A SIMULATION STUDY OF THE PERFORMANCE OF MULTICAST PROTOCOLS

ANDREA M. SEGOVIA

A Simulation Study of the Performance of Multicast Protocols

by

Andrea M. Segovia

A thesis submitted to the
School of Graduate Studies
in partial fulfilment of the
requirements for the degree of
Master of Science

Department of Computer Science
Memorial University of Newfoundland

September 1996

St. John's Newfoundland

Abstract

We have chosen to study multicast protocols because of their applicability to a growing number of network applications. We have designed and implemented a simulation system to study multicast protocols without the need for a dedicated testbed network, and used the simulator to study two simple multicast protocols, the stop and wait and block acknowledgment protocols. We found that these protocols outperform equivalent unicast protocols for small number of receivers in error-free conditions. The block acknowledgment protocol shows higher throughput with larger window sizes, but has greater latency. Both protocols, however, deteriorate quickly as the number of receivers and failure rate of the underlying network grow. This deterioration is caused by a rising collision rate among receivers sending acknowledgments which in turn aggravates network congestion. Throughput of both protocols is improved by using randomly timed acknowledgments to reduce the collision rate; however, this technique does not affect the high sensitivity of the protocols to error conditions.

Acknowledgments

I would like to thank my supervisor, Dr. Rodrigue Byrne, for his invaluable assistance throughout the course of this project. Dr. Paul G. Gillard provided academic advice and support during my entire Master's programme.

I would also like to thank Larry W. Coady, former Regional Director of Science, Dr. J. Scott Campbell, Acting Regional Director of Science, Dr. Derek H. Shaw, Division Manager, Environmental Sciences, and Garfield D. Somerton, Section Head, Environmental Monitoring, Science Branch, Department of Fisheries and Oceans, for generously providing educational leave and departmental resources so I could complete this project. Special thanks to Garfield D. Somerton for his expert SAS advice.

Finally, I would like to thank my husband, James F. Linehan, for his unfailing support and encouragement, without which this project would never have been completed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The growth in the availability and speed of computer networks has resulted in a growing interest in distributed network applications, and subsequently, in the development and study of the multicast protocols on which many of these distributed systems are based. Extensions to the Unix operating system [17] [9] and to the Internet suite of protocols [10] [11] to support multicasting have been proposed and developed.

Some multicast protocols [20] [18] [21] [29] [30] [28] are designed specifically for use by distributed database systems, providing strict ordering properties. Other multicast protocols are designed to enable distributed processing [12] [32] [19]. Still other multicast protocols concentrate on LAN environments, especially those with hardware multicasting capabilities. Reliable multicast protocols for such LAN environments [3] [2] [15] [33] [1] [4] [5] [7] [8] have been discussed in the literature.

We chose to study multicast protocols in an Ethernet LAN with hardware multic-

asting capabilities. We have designed a simulator in which to study such protocols, and use the simulator to study two simple multicast protocols.

Chapter 2 begins by giving basic definitions. It then discusses the characterization of multicast protocols by some common features: support of static or dynamic multicast groups, support for multiple sources, ordering guarantees for transmitted messages, types of acknowledgment schemes, buffer and bandwidth requirements, congestion avoidance techniques, and fault tolerance and detection. Some common performance measures for multicast protocols are then discussed, including data throughput, which measures how much data the protocol can deliver within a unit of time, and message latency, which measures how long the protocol takes to reliably deliver messages.

The chapter then covers the details of the two simple multicast protocols to be studied, the stop and wait and block acknowledgment protocols. The stop and wait protocol sends messages one at a time, waiting for acknowledgments from all receivers before proceeding. The block acknowledgment protocol sends messages in blocks, permitting receivers to acknowledge multiple messages with a single acknowledgment message.

Three multicast protocols from the literature are described. The main feature of the first protocol [24] is the use of a sequencer node to order messages from different receivers and handle retransmissions. One disadvantage of the protocol is that messages are sent twice; once to the sequencer node and again in a multicast transmission. This protocol is also highly dependent on the sequencer node.

The protocol by Erramilli and Singh [14] is highly dependent on timeout parameters for its operation. In fact, the protocol can fail if the parameters are not carefully chosen. Receivers advertise their status using status/sanity messages sent at specified intervals during idle periods. Messages from multiple senders are independent sequences of messages, and no guarantee of their relative ordering is given.

The protocol by Maxemchuck and Chang [6] is actually a family of protocols with different fault-tolerant properties. The protocols combine the features of a multiple source/single receiver system, which simplifies message ordering, and a single sender/multiple receiver system, which uses negative acknowledgments to reduce the number of acknowledgments required.

A simulator developed to study the behaviour and performance of multicast protocols in an Ethernet LAN environment is discussed in chapter 3. The simulator currently supports the stop and wait and block acknowledgment protocols. The simulator's layered design is easily extensible; other multicast protocols can be added to the simulator. The simulator provides timing data to measure protocol performance which can then be analysed and plotted for further study. This chapter describes the overall design of the simulator, including what is required to add a new protocol.

The fourth chapter presents and explains the results of the simulation studies. The performance of both the stop and wait protocol and the block acknowledgment protocols under simulation was measured and analysed. Both protocols performed better than equivalent unicast protocols in error-free conditions. Both protocols' throughput was improved by using randomly timed acknowledgments to reduce the rate of

collisions. The block acknowledgment protocol's throughput was better than the stop and wait protocol's throughput as the window size of the block increased; however, the block acknowledgment protocol had significant increases in message latency as a result. Both protocols had a low tolerance for errors.

Finally, the fifth chapter gives some concluding remarks, and presents suggestions for further work, including suggestions for improving the error tolerance of the two protocols studied, and an outline of a third simple multicast protocol, the round robin acknowledgment protocol.

Chapter 2

A Description of Multicasting Protocols

2.1 Motivation

Computer technology has evolved from single isolated computers performing mostly numerical computations to large networks of computers which cooperate to perform a wide variety of tasks. Much of the cooperation between computers is achieved by having the computers communicate information among themselves.

Computers communicate using predefined protocols. These protocols define how and when each computer should communicate its information to other computers. Communications protocols are often reliable, meaning that the protocol includes provisions for detection and recovery of errors encountered on the communication me-

dium.

Many common network applications, such as file transfer, remote terminal access, and electronic mail only require that two computers be able to participate in a communication at any one time. These applications use point-to-point (unicast) protocols to communicate.

Other applications, such as distributed processing, distributed database systems, and multimedia teleconferencing systems, involve communication among several computers at once. While this can be achieved using multiple point-to-point communications, these applications would benefit from communication protocols designed for one-to-many or many-to-many communication. Broadcast protocols permit communication among all computers in a network. These protocols are often used for network management functions in which all computers must participate, such as the broadcasting of network routing information. Most broadcast protocols are not reliable, but some reliable broadcast protocols do exist. Multicast protocols permit designated groups of computers, often termed multicast groups, to communicate among themselves. Multicast protocols can be either reliable or unreliable.

Applications which are distributed over a group of computers derive the most benefit from multicast protocols. Consider a fault-tolerant distributed system which replicates its file structure in several computers at once. All modifications to the file system must be sent to all computers storing the file system. Implementing this transfer using several point-to-point links is wasteful, as exactly the same information is being transmitted several times. However, reliability and certain ordering properties

are required, so conventional unreliable broadcast or multicast algorithms won't do.

As well, continuing improvement in multimedia technologies has sparked interest in desktop teleconferencing applications. The large bandwidth required for the transmission of real-time audio and video data necessitated by teleconferencing is a significant drain on network resources. Multicasting this data to teleconference participants will reduce the required network bandwidth significantly, and may also reduce the latency of the teleconferencing application, a critical performance measure in a real-time environment.

