An Empirical Study of Reliable Multicast Protocols over Ethernet–Connected Networks *

Ryan G. Lane Scott Daniels Xin Yuan
Department of Computer Science, Florida State University, Tallahassee, FL 32306
{ryanlane,sdaniels,xyuan}@cs.fsu.edu

Abstract

Recent advances in multicasting over the Internet present new opportunities for improving communication performance in clusters of workstations. 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 architectural features of local area networks (LANs) on 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 study 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. We evaluate the performance of the protocols over Ethernet–connected networks, study the impact of architectural features of the Ethernet on the performance of the protocols, and investigate the methods to exploit these features to achieve the best 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 [14] and PVM [16], have been implemented for such systems. Current implementations, such as MPICH[8] and LAM/MPI[19], 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 current implementations of message passing libraries 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. The standard IP multicast [6] performs unreliable multicast over most local area networks (LANs), such as Ethernets, without extra hardware/software support. In such systems, efficient reliable multicast primitives 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 network and the CPU processing at the end hosts.

Although many reliable multicast protocols have been proposed [1, 3, 4, 7, 9, 10, 13, 15, 20, 21, 22], mostly for the wide area Internet environment, the impacts of architectural features of LANs on the performance of the protocols have not been thoroughly studied. To develop efficient reliable multicast protocols for clusters of workstations, the implication of running parallel applications over clusters must be examined and special features of LANs must be considered.

This paper studies the impacts of LAN features on the reliable multicast protocols. All the major existing reliable multicast protocols, including the *ACK–based protocols* [20, 15], the *NAK–based protocols* with polling [17], the *ring–based protocols* [3, 21], and the *tree–based protocols*[15, 22], are considered. We present our implementation of the four types of reliable multicast protocols, evaluate the performance of the protocols over Ethernet–connected networks, study the impact of architectural features of the Ethernet on these protocols, and investigate the

*This project was supported in part by NSF grants: CCR-9904943 and CCR-0073482.

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 for 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 presents the protocols we studied. Section 4 discusses the implementation issues. 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, 13, 15, 20, 21, 22]. A summary of recent development of reliable multicast protocols can be found in [12]. The protocols can be broadly classified into four types, the ACK–based protocols [20, 15] where all receivers send positive acknowledgments for each packet that they receive, the NAK–based protocols [4, 7, 9, 10, 22] 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, 21] where the receivers are organized as a logical ring and take turns to acknowledge the packets received to ensure reliability, and the tree–based protocols [15, 22] where the receivers are organized as subgroups to release the sender from processing all control messages from all receivers. This research does not invent new protocols. Instead, we investigate the impact of architectural features of 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.

3 Reliable multicast over the Ethernet

In the general Internet environment, reliable multicast 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.

In a homogeneous cluster, some of the issues are no longer problems. In such a system, all workstations have the same CPU power and network connection, the receivers are homogeneous. In addition, communication patterns in parallel programs change very slowly [18], which indicates that for most communication patterns in parallel programs, multicast groups are static and that the group membership maintenance is not a problem. We will assume in this paper that a multicast group does not change and focus on the performance issue. Some LAN features that may affect the reliable multicast protocols are the followings.

- 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.
- The sender and the receivers are close to each other in LANs and the 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 quickly, large buffers may not be necessary to achieve good performance.
- In a wired LAN, the transmission error rate is very low. The retransmission mechanism may not be as critical as that in the general Internet environment.

This paper studies the impacts of these features. Next, we will describe the four types of protocols used in the study, the ACK–based protocols, the NAK–based protocols with polling, the ring–based protocols, and the tree–based protocols. Since some of the protocols were designed for the wide area Internet, we modify the protocols slightly for the LAN environment.

ACK–based protocols

The ACK–based protocols [7, 15, 20] are extensions of reliable unicast protocols. The sender uses multicast to transmit data to the receivers and the receivers unicast acknowledgments (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. 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, 22] 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 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 required. Thus, in practice when the buffer space is limited, an additional mechanism, such as polling [17], must be incorporated. 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 better scalability. The limitation, however, is that other mechanisms must be incorporated into a NAK–based protocol to offer reliable multicast service. Our implementation also incorporates a NAK suppression scheme to avoid retransmit packets too eagerly. The sender retransmits a packet only after a designated period of time has passed since the previous transmission and at least one NAK is received. Thus, the receivers may send multiple NAKs to the sender while the sender performs retransmission only once.

