig, "Flow Control for Limited Buffer Multicast," *IEEE Transactions on Software Engineering,* 20(1-12), January 1994.

[6] S. Deering, "Multicast Routing in Internetworks and Extended LANs", *ACM SIGCOMM Computer Communication Review*, 1995

[7] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing" *IEEE/ACM Transactions on Networking*, December, 1997.

[8] William Gropp, E. Lusk, N. Doss and A. Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard", *MPI Developers Conference*, 1995.

[9] H.W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log–based receiver-reliable multicast for distributed interactive simulation," *proceedings of SIGCOMM'95*, Cambridge, MA, 1995.

[10] A. Koifman and s. Zabele, "RAMP: A Reliable Adaptive Multicast Protocol," *Proceedings of IEEE INFOCOM*, pp1442–1451, March 1996.

[11] D. Lahaut and C. Germain, "Static Communications in Parallel Scientific Programs." *Proceedings of PARLE*, 1994.

[12] R. Lane, S. Daniels, X. Yuan, "Performance of ACK–Based and Tree–Based Reliable Multicast Protocols over Ethernet–Connected Networks." *Submitted for publication.*

[13] B. N. Levine and J.J. Garcia-Luna-Aceves "A Comparison of Reliable Multicast Protocols." *Multimedia Systems*, 6:334–348, 1998.

[14] P.K. Mckinley, R. T. Rao and R. F. Wright, "H-RMC: A Hybrid Reliable Multicast Protocol for the Linux Kernel", *Proceedings of IEEE SC99: High Performance Networking and Computing*, Portland, Oregon, November 1999.

[15] The MPI Forum, The MPI-2: Extensions to the Message Passing Interface. *http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html*, July, 1997.

[16] S. Paul, K. K. Sabnani, J.C. Lin and S. Bhattacharyya, "Reliable Multicast Transport Protocol RMTP", *IEEE Journal on Selected Areas in Communications*, vol. 15, pages 407–421, April 1997.

[17] S. Pingali, "Protocol and Real–Time Scheduling Issues for Multimedia Applications." *PhD thesis*, University of Massachusetts, Amhest, 1994.

[18] J. Postel, "User Datagram Protocol", RFC 768. 1980. Available at "ftp://ftp.isi.edu/in-notes/rfc768.txt".

[19] J. Postel, "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793. Available at "ftp://ftp.isi.edu/in-notes/rfc793.txt".

[20] R. Manchek, "Design and Implementation of PVM version 3.0", Technique report, University of Tennessee, Knoxville, 1994.

[21] S. Ramakrishnan, B. N. Jain "A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LAN", *IEEE INFOCOM*, pages 502–511, April 1997.

[22] C. Salisbury, Z. Chen and R. Melhem, "Modeling Communication Locality in Multiprocessors", *The Journal of Parallel and Distributed Computing*, vol 56, no 2, pp. 71-98, 1999.

[23] J.M. Squyres, A. Lumsdaine, W.L. George, J.G. Hagedorn and J.E. Devaney "The Interoperable Message Passing Interface (IMPI) Extensions to LAM/MPI" *MPI Developer's Conference*, Ithica, NY, 2000.

[24] R. Talpade and M.H. Ammar, "Single Connection Emulation (SCE): An Architecture for Providing a Reliable Multicast Transport Service," *Proceedings of the IEEE International Conference on Distributed Computing Systems*, Vancouver, Canada, June 1995.

[25] Andrew Tanenbaum, *Computer Networks*, 3rd edition, Prentice Hall, 1996.

[26] B. Whetten, S. Kaplan, T. Montgomery, "A High Performance Totally Ordered Multicast Protocol," *Proceedings of INFOCOM'95*, 1995.

[27] R. Yavatkar, J. Griffioen and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications," *Proceedings of the ACM Multimedia'95*, November 1995.