Other applications which may benefit from multicast protocols include the transmission of Usenet news, multi-user chat programs, distributed multi-user computer games, etc.

2.2 Characterization of Multicast Protocols

Communication protocols [22] can be characterized as reliable, or unreliable. An unreliable protocol will send information on a "best effort" basis, but will not resend lost messages or guarantee that messages will arrive in order or uncorrupted. Unreliable protocols are often datagram or connectionless protocols; messages are sent to the destination in a manner analogous to a letter posted at the post office.

A reliable protocol will ensure that messages arrive at the intended destination; retransmissions and acknowledgment messages from the receiver are the most commonly used mechanisms to implement this reliability. Reliable protocols may also be

connection-oriented; messages are sent to the destination in a manner analogous to a telephone call, i.e., using distinct phases to perform call-setup, conversation (or data transfer), and call-disconnect.

Unicast protocols describe communication between exactly two computers; unicast protocols may be reliable or unreliable, connectionless or connection-oriented. Multicast protocols describe communication between many computers; multicast protocols may also be reliable or unreliable, but are rarely connection-oriented. Broadcast protocols describe communication between all the computers in local area network; as such, they may be considered as a special case of multicast protocols.

Reliable multicast protocols [3] [2] [15] [33] [1] [4] [5] [7] [8] [20] [18] [21] [29] [30] [28] can be characterized by a few important features, as follows:

- support of static or dynamic multicast groups. Static multicast groups retain the same membership throughout the course of a multicast communication. Dynamic multicast groups, however, may change their membership during the course of a multicast communication, by adding or deleting nodes dynamically throughout. In order to support dynamic multicast groups, a multicast protocol must provide mechanisms for adding and deleting nodes from the group as well as a mechanism to keep all nodes informed of the current membership status.

- support for multiple sources. A source is a node in the multicast group which transmits multicast messages. A multicast protocol may support only single source or multiple sources in one communication.

- ordering properties of transmitted messages. A protocol may deliver all messages reliably, but make no guarantee about the ordering of the messages received. Such a protocol is said to deliver messages unordered.

  Other protocols may deliver messages in the order sent by the sender. When multiple senders are involved, these protocols do not make any guarantees about the relative ordering of messages sent by different senders, or even that all the receiving nodes will receive messages from different nodes in the same order. However, if only one sender is involved, the result is an ordering of messages which is identical at all receiving nodes.

  Yet other protocols provide even stronger ordering properties for messages, guaranteeing a "global" order for messages. These protocols will guarantee that all messages from all senders are received by all nodes in exactly the same order. These protocols do not necessarily guarantee that all messages will be delivered in exactly the time sequence sent when originating from different senders. Two of the more common mechanisms for ensuring global ordering of messages, the use of a timestamp or a sequencer node, will accomplish that goal.

- acknowledgment schemes for reliability. Acknowledgment schemes use either positive acknowledgments, that is, all messages properly received are acknowledged by sending acknowledgment messages to the source (or other intermediary node); or negative acknowledgments, that is, all messages detected as corrupt or lost by the receiver are acknowledged by sending a negative acknowledgment

(implied retransmission request) to the source (or other intermediary node); or a combination of both. Attempts to reduce the number of acknowledgments needed by sending implicit multiple acknowledgments in one acknowledgment message and/or by "piggybacking" acknowledgments onto other data messages are also common variations.

- buffer requirements for sources/receivers. Buffer requirements differ widely among various protocols. A protocol may require as few as a one message buffer for all nodes (stop and wait), or n message buffers at source nodes and a single message buffer at receivers (go-back-n), or n message buffers at both source nodes and receivers (selective repeat), or n message buffers at sources and m_i message buffers at each receiver i (selective repeat protocols with flow control). Yet other protocols can dynamically adapt to changing buffer resource levels, using as many or as few buffers as the system will allow.

- congestion avoidance techniques. Protocols may make no provision for congestion control; or attempt to reduce the number of transmitted messages in order to avoid congestion; or include mechanisms for detecting and dealing with network congestion.

- fault tolerance and detection. A multicast protocol may break down completely if a fault occurs in one or any of the multicast group; it may be able to detect the fault and stop; or it may be able to operate when a given number of sites fail.

- quality of service guarantees. Some multicast protocols can provide differing levels of service as requested; most often these levels of service involve reliability and ordering guarantees. Others provide just one level of service.

2.3 Performance Measures

How do we compare the performance of one protocol to another? We need to have a series of performance measures which quantify the performance of a protocol in different ways, so that we can decide which protocol is best for our needs.

The most obvious measure of a protocol is the amount of data per unit of time which it can deliver, normally termed throughput. In the case of multicast protocols, the measurement is based on the total amount of data which is received by the multicast nodes within a unit of time. This measurement reflects the fact that the amount of "effective" data is actually the data transmitted multiplied by the number of receivers receiving that data.

Throughput values in isolation are not always informative; comparing the throughput values of multicast protocols to the throughput values of the equivalent multiple unicast transmissions will more clearly show whether the use of a given multicast protocol is beneficial. To that end, we will measure the relative throughput of a multicast protocol as the ratio of the absolute throughput of the multicast protocol to the absolute throughput of the equivalent unicast transmissions; a multicast protocol with a relative throughput of 1.0 has performance equivalent to using multiple point-to-point

transmissions. Multicast protocols with relative throughputs greater than 1.0 show increased performance over the equivalent unicast point-to-point transmissions.

Both absolute and relative throughput measurements will depend on the error rate of the underlying communication media, and is usually given in both best case (no error rate - best possible throughput), and average case (average error rate - most likely expected throughput value) conditions. The best case throughput can be used to measure the message overhead incurred by a protocol.

Another measure of performance is message latency. Message latency measures the amount of time from the initial transmission request until the application receives the transmitted data. This measure includes transmission time of the medium, processing times, retransmittal and other overhead incurred in the transmission of a message. In the case of multicast protocols, the message latency measure includes the amount of time required in order to ensure that all receivers receive the message, and in the ordering guaranteed by the protocol. This measure is also affected by the transmission and error rate of the medium, so it is often given in both best case and average case conditions. Moreover, this measurement may also be given in worst case conditions. This worst case measurement gives the greatest amount of time required to transmit a message from application to application. It is an important factor when evaluating a protocol for real-time applications, where response time is critical.

One aspect of protocol performance which only applies to multicast protocols is scalability with respect to the number of receivers. How do other measures of performance discussed above degrade as the number of nodes increases? If the degradation

is linear with respect to the number of receivers, then the protocol scales well, and will not be unduly affected by increasing numbers of receivers. If the degradation is exponential or worse, then the protocol will quickly become unusable as the number of receivers increases. This measurement will help in evaluating protocols for applications where the number of receivers is large.

Another aspect of protocol performance is the amount of resources required to achieve the given performance. For example, a measure of the amount of network bandwidth utilized by the protocol as a function of transmitted data quantifies the amount of overhead messages such as acknowledgments and other status messages required by the protocol. It provides us with a measure of the network bandwidth required by the protocol in relation to the amount of data being transmitted and can be useful in evaluating protocols to be used in environments with high network traffic.

Another measure of the resources required by the protocol is buffer space requirements of sender(s) and receivers as a function of the number of receivers and the data throughput value. This measure helps determine how increased buffer space will improve data throughput for a given number of receivers, and helps determine an appropriate tradeoff point between space requirements and throughput achieved.

The number of interrupts generated at a node in order to process incoming packets often affects the processing ability of the node. In the case of the multicast packets, all nodes which receive the multicast packet generate an interrupt to process the packet and determine if it should be kept and passed along to upper network software layers. A measure of how many interrupts are generated per receiver as a function of data

throughput (perhaps number of messages, instead) will clearly show the impact of various protocols on the processing ability of the receivers as affected by interrupt processing.