Ring-based protocols

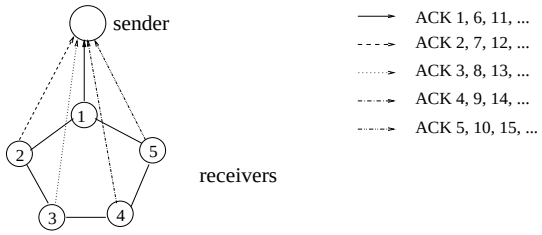


Figure 1. The ring–based protocol

In ring–based protocols [3, 21] , for each packet, only one designated receiver, called the token site, is responsible for sending an ACK packet. The ACK packet is also multicasted to all other receivers. Receivers send NAKs to the token site to request the retransmissions if they detect transmission errors. Each receiver will take turns to be the token site. This rotating token passing scheme enables the source to know the status of each of the receivers once all receivers send ACKs. 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 which implies it has received packets 0 and 1 successfully, and so on. After the sender sends packet $N-1$ and receives the ACK for packet $N-1$ from receiver $N-1$ (and all previous ACKs from other receivers), it can safely release packet 0 from memory. Figure 1 depicts a ring–based protocol.

Three modifications were made in our implementation. First, receivers unicast ACKs to the sender, but not multicast ACKs to other receivers. Each receiver ACKs packets based on its node number. This modification may delay the detection of transmission errors. However, it reduces the computational load at the receivers since receivers do not have to process all ACKs. More importantly, this modification decreases the possibility of out-of-order ACK packets, which makes the protocol more robust. Second, we modify the protocol so that all receivers always acknowledge the last packet of a message. Third, NAKs are sent directly to the source (not the token site) and the source will perform the retransmission. This modification has minor effects on the source since retransmissions are rare due to the low transmission error rate.

Tree-based protocols

The tree–based protocols [15, 22] are characterized by dividing the receiver set into groups with each group having a group leader. The sender only interacts with the group leaders while the group leaders interact with their group members. In these protocols, the group leaders aggregate the acknowledgments in the group and are responsible for packet retransmissions within their groups. Tree–based protocols overcome the ACK implosion problem and reduce the CPU processing at the sender in comparison to the ACK–based protocols.

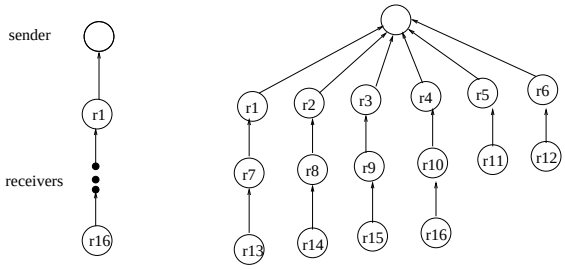


Figure 2. The linear tree structure

We made two modifications in our implementation. First, the group leaders only aggregate ACKs, but do not perform retransmission. The sender is responsible for all data packet transmissions and retransmissions. This modification does not significantly affect the computational load at the sender since the transmission error rate is very low.

This modification greatly reduces the implementation complexity. Second, we use a logical structure called a *linear tree* to organize the receivers. The logical structure of a linear tree is determined by two parameters, the number of receivers, N, and tree height, H. Logically, the receivers are organized as H rows and $\frac{N}{H}$ columns as shown in Figure 2. Notice that the ACK–based protocol is a special case of the tree–based protocols, a linear tree with $H = 1$. A receiver will wait for the acknowledgment for a packet from its successor in the same column before it sends the acknowledgment to its predecessor in the same column. Thus, This logical structure limits the maximum number of simultaneous transmissions to be $\frac{N}{H}$.

Each type of protocols has its own advantages and limitations. Table 1 summarizes the protocols.