2.4 Unicast versus Multicast Protocols

Unicast protocols are much simpler to design and implement than multicast protocols for several reasons. Unicast protocols need to provide reliability; however, they only need to handle acknowledgments from one node; only need to order messages from one source; do not have to provide fault-tolerant mechanisms, since when one of the two nodes in the communication fails, the communication stops; and do not have to provide support for nodes dynamically joining or leaving during a communication. Multicast protocols, on the other hand, must handle acknowledgments from a (possibly dynamic) number of receivers, as well as the ensuing network congestion; must be able to detect errors in data transmission to multiple receivers and handle the required (possibly multiple) retransmissions; may enforce stricter ordering properties for received messages; may provide some measure of fault-tolerance, since the failure of one or more nodes does not automatically make continuing the communication fruitless; and may also have to support dynamic multicast groups.

These issues add a considerable amount of complexity to a multicast protocol. Consider the number of acknowledgments required in a multicast protocol for r nodes. If one acknowledgment from each receiver is required for each message, then the num-

ber of acknowledgments per message is simply r. Now the number of acknowledgment messages is tied to the number of nodes participating in a communication, and therefore the protocol's throughput deteriorates as the number of nodes increases. This reduces the scalability of the protocol. Many multicast protocols attempt to reduce the number of acknowledgment messages required in an effort to improve the protocol's performance when using a large number of nodes. Mechanisms for reducing the number of acknowledgments include piggybacking several acknowledgments in one message; using negative acknowledgments, and various other combinations. Of course, these mechanisms come with a price: the amount of time required to detect a lost message usually increases, and the amount of buffer space required may also increase.

Another important issue is that of ordering. Many applications require that multicast messages be totally ordered. For example, a fault-tolerant distributed database system that is sending transaction information originating from various nodes to a replicated database using multicast messages, must guarantee that all transaction information arrives in exactly the same order at all nodes to maintain consistency among the replicated copies of the database. Totally ordering messages from different sources requires some sort of global timestamp or a sequencer node to sequence all messages. Synchronization of clocks in a distributed environment is complex. Using a sequencer node reduces the fault-tolerant properties of the protocol, since failure of the sequencer node will cause the entire communication to fail. Mechanisms to reduce this vulnerability usually involve rotation of the sequencer node responsibilit-

ies among all nodes coupled with a recovery mechanism should the acting sequencer node fail. While these problems are surmountable, they add a considerable amount of complexity to the design of a reliable multicast protocol.

Yet another issue is retransmission policy implemented by the multicast protocol. Unicast protocols simply retransmit a lost message to its intended recipient; multicast protocols must determine which receiver lost the message and may handle the retransmission by either sending a unicast message to the affected receiver or multicasting the retransmission to the entire group. Unicasting retransmissions may reduce the amount of time spent by receivers processing multicast messages; however, multicasting retransmissions may result in fewer retransmissions overall, reducing network congestion under high traffic conditions.

2.5 Multicast Stop and Wait Protocol

The simplest unicast protocol is a stop and wait protocol. In this protocol, the source sends its message and waits for an acknowledgment from the single receiver before sending another message. Advantages of this protocol include a single message buffer requirement both at the source and receiver, and a simple implementation.

Extending the stop and wait paradigm to a multicast protocol results in a rather simple protocol which supports only a single source and static multicast groups and does not provide any fault tolerance. The acknowledgment scheme uses positive acknowledgments. Sequence numbers are only single-bit, since at most one packet is

outstanding at any one time. The proposed multicast stop and wait protocol also includes several options: random wait intervals before sending acknowledgments, and using either unicast or multicast messages for retransmissions. The basic multicast stop and wait protocol using unicast retransmissions will be described first, followed by the options.

The protocol requires that all receivers and the sender agree on a multicast address and the composition of the multicast group. This can be done either statically or by a preliminary negotiation phase. The sender initializes its data structures, which include a single message buffer, and one timer, retransmission count and acknowledgment buffer per receiver. The sender checks its transmission queue for messages to be sent; if the queue is not empty, the sender encapsulates the first message with the following header information: the sender's address (source), the previously agreed upon multicast address (destination), the sequence number of the message (0 for the first message), and a message type of data. The resulting packet is then multicast to the local area network, a timer is started for each receiver, and the sender enters a waiting phase.

The receivers initialize themselves to expect a data message with sequence number 0, and prepare a single message buffer. Then, the receivers wait for messages. Once a message has arrived for a particular receiver, either explicitly addressed to it or destined for the previously agreed upon multicast address, the receiver checks if the message number is the expected message number. If so, the receiver prepares a unicast packet with a message type of ACK, the sequence number of the message being ac-

knowledged, the receiver's address as the source address, and the sender's address as the destination, and sends the packet. It then increments its expected sequence number by one (note the sequence number is a single-bit quantity), and resumes waiting for messages.

When an acknowledgment is lost, the sequence number of a message received will not be the expected sequence number. When this occurs, the receiver will prepare and send an acknowledgment message as described above, but does not update its expected sequence number. This ensures that lost acknowledgment messages are eventually resent.

When a sender is in its waiting phase one of two events may occur; either an acknowledgment is received or a timer expires. If an acknowledgment is received, the sender will check if the sequence number contained in the acknowledgment message is the sequence number of the currently outstanding message, and whether the source of the acknowledgment is among the multicast group. If the acknowledgment passes both validity tests, then the acknowledgment buffer for the corresponding receiver is updated, and its timer reset.

The sender now checks if all receivers have received the currently outstanding message by examining the acknowledgment vector containing the acknowledgment buffers for each receiver. If all receivers have received the message, then the sender determines that the message has been successfully and reliably multicast and therefore gathers timing information for performance statistics, re-initializes its data structures, including invalidating all active timers, increments the current sequence number, and

informs the upper layer application protocol of its success. It is now ready to send the next message in the transmission queue as described above. If some of the receivers have not acknowledged the currently outstanding message, then the sender continues in the waiting phase.

If a timer expires, then the sender concludes that the corresponding receiver has not received the current message. Note that the sender cannot distinguish between the case where the receiver does not receive the message, and the case where the acknowledgment packet sent by the receiver does not arrive at the sender. The sender checks that the retransmission count for the receiver. If this does not exceed the maximum retransmission count, the sender prepares and sends a unicast retransmission of the current packet to the receiver whose timer expired, and also increments the appropriate retransmission counter.

If the maximum retransmission count has been exceeded, then the protocol fails, having been unable to successfully transmit the multicast message reliably within the allotted time. However, an implementation may just note the failure and continue; relying on an upper layer to detect and correct the problem. This is done in order to place an upper bound on the amount of time used to transmit a message; it is also often the case that upper layer protocols will have some error detection and correction mechanisms that will permit the communication to continue.

2.5.1 Adding Randomly Timed Acknowledgments

In an Ethernet CSMA/CD environment, the multicast stop and wait protocol discussed above can cause congestion in the network. To see why this is so, consider the case of 10 nodes of relatively equal processing power and load participating in a multicast communication. All nodes will simultaneously receive the multicast communication, process it and attempt to send an acknowledgment message on the Ethernet medium. The most likely scenario is that a slightly faster node captures the Ethernet medium, causing all other nodes to wait until the first idle period to send their acknowledgments. When the nodes then attempt to send their acknowledgments, a collision results. While it is true that the collision contention algorithm of CSMA/CD will attempt to prevent further collisions in the future, at least one collision detection cycle must occur for this to happen. So, a variation on the multicast stop and wait protocol discussed above attempts to prevent collisions from occurring by building in a random wait interval to be observed by all receivers before an acknowledgment message is transmitted.

The random wait interval is implemented as follows: the upper bound of the random wait interval is chosen for all receivers; when a receiver is ready to transmit an acknowledgment message, it selects a random number within the chosen time interval, and delays sending the acknowledgment message for this amount of time.

We expect this variation in the multicast stop and wait protocol to improve throughput values significantly, mainly due to a reduction in the number of collisions. The probability of collisions decrease as the chosen wait interval becomes

larger; however, the delays incurred transmitting acknowledgments as the interval increases can also reduce throughput.

This interval also affects other parameters of the protocol, most notably the timeout value used by the sender to determine when a receiver has lost a message. If this value is not chosen with the maximum random wait interval in mind, the sender may send unnecessary retransmissions because it believes that a receiver has lost the current message although the acknowledgment may have been purposefully delayed. The network congestion thus caused may result in a significant deterioration of protocol performance.

2.5.2 Varying the Retransmission Method

Multicast packets in an Ethernet network may be lost in one of two ways. Either the entire multicast packet is garbled on the bus and no node receives it, or particular nodes have difficulty grabbing the packet off the wire. In the first case, it should be obvious that multicast retransmissions are more effective than unicast retransmissions; once the initial multicast packet is lost, using unicast retransmissions reduces the protocol to multiple point-to-point transmissions to correct the error.

In the second case, if the number of receivers which lost any one packet is two or more, the amount of time required for retransmissions is significantly reduced if the retransmissions are multicast. The second variation on the multicast stop and wait protocol discussed above uses multicast retransmissions instead of unicast retransmissions in order to improve protocol performance.

Multicast retransmissions are simple to implement. Retransmissions use the same multicast packet format used for the original transmission of a message; the only significant changes to the above protocol description are the use of a single retransmission count and timer instead of one retransmission count and timer per receiver previously required. Note that the semantics of the maximum number of retransmissions parameters is changed; in the original protocol, this placed an upper bound on the number of retransmissions per message to any one node, in the modified protocol, this places an upper bound on the total number of retransmissions per message.

2.6 Multicast Block Acknowledgment Protocol

The previous multicast stop and wait protocol requires, for r receivers, r acknowledgment messages per message. The multicast block acknowledgment protocol is a simple enhancement to the multicast stop and wait protocol which reduces the number of acknowledgment messages required per message. The multicast block acknowledgment protocol allows a block of n messages to be transmitted at once; each receiver will send one acknowledgment message acknowledging the successful receipt of some or all of the block of messages.

The proposed multicast block acknowledgment protocol supports only single-source transmissions with static multicast groups. It guarantees totally ordered messages, uses a positive acknowledgment scheme, and attempts to prevent network congestion by "piggybacking" several acknowledgments in one message, but does not include any

fault tolerance. It requires an n message buffer at the sender, but only needs a single message buffer at each of the receivers.

The sender, which has a maximum window of size n, may send up to n multicast messages at once to the multicast group. The receivers in the group will send one acknowledgment response acknowledging all n received messages. The sender processes the acknowledgments and sends the next n messages. Under ideal conditions, no messages or acknowledgments will be lost and the protocol would be quite simple. However, in more realistic environments, the protocol must be prepared to handle lost packets, and other errors, adding to the complexity of the protocol.

Similar to the stop and wait protocol discussed earlier, messages have sequence numbers associated with them to ensure synchronisation of messages between the sender and receivers. In the stop and wait protocol, however, the sequence numbers are either 0 or 1, reflecting the fact that at most one message was outstanding at one time. In the block acknowledgment protocol, the sender has a window of up to n messages which may be outstanding at one time. The range of sequence numbers must therefore be 0 to $(2n - 1)$ [22]. The next sequence number can be calculated as

$$s_{\mathrm{i}\,+\,1} = (s_i + 1) \,\mathrm{modulo}\, 2n. \tag{2.1}$$

The sender's window is defined as the ordered list of sequence numbers corresponding to currently outstanding messages, which may be fully specified by giving the current size of the window, c, the sequence number of the first message sent in the window, s, and the maximum size of the window, w. Given this specification of the sender's window, we can test whether a message falls into the window. As well, given

a vector of sequence numbers representing the last message acknowledged by each of the receivers, we can calculate the most recently sent message acknowledged by all the receivers. This calculation is done by determining the rank (ordering) of each sequence number within the current window, selecting the lowest rank, and converting the rank back into a sequence number.

The sender must keep all the messages in its window in a buffer; if a message is lost, the sender must retransmit the lost message and all subsequent messages in the current window. The sender must also keep a vector of acknowledgments, one per receiver, to be able to calculate how many messages in its window have been correctly received by all receivers, and another vector to count the number of retransmissions per message in the current window. The sender also has one timer, which is used to bound the amount of time spent waiting for acknowledgments from receivers.

Each receiver keeps a count of messages successfully received but not acknowledged, the maximum size of the sender's window, and the sequence number and source of the last message received. It also uses a timer, which is used to bound the amount of time before an acknowledgment is sent.

The sender and receivers participating in the multicast block acknowledgment protocol must agree on the maximum window size used by the sender, the multicast address for the group, and the membership of the group. This is done either statically or with a negotiation phase before communication starts.

The communication is initiated by the sender, which sends up to n messages to the multicast group. The active window is initially of size zero, with a starting sequence

number of 0. Each message has the appropriate header information attached, including the source sender's address, the destination multicast group address, and the sequence number of the message, and is copied to the sender's n message buffer. As each message is sent, the sender's packet timer is set to the packet timeout value, and the sender's window is increased by one. The result is a window of up to n outstanding messages, starting at sequence number 0.

The sender then waits for acknowledgments from the receivers. When one does arrive, the sender validates the acknowledgment by checking that the source of the packet is one of the multicast group, and its sequence number falls within the currently active window. If the acknowledgment is a duplicate, that is, it has been received and processed before, then the sender continues waiting for acknowledgments.

Otherwise, the sender processes the acknowledgment by first updating the acknowledgment vector to include the newly received acknowledgment. This is done by setting the validity bit and copying the sequence number of the acknowledgment into the slot of the acknowledgment vector corresponding to the source of the acknowledgment packet. The last acknowledgment received from all receivers by the sender is then calculated as outlined previously. If there is at least one message in the currently active window which has been acknowledged by all receivers, then the calculation of last acknowledgment will result in the sequence number of the most recently sent message in the currently active window to have been acknowledged by all receivers.

The sender can now advance its window by the number of messages acknowledged. First, the upper layer is notified of the successful transmission of the acknowledged

messages. Then, the current window is advanced by recalculating the current window parameters, and resetting the appropriate slots in the acknowledgment vector and message buffer. The sender now checks for messages in its transmit queue, and transmits as many messages as its window will allow. The procedure for transmitting the message is the same as described above, with the exception that sequence numbers continue in sequence.

The sender can also receive a packet timer expiry event while waiting for acknowledgments. A packet timer expiry event occurs when the sender has been waiting for some time for acknowledgments which have not arrived. In this case, the sender assumes that all or some of the messages outstanding in its window have not been properly received, If the retransmission count has not been exceeded, the sender retransmits all the messages in its currently active window, updating the retransmission counts appropriately. The retransmission limit bounds the amount of time required to send messages, however, the protocol fails if the retransmission count is exceeded.

The receivers are initialized with the size of the sender's window, a value of -1 for the last sequence received, etc., and enter a waiting phase. In the waiting phase, the receivers wait for messages from the sender. When a message arrives, the receiver verifies that the message is the next message in the sequence; if it is not, then the message is discarded without acknowledgment. However, if the message is valid, then the receiver updates its counter for messages correctly received but not yet acknowledged and resets the acknowledgment delay timer. When this message counter is equal to the size of the sender's window, the receiver transmits a unicast acknowledgment to the

sender acknowledging all the messages in the sender's window and resets the message counter.