Table 1. Comparison of the protocols

protocol	network traffic	memory req.	CPU req.	protocol complexity
ACK	high	low	high	low
NAK	low	high	low	low
Ring	low	high	low	high
Tree	high	low	low	high

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 control scheme and maintain a timer at the sender side to implement the sender–driven error control mechanism. Due to the space limitation, we will omit the discussion of the implementation issues. More details about the implementations can be found in [11].

5 Performance

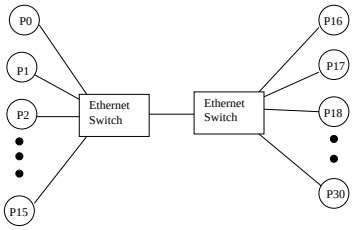


Figure 3. Network configuration

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 3. In the figure, $P0$ is the sender and all others are receivers. To obtain accurate experimental results, we measure the communication time three times for each experiment and report the average of the three measurements. Since transmission errors rarely occur in the experiments (among around 150 to 200 experiments, retransmissions only occur 1 to 2 times), all the results reported are the cases without retransmission.

ACK–based protocols

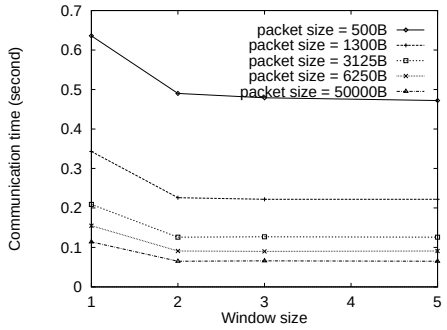
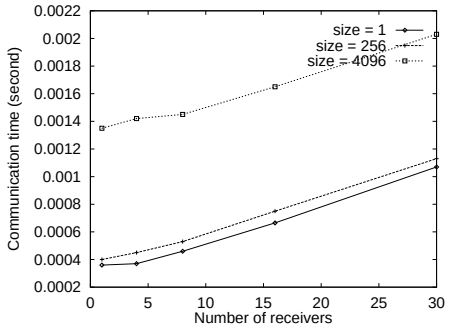


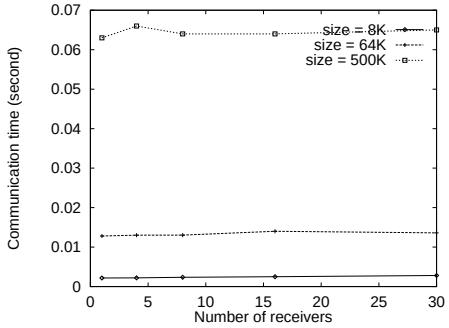
Figure 4. ACK–based protocols with different packet sizes and window sizes

Figure 4 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. Notice that the total amount of buffer memory is equal to packet size times window size. The performance of the ACK–based protocol is very sensitive to the packet size, the larger the packet size, the better the performance. A large packet size reduces the number of acknowledgments that the receivers send, which reduces both the network traffic and the CPU processing in the sender. Another interesting observation is that for any given packet size, the best performance can be achieved with a window size of 2. The ACK–based protocols cannot exploit the pipeline communication and make use of large buffers since data packets are competing with acknowledgment packets for the same network resources and since the round trip delay is small in the Ethernet.

Figure 5 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 (UDP) 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



(a) Small message sizes



(b) Large message sizes

Figure 5. Scalability of ACK–based protocols

receivers as shown in Figure 5 (a). This is because the total number of messages increases linearly as the number of receivers increases. Figure 5 (b) shows this case when the message size is large ($> 8KB$). In this case, the acknowledgment overhead is negligible and the protocol is scalable. Similar results are obtained for the other three protocols.

NAK–based protocols

The performance of NAK–based protocols with polling is greatly affected by the timing of the polling. Due to the space limitation, we omit the results for the study of the timing of polling. The results can be found in [11]. Our main conclusion from the study is that when the polling interval is about 80% to 90% of the window size, the NAK–based protocols perform the best. In the rest of the experiments, we will assume that the polling interval is set in this range.