When the acknowledgment delay timer expires, the receiver has been waiting for a message from the sender for some time. The receiver assumes that either a message from the sender has been lost or the receiver's previous acknowledgment has been lost. The receiver sends an acknowledgment to inform the sender of its current status and sets the acknowledgment delay timer once more. All messages correctly received are acknowledged by including the sequence number of the last successfully received message in the acknowledgment packet.

This rather simple block acknowledgment algorithm can reduce the number of acknowledgments per message by a factor of $\frac{1}{n}$ over the previous multicast stop and wait protocol in ideal conditions. We define ideal conditions to be an error-free environment with a steady supply of messages to be transmitted. When n messages are sent to r receivers under these conditions, the previously discussed multicast stop and wait algorithm requires nr acknowledgments. Multicast block acknowledgment, however, only requires r acknowledgments. Since up to n messages can be transmitted without waiting for an acknowledgment, the receiver in a multicast block acknowledgment protocol will attempt to accumulate several messages before sending any reply. It will then send a single acknowledgment message when either no messages have been received for some time or all n messages have been received. Under the ideal condition described earlier, each receiver will send a single acknowledgment for all n messages, resulting in a total of r acknowledgments for n messages. This improvement, espe-

cially when the number of outstanding messages n is related to the number of receivers r, improves the scalability of the protocol and therefore its performance as the number of nodes increases.

2.6.1 Adding Randomly Timed Acknowledgments

The multicast block acknowledgment protocol suffers from the same problem as the multicast stop and wait protocol: acknowledgments from the receivers are transmitted all about the same time, causing excessive collisions in an Ethernet environment. The same solution can be applied for this protocol as well; transmission of all acknowledgments is delayed to some random time within a specified interval, thus reducing the probability that two or more acknowledgments are sent at exactly the same time.

The interval to be used when implementing randomly timed acknowledgments must be carefully chosen; this interval will affect other protocol parameters such as the packet timeout value used by the sender.

2.7 Kaashoek, Tanenbaum et al.

The protocol presented is a simple reliable broadcast [24]. If the group of nodes is determined beforehand, then this broadcast protocol can be trivially modified to work in a multicast environment. We will present it as a multicasting protocol, describing any changes required.

The protocol supports static multicast groups, and multiple senders. It guarantees

globally ordered messages through the use of a sequence node. It uses a positive acknowledgment scheme which is coupled with explicit retransmission requests for quick response to missed messages. The acknowledgment scheme also attempts to reduce the number of separate acknowledgment messages required by piggybacking acknowledgments onto multicast request messages and using implicit acknowledgments for sequences of properly received messages. However, the protocol as described has no fault tolerance, and is particularly vulnerable to a failure of the sequence node. The authors do suggest an extension of the protocol to include an election procedure to replace the single sequencer node on failure. Buffer requirements at both sender and receiver are flexible, since the protocol will use any available resources, and can adapt to differing resource levels dynamically. Of course, the amount of buffer resources available will impact the performance of the protocol.

For the purposes of this description, a distributed system is defined as a group of n processes which communicate via a broadcast network. Each process runs on a separate node, which has a kernel process (for the operating system and networking software) and an application process. Any of the application processes can send messages to all other processes at any instant.

The protocol will make use of a special node called a sequence node to coordinate all multicasting activities. The sequence node has the responsibility of sequencing outgoing multicasts, and ensuring that all nodes receive the messages correctly. All the nodes have the ability to be the sequence node, but only one should act as the sequence node per multicast conversation.

The protocol is initialized by electing a sequence node from among the nodes in the multicast group. All nodes should be informed of this choice as well the multicast address to be used throughout the multicast communication. This is only a slight modification from the broadcast case, where only the sequencer node's address is required by the nodes (since the broadcast address for a network is usually fixed).

A process wishing to send a multicast message will pass the message to the kernel process, which then packages it into a point-to-point multicast request message which is sent to the sequence node. The sequence node receives and buffers the message, assigns it a unique sequence number, and multicasts the message over the network.

Reliability is achieved by having each of the nodes in the network keep a counter of the sequence number of the last message received. If a message is received and its sequence number does not correspond to the sequence number expected, then the node has missed one or more messages in between. The node sends a point-to-point retransmission request message to the sequence node, requesting the missing messages. It also buffers the out of sequence message received, and waits until the missing messages are received before it passes along the messages in the correct sequence to the application process. If no buffers are available, then the node just discards the message, and sends another retransmission request to the sequencer node.

To reduce overhead, the nodes do not acknowledge every multicast message received. Acknowledgments are piggybacked to multicast request messages sent to the sequence node. A number k in the sequence number field of the header of a multicast request message informs the sequence node that all multicast messages up to and

including k have been received by the node. The sequence node keeps a table of the acknowledgments sent by the nodes; if all nodes have acknowledged receiving up to and including some sequence number j, then all messages with sequence numbers less than or equal to j in the sequence node's buffer are deleted. As well, if a node has not sent a multicast request to the sequence node for some time, it sends a dummy multicast message to the sequence node to keep it informed of recent acknowledgments.

If, due to excessive data loss in network, the sequence node has exhausted its buffer space, it stops accepting multicast requests and performs a synchronization protocol utilizing two-phase commit to ensure that all nodes have received all multicast messages. All nodes are sent a phase 1 synchronization message, which informs the nodes of the last sequence number transmitted by the sequence node and instructs them to send up-to-date information on their latest multicast message received. The sequence node then uses this information to retransmit all missing messages to all nodes. Once all nodes have acknowledged their missing messages, the sequence node enters phase 2, deleting all messages in its buffer and informing all nodes that the synchronization is completed. The nodes reply with an acknowledgment, and normal operation is resumed.

Message headers have the following fields:

```
struct header {
    unsigned int type;
    unsigned int sequenceNr;
    unsigned int messageNr;
    unsigned int senderID;
    unsigned int destID;
}
```

The information contained in the header and its interpretation differs slightly based on the type of message. Valid message types are DATA (multicast request), MULTICAST (multicast message), RETRANS (retransmission request), PHASE1 (phase 1 intention message), PHASE2 (phase 2 message), ACK_COMMIT (acknowledgment message for 2 phase commit).

If the message type is DATA, then the sequenceNr field holds the piggybacked acknowledgment from the node to the sequencer; this sequence number is the sequence number of the last consecutive multicast message received by the node. The senderID identifies the node making the multicast request; the destID is the address of the sequencer node to which the multicast request is being sent. The messageNr uniquely identifies messages from this node and is used to discard duplicate multicast requests from the same node.

If the message type is MULTICAST, the sequenceNr field holds the sequence number of the multicast message contained in the body of the message as assigned by the sequencer node. The senderID is the address of the sequencer node; while the destID is a multicast address.

If the message type is RETRANS, then the sequenceNr field holds the sequence number of the message being requested for retransmission. The senderID is the address of the node making the retransmission request; the sequencer requires this address to correctly address the point-to-point message containing the missing message to the requesting node. The destID is the address of the sequencer node to which the retransmission request is directed.

If the message type is PHASE1, it signals the beginning of a synchronization phase whose purpose is to send all outstanding multicast messages to all nodes and flush the buffer of the sequencer node. The sequenceNr field holds the sequence number of the last broadcast message sent by the sequencer node. The senderID is the address of the sequencer node; while the destID is a multicast address.

If the message type is PHASE2, it signals the second phase of the synchronization phase. It informs the nodes that the sequencer has completed the synchronization and normal operation will resume once all nodes have acknowledged the message.

If the message type is ACK_COMMIT, the node is informing the sequence node that it is up to date. The sequencrNr field holds the sequence number of the last consecutive message received by the node; the senderID is the address of the sending node; and the destID is the sequencer node's address.

2.8 Erramilli and Singh

The second protocol from the literature is by Erramilli and Singh [14]. It is a multicast protocol designed for a broadband broadcast network, which is characterized by high bandwidth, low error rates, and relatively low cost buffering. In order to maximize throughput, the authors argue that a protocol designed for this environment should reduce processing time for network transmission/reception at the expense of higher network bandwidth and buffer requirements.

This protocol supports static multicast groups and either single (lecture mode) or

multiple senders (conference mode). It guarantees that messages sent by individual senders will be received in order by all the receivers, but does not provide a consistent global ordering of messages. It uses a negative acknowledgment scheme coupled with timers and status messages to provide reliability. Buffer requirements for the senders' buffer is fixed, while the receivers' buffer requirements are not as clearly defined; receivers may have a fixed buffer pool or may dynamically grow and shrink the buffer in response to changing requirements. The protocol attempts to avoid network congestion by reducing the number of acknowledgments required during periods of intense network activity at the expense of adding overhead status messages during more idle times. As well, the protocol includes some rudimentary flow control mechanisms to prevent buffer overflow in the receivers. A simple fault detection mechanism, which allows all multicast group nodes to detect node failures by monitoring network activity, is provided. Although the protocol does not fail when a node fails, the protocol does not provide any means of reconstructing the thread of communication once a node recovers.

This protocol is differentiated from other multicast protocols mainly by the parameterization of its behaviour into several important variables. These variables control the amount of buffer space, B, required by the sender(s), the number of repetitions, K, and timer intervals, T_1 and T_2, for status/sanity messages, the valid range of sequence numbers, $1 \ldots N$, and timer intervals for flow control, T_3. The correct values to use will depend on the underlying characteristics of the physical network environment, including its error rate as well as expected traffic patterns and network delays.

Changes in these values will modify the behaviour and performance of the protocol; if these values are ill-chosen, the protocol may fail.

In presenting this protocol, the authors make the following assumptions: the multicast group management and control is done by some other protocol; the underlying network environment is a broadband broadcast network with low error rates, high bandwidth, relatively inexpensive buffering, and physical multicasting capabilities; all nodes in the network are relatively equal in terms of processing power and buffering capabilities, thus flow control is not an important issue; and absolute ordering of messages from different senders is not required.

As well, in order to simplify the buffer management at the sender, an implicit back window of B messages is assumed. The last B messages sent are kept in a buffer in case retransmission is required. The increased buffer requirement is balanced by the reduced processing time which would be required in order to determine what messages have been acknowledged by all nodes and therefore can be discarded from the buffer. However, it is important to note that if B is not chosen carefully, then the protocol could fail when a negative acknowledgment (NAK) message arrives requesting retransmission of a message which has been flushed from the buffer.

The protocol assumes that the initialization of the multicast group, including negotiation of the parameterized values N, B, K, T_1, T_2, and T_3, have been performed by some other protocol. In order to keep the implementation of the protocol simple, this initialization will done as previously discussed for other protocols. One node will send the designated multicast group address, as well as the above values (no negoti-

ation), to all nodes using unicast reliable transmission. When all nodes have positively responded, then the multicast protocol can begin.

All senders keep any messages sent in a buffer sendBuf of size B. They also keep the sequence number of the next message to be sent in a variable nextSeqToSend. Note that each sender has an independent stream of data messages to transmit and thus, messages are uniquely identifiable by a combination of the message source and its sequence number. Sequence numbers range from $1 \ldots N$. In order for messages stored in a sender's buffer to be uniquely identifiable, the cardinality of the set of sequence numbers must be greater than the size of the sender's buffer, B.

When an application wishes to send a multicast message to the group, the sender prepares the message by encapsulating it with a header specifying the source address of the transmission, the sequence number of the message, the designated multicast address as the destination address, and a data message type. The prepared message is placed in the sender's buffer sendBuf. If the maximum buffer size of B has been reached, the sender simply replaces the least recently sent message with the newly sent message.

The prepared message is then multicast to the group, the sequence number of the next message, nextSeqToSend is incremented by one modulo N, and the timer T_1 is reset. If the timer T_1 expires before the sender has to send another multicast transmission to the group, then the sender will prepare and send a status/sanity message containing the following information in the header: the source address of the status/sanity message, the multicast group address as the destination, the sequence

number of the next message to be sent, nextSeqToSend, and a message type of status or sanity, as appropriate. The purpose of this message is to inform all receivers of the sequence number of the next message to be transmitted if that message is not yet available, and allows receivers to detect lost messages from a particular sender when the sender is idle and has no data messages pending.

The status/sanity message is sent only during idle periods for the sender, initially up to K times in time intervals of T_1, subsequently falling back to a much longer time interval, T_2. The transmission of sanity/status messages ceases immediately when the sender has data messages pending, and resumes when the sender is once again idle. If a sender never sends any messages, status/sanity messages are sent in intervals of T_2 during the communication. If the messages are being sent in intervals of T_1, the messages are referred to as status messages; if these messages are being sent in intervals of T_2, they are termed sanity messages.

If a sender receives a NAK message from any of the receivers, it searches its buffer, sendBuf, for the requested message, and retransmits that message. The protocol is not clear whether retransmissions are directed to the intended receiver, or multicast to the whole group. A multicast transmission would add processing time at each of the receivers, as each receiver must examine the incoming retransmission to determine if it is required or is to be discarded. Since this protocol attempts to keep processing times to a minimum, retransmissions should probably be unicast transmissions.

Receivers must keep the following information about each of the potential senders: the sequence number of the next expected data message from each sender, the timestamp

of the last received status/sanity message from each sender and a receive buffer recvBuf for each sender. In a multicast group of $M + 1$ nodes, this information is kept in a table of M entries.

If an incoming message is a data message, then the sequence number of the data message is compared to the expected sequence number from the appropriate sender. If this sequence number is less than the expected one, then the newly arrived data message is a duplicate, and can be discarded. If this sequence number is exactly equal to the expected sequence number, then the message is accepted and placed in the receive buffer to be delivered to the application upon request. The expected sequence number for this sender is then incremented by one modulo N.

If, however, the sequence number of the data message is greater than the expected sequence number for this source, then the receiver has lost one or more data messages from this source. The receiver now prepares a negative acknowledgment message (NAK) to be directly sent to the source, with the following information: the source address of the data message as the destination address, the receiver's address as the source address, and the expected sequence number from this sender.

If an incoming message is a status/sanity message, then the timestamp value of the source of the message is updated to reflect the arrival of the status/sanity message. The sequence number of the message is compared to the next expected sequence number for that sender; if the values are equal, then no messages have been lost. If the expected sequence number is less than the value advertised by the status/sanity message, then this receiver has lost one or more messages from the source of the

status/sanity message. The receiver, therefore, prepares and sends a negative acknowledgment message containing the source address of the status/sanity message as the destination, its own address as the source, and the next expected sequence number from the sender.

Node failures are detected by receivers whenever the time interval between the timestamp of the last received status/sanity message and the current time exceeds T_2. Thus, any node failure is detected by all nodes within T_2 of its occurrence.

The protocol achieves a simplified form of flow control by having a node which is in danger of overrunning its buffer send a timed-choke message to all members of the multicast group. This message contains a time interval T3, which is the amount of time the node is requesting to clear up its backlog in order to receive messages again.

Note that all the multicast group nodes are both senders and receivers. This means that the implementation of the protocol must include both the receiver and sender functions at each node. Even in lecture node, nodes which do not send data messages are regularly sending sanity messages in order to inform all nodes of their current status.