Figure 6 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. The window size can be determined by dividing the buffer size by the packet size. The results show that both a small packet size (500B) and a large packet size (50KB)

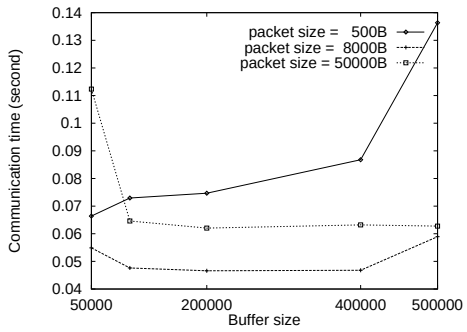


Figure 6. NAK–based protocols with polling for different buffer sizes

result in poor performance. A medium packet size (8KB) achieves the best performance. In the NAK–based protocol, the sender does not need to wait for and process the ACK packets. Thus, a balance between the data packet transmission time and the data packet processing time (copying data from the user domain into the buffer and other processing) in the sender is the key to maintain efficient pipeline communication and to fully utilize the system resources. Such balance is achieved with a packet size of around 8KB in our experiment. This figure also shows that for a given buffer size, the NAK–based protocols require a large window size to achieve good performance. For example, when buffer size $= 50K$, a packet size of $500B$ (window size $= 100$) performs better than a packet size of 50KB (window size $= 1$). A final observation is that the performance of the protocol is not strictly increasing or decreasing relative to the packet size, which indicates that the performance of the NAK–based protocols depends on a number of factors, including the buffer size, the packet size, and the poll interval.

Ring–based protocols

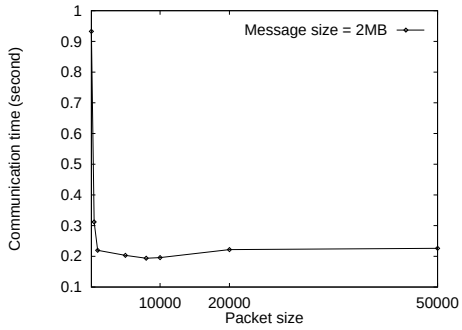


Figure 7. Ring–based protocols with different packet sizes

Figure 7 shows 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 35. As can be seen from the figure, similar to the NAK–based protocols, both a large packet size and a small packet size result in poor performance. The best results were obtained when the packet size was between $5KB$ and $10KB$. The ring–based protocols are similar to the NAK–based protocols and similar arguments for the NAK–based protocols can also apply to this experiment.

Figure 8 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 achieves the best performance is related to the size of the packets. The ring–based protocols require the window size to be at least one more than the number of receivers. For a reasonable packet size, such as $8KB$, when the number of receivers is sufficiently large, this minimum window size requirement is typically sufficient to give good performance. Notice that like in the NAK–based protocols, the sender in the ring–based protocols only processes a small number of acknowledge packets, which allows the pipeline communication to be exploited with a large window size.

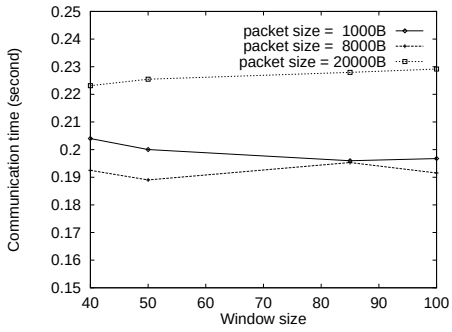


Figure 8. Ring–based protocols with different window sizes

Tree–based protocols

Figure 9 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 ($H = 1$ and $H = 30$) do not produce the best communication performance for either packet size. The performance of the tree–based protocols depends on both the tree structure and the packet size. For example, when packet size $= 8KB$, the tree structure with $H = 15$ results in the best performance. When packet size $= 50KB$, the tree structure with $H = 6$ performs the best. In all cases except

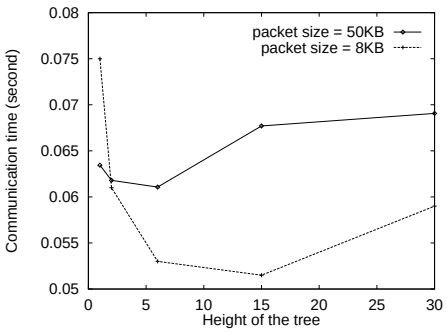


Figure 9. Tree–based protocols for different logical structures

$H = 1$, using an 8KB packet size performs better than using a $50KB$ packet size. Small packets help in creating the pipeline communication, which results in better communication performance. Although using small packets requires more acknowledgments, the sender only processes the aggregated acknowledgments. The accumulated effect is that using $8KB$ packets performs better than using $50KB$ packets for most cases.