2.9 Chang and Maxemchuck

The protocol proposed by Chang and Maxemchuck [6] is described as a broadcast protocol, but it is really a multicast protocol with static multicast groups. Notable features include the ability to reduce control message overhead in high message traffic,

and the ability to continue the communication in the presence of multiple site failures, including failure of the token site, without any loss of committed messages. Chang and Maxemchuck's work actually describes a family of multicast protocols, differentiated by their fault tolerant properties, message latency and site storage requirements.

The protocol only supports static multicast groups, but does support multiple sources. All messages from a single source are ordered as sent, and messages from different sources are totally ordered. The protocol uses a unique combination of positive and negative acknowledgment schemes to balance the advantages of both. It also avoids congesting the network by piggybacking other functions to acknowledgment messages, and reducing the requirement for acknowledgment messages through a negative acknowledgment scheme. Fault tolerant features include frequent rotation of the token site responsibilities, as well as a sub-protocol to construct a fully functional subset of the multicast group in the presence of multiple site failures.

The philosophy of this protocol is to model a multiple source/multiple receiver multicast system as a combination of multiple source/single receiver and single source/multiple receiver subsystems to reduce the complexity of the protocol. The advantages of this division should be clear: the single receiver system simplifies the sequencing of messages from different sources, while a single source/multiple receiver system can be exploited to reduce the number of acknowledgment messages through the use of negative acknowledgments.

The multiple source/single receiver subsystem is implemented by channeling all multicast messages through a single token site, which issues unique timestamps,

thereby totally ordering all messages from multiple sources. This subsystem uses a positive acknowledgment scheme between the source and the token site to explicitly acknowledge each multicast message sent, and also utilizes the acknowledgment message to disseminate to all sites the timestamp generated by the token site. The single source/multiple receiver subsystem is implemented by the interaction between the token site and all the other receivers. The token site stores all messages not yet committed by all sites, and can thus handle all retransmission requests, which are specified in negative acknowledgment messages from the receivers to the token site.

Since the token site now is a single point of failure for the protocol, the responsibilities for the token site are rotated among all sites in the multicast group. Resiliency is added by delaying the passing of messages to the application until it has been received properly by at least a specified number of sites; this number is directly related to the number of failures the protocol can tolerate before failing altogether. The resiliency mechanism is piggybacked onto the token passing mechanism, thus reducing the number of control messages required.

The family of protocols is generated by varying the token passing rate and the resiliency of the protocol. Increasing the token passing rate as a function of the message rate increases the number of control messages required per broadcast, but also decreases each site's storage requirement. Increasing the resiliency of the protocol increases the message latency. Note, however, that increased resiliency does not affect the number of control messages required when a steady stream of multicast messages is available, since the resiliency mechanism is piggybacked onto the token passing

mechanism. It does require additional control messages when no multicast messages are available.

The authors suggest that the best performance is obtained when the token passing rate is one token transfer per multicast message, since this requires only one control message per broadcast when the system is busy and two control messages per broadcast when the system is idle (assuming a resiliency of 1). While the resiliency value does not affect the number of control messages required when the system is busy, it will require additional control messages when the system is idle.

The protocol has two phases, the normal phase and the reformation phase. During the normal phase, messages are multicast, sequenced, and acknowledged. Retransmissions of lost messages and acknowledgments occur, and the token is transferred as needed. During the reformation phase, no messages are multicast, and the sites perform a three phase protocol to construct a new list of operational sites.

All nodes store the next timestamp expected (to detect lost messages), the next sequence number expected from each source (to detect duplicate multicast messages), the version number of the current token list (to detect discrepancies in token lists used), the actual token list (to commit messages and to decide which node should be the next token site), and queues of received messages and acknowledgments.

The protocol sends three types of messages with varying purposes and information. A multicast message is sent from a source to all the sites (but is acknowledged solely by the token site), and contains the address of the source, the sequence number of the message (with respect to the source), the multicast address of the group, and

the data. An acknowledgment message is sent from the token site to all sites, and contains the timestamp (which totally orders this message with respect to all others in the system), the token site address, and the source and sequence number of the message being acknowledged. The acknowledgment message may also pass the token to the next token site, and therefore also carries the next token site address, and the version number of current token list for this purpose. If the token site is not transferred, then these fields contain the current token site and no action is taken. As well, if the token must be passed but no message needs to be acknowledged, the acknowledgment message contains empty fields for all the acknowledgment data. The last type of message sent by the protocol is the confirmation message, which is sent by a new token site to all sites, acknowledging receipt of a token. This message is only sent if, when the token is transferred, the new token site has no messages to be committed or acknowledged after some designated timeout period.

2.9.1 Normal phase

Sources send multicast messages whenever the messages become available. However, each source must wait for an acknowledgment from the token site before sending subsequent multicast messsages. Thus, the communication between each of the sources and the token site follows a stop and wait strategy. Each source orders its own messages with a unique sequence number consisting of the source address and its sequence number. This ordering reflects the transmission order of the multicast messages.

If an acknowlegement for a multicast message is not received after a designated

timeout period, the source retransmits the message up to R times, after which the source decides that the token site has failed, and commences the reformation phase.

The current token site processes incoming messages as follows: If the incoming message is a multicast message, the token site checks to see if this is the next expected message from the message's source. If not, then the message is a duplicate which has already been acknowledged, the message is discarded, but the acknowledgment is repeated. If it is the expected message, then the token site timestamps and buffers the message, and increments its next timestamp value. It prepares the acknowledgment information for the acknowledgment message. It decides whether the token should be transferred (this depends on the token passing rate). If the token is to be transferred, then the token site determines which is the next token site from the current token list and adds that information to the acknowledgment message before transmitting the message to all sites.

If the incoming message is a retransmission request for either a multicast message or an acknowledgment message, then the token site retransmits the required message to the requesting site. Note that, in this protocol, acknowledgments may need to be retransmitted since they not only acknowledge a multicast message but also provide the timestamp information that the receivers require to order multicast messages.

Note that the token site continues to process retransmission requests after it has attempted to pass the token to the next site. This is necessary to allow the next token site to bring its buffers up to date so it can accept the token. It does not, however, acknowledge multicast messages during this time. If the new token site cannot accept

a message during this period, the source will retransmit the message up to R times.

The next token site can accept the token if it has all the messages and acknowledgments received by the previous token site. This is true if the next token site can process the acknowledgment which carries the token. To accept the token, the next token site either acknowledges the next broadcast message, transfers the token to the next site on the token list, or, in the absence of messages to be acknowledged or committed, transmits a confirmation message after a designated timeout period. If the token cannot immediately accept a token transferred to it, the site will request from the previous token site all messages and acknowledgments it requires in order to accept the token. Once all these messages and acknowledgments have been successfully received, the next token site will accept the token as described above.

All sites are actively receiving all broadcast messages, and the timestamped acknowledgments (including the token passing information). Each site attempts to process all incoming transmissions as follows: if the transmission is a multicast message, then the site will check the sequence number of the message for duplication or missed messages. If the message is a duplicate (the sequence number is less than the expected sequence number for that source), then the message is discarded. If messages are missed (the sequence number of the message is greater than the expected sequence number for that source), then the message is discarded, but a retransmission request for the missing message(s) is sent to the current token site.

If the incoming message is an acknowledgment, the site checks the timestamp of the acknowledgment against the next expected timestamp. If this timestamp is less than

the expected timestamp, then the acknowledgment is a duplicate, and is discarded. If the timestamp is equal to the next expected timestamp, then the site checks to see if the message being acknowledged has been received. If it has, then the site orders the message with respect to all other received and acknowledged messages and increments its next expected timestamp. If it hasn't, the site transmits a retransmission request for the missing message to the current token site. The site also checks to see if it can commit any messages at this time. Messages are committed only after the token has been transferred at least L times after a message has been acknowledged. This condition ensures that at least L sites have received the message.

The normal phase continues until a node initiates a reformation, based on the assumption that a failure has occurred. The reformation phase simply regenerates a list of operational sites using a three phase protocol. More details on the reformation process can be found in Maxemchuck and Chang [6].

Chapter 3

The Simulator

3.1 Introduction

Studying the behaviour of multicast protocols in an Ethernet LAN is resource-intensive, requiring a number of computers interconnected in a testbed network with monitoring software. We have developed a simulator that allows study of the behaviour and performance of multicast protocols without a dedicated testbed network. The simulator is designed to allow easy integration of different multicast protocols. We use the simulator to study the two basic multicast protocols described earlier, the stop and wait and block acknowledgment protocols.

The multicast protocols to be studied fall into the logical link (LLC) sublayer of the data-link layer as described in the OSI reference model. These protocols use the services of the media access (MAC) sublayer, which in turn uses the physical layer

services to actually transmit data on the Ethernet bus. The simulator follows this layered approach: the Ethernet LAN is modelled by two layers, the physical and MAC layers, while alternative multicast protocols are modelled as alternate logical link layers. This approach provides the flexibility to develop and use alternate logical link layers in the same simulation framework for comparison.

As in the OSI reference model, the logical link layer (LLC layer) of the simulator is responsible for the sequencing of multicast messages, error detection, and the transmission and retransmission of multicast messages as specified by the protocol. This layer accepts requests for the transmission of messages from higher layer applications which use the multicast protocol and guarantees that multicast messages are reliably received by all recipient nodes. Each multicast protocol included in the simulator is implemented as an alternative logical link layer.

The MAC and physical layers of the simulator model the Ethernet LAN. The media access layer is responsible for all details of access to the Ethernet media, including resolving collisions by implementing the adaptive backoff policy, and the transmission and reception of Ethernet frames. The physical layer of the simulator models the details of the hardware interface and the Ethernet bus, including the transmission and reception of data bits, transmission of the jamming signal when a collision occurs, sensing when the Ethernet media is busy, and detecting a collision.

A diagram of the simulator is found in Figure 3.1. A description of the various layers as implemented in the simulator follows.

3.2 MAC layer

The MAC layer protocol is fairly complex, since it must be able to send and receive frames (encapsulating the data and acknowledgment packets sent/received by the logical link layer) and is driven by requests from both the logical link layer above it and the physical layer below it. The MAC layer protocol is identical for all nodes in the network. The MAC layer will receive any frames accepted by the physical layer, and pass them up to the logical link layer. It will also transmit frames as requested by the logical link layer, retransmitting frames if collisions occur. Retransmissions of frames are governed by the adaptive backoff policy, which specifies the minimum time intervals at which retransmissions may occur.

The MAC layer takes packets placed in its transmit queue and transmits them one at a time. Transmission at the MAC layer merely passes the packet to the physical layer for actual transmission on the medium; however, the MAC layer is entrusted with the details of retransmissions should a collision occur. The physical layer sends every MAC layer one of three events for any relevant activity occurring on the medium: success, receive, or collision. A success event informs the MAC layer that the packet it passed to the physical layer for transmission was indeed successfully transmitted. A collision event informs the MAC layer that the packet it sent was involved in a collision. A receive event informs the MAC layer that a frame destined for it was received. The physical layer does not signal the MAC layer of a node if the frame "on the wire" is not destined for it; thus, the number of receive events processed by the MAC layer is the same as the number of interrupts which must be serviced by an

actual Ethernet adapter card.

In the case of a collision event, the MAC layer must implement the adaptive exponential backoff algorithm as specified by the Ethernet standard. The algorithm works in the following manner: for the ith retransmission attempt, where i is less than 16 attempts, select a random number between 0 and $2^{\min(i,10)}$. This integer is the number of slot times (512 bit times) the MAC layer is required to wait before it may retransmit again. The MAC layer implements this wait by passing a transmit request to the physical layer with a delayed start time. Once the 16th attempt has been unsuccessful, the MAC layer is required to discard the frame and continue.

3.3 Physical Layer

The physical layer of the simulator models the behaviour of the Ethernet hardware and bus; however, many of the details of this behaviour does not affect the latency of data/acknowledgment packets generated by the logical link layer, and therefore do not affect any of the parameters we wish to measure. In the interest of simplicity, the physical layer model encompasses only those aspects which affect these parameters, mainly collision detection, and frame transmittal times.

Some assumptions are made in order to simplify the model. Collisions are always the maximum collision window, 32 bit times. In an actual LAN, the collision jamming signal is only sent as long as necessary for the colliding nodes to recognize the collision; this depends on the distance between the colliding nodes. We assume that

the small fixed interframe gap which exists between the end of a transmission and the first possible transmission time is zero, instead of its usual small value. Since this parameter is fixed, it will not affect relative timings, and only minimally affect comparisons with actual timings.

The physical layer of the simulator examines all of the physical layer requests from each communicating node's MAC layer, and makes a decision to either transmit a frame, detect a collision, or remain idle, adjusting the global time variable accordingly. Each physical layer request will have a time associated with it; this is the time the MAC layer requested the transmission. The simulator examines the physical layer request queue, which is ordered by the requested transmission time, for impending requests. If two or more nodes request transmission at the same time, then the simulator "detects" a collision and places collision events in the event queues of the MAC layers of the nodes involved in the collision. The global simulation time is advanced by the maximum collision window, 32 bit times. If only one node requests transmission, then the first request is "transmitted" by sending a receive event (containing the packet) to the MAC layers of the node(s) receiving the packet, and a success event to the MAC layer of the node transmitting the packet. The simulator then advances the simulation time by the time required to transmit the packet. If there are no impending requests, nothing is done and the medium is deemed to be "idle".

Whenever the simulation time is advanced, all the requests in the physical layer request queue whose time is less than the newly advanced time is adjusted to the new time. This behaviour is meant to simulate the carrier sensing ability of the Ethernet

hardware interface; all Ethernet adapters can "sense" when the channel is busy with another transmission, and defer any transmissions to the first "idle" time. Note that the carrier sensing ability cannot prevent collisions; nodes may attempt to transmit simultaneously. In actual practice, a nonzero signal propagation delay extends the amount of time a collision may occur to the amount of time required for the signal to propagate from one transmitting node to another, although we will not be including this window into our calculations.

3.4 Error Model

Errors which may occur in an Ethernet network environment include packets dropped by an interface on transmittal or reception of a frame (possibly due to buffer overflow or the inability of the interface to keep up with full speed, large volume traffic), and garbled messages on the media (due to surrounding interference, improper wiring and termination, and/or the inability to detect collisions within the allotted collision window). For our purposes, the results are the same: a lost packet. Thus, we base our error model on the former, and randomly dropped packets on transmission/reception at individual interfaces. We also assume that the error rates on both transmission and reception are equal, that is, it is equally likely that an interface will drop an outgoing packet as an incoming one.

The implementation of the error model is done at the MAC layer, on receipt or transmittal of a packet, by randomly dropping packets at an user-configurable error

rate. Thus, this model includes errors by the sending node which prevent the packet from being transmitted, as well as errors by the receiving node in which a packet is dropped before being processed by the MAC layer. In a multicast environment, this implies that a multicast packet may be dropped in one of two ways: either by the sending node, resulting in none of the multicast group ever receiving the packet, or by one or more receiving nodes, resulting in a partially successful multicast transmission.

3.5 Performance Statistics

An important aspect of the simulator is how it measures the performance of the protocols it simulates. The simulator produces the following data for each packet sent:

- unique packet ID, used for tracing a packet throughout the simulation process
- size of the data packet, used to calculate data throughput.
- current window size, used to calculate average latency per message for the block acknowledgment protocol.
- time the packet was sent by the LLC layer.
- time the