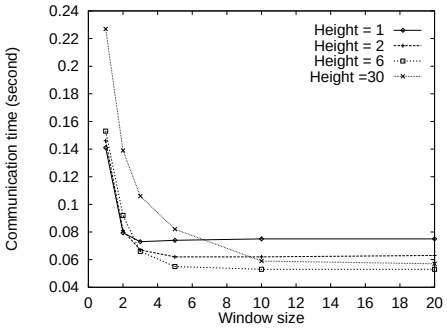


Figure 10. Tree–based protocols for different window sizes

Figure 10 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. 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 ($H = 1$) performs worse than other tree protocols with a sufficiently

large window size. Two factors contribute to the performance gains. First, the tree–based protocol reduces the load on the sender which is the bottleneck in the multicast communication. 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.

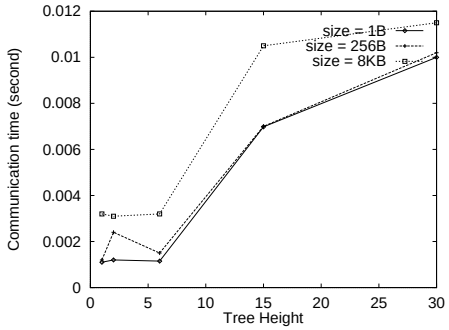


Figure 11. Tree–based protocols for small messages

Figure 11 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.

Comparison of the protocols

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 are the same. All these protocols perform better than the tree–based protocols as shown in Figure 11. This is because the tree–based protocols relay ACK packets while the other protocols do not.

Table 2 summarizes the results for large messages. This experiment compares the "best" protocol in each type for transferring $2MB$ data to 30 receivers. The "best" protocol is obtained by probing the parameter space for each type of protocols and selecting the ones that can provide the best performance. 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 (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. For large messages, the performance of the protocols is ordered as follows:

Table 2. Throughput for large messages

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

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

The NAK–based protocol with polling is able to achieve the best performance since it requires the least CPU processing at the sender. The ring–based protocol also achieves high performance due to the fact that it only processes one ACK for each packet. The tree–based protocol out–performs the ACK–based protocol since it reduces the CPU load at the sender and allows more pipeline communication. However, The overhead of relaying messages prevents the protocol from achieving the throughput of the NAK–based protocols. The ACK–based protocol exhibits extremely poor performance due to the ACK implosion problem.

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 protocols. The main conclusions we obtained for reliable multicast over Ethernet–connected networks are the following.

• **ACK–based protocols**. First, the 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. Second, the performance of ACK–based protocols is quite sensitive to the packet size. The larger the packet size, the better the performance.

• **NAK–based protocols with polling**. First, the packets size must be carefully selected to achieve good performance. Second, a large window size is necessary to achieve good performance. Third, the polling interval plays an important role in the performance of the NAK–based protocols.

• **Ring–based protocols**. The ring–based protocols are somewhat similar to the NAK–based protocols: they require

a large window size and a carefully selected packet size to perform well.

• **Tree–based protocols**. First, 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, the optimum packet size for 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. All of these protocols perform better than the tree–based protocols. For large messages, the performance of the protocols is ordered as follows:

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

Based on the study in the paper, we conclude that the NAK–based protocol is the most efficient reliable multicast protocol for Ethernet-connected clusters.

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*, Dec., 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] R. Lane, S. Daniels and X. Yuan, "An Empirical Study of Reliable Multicast Protocols over Ethernet–Connected Networks." *TR-010503*, Technical Report, CSD, Florida State University, May 2001.

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

[13] 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*, Nov. 1999.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

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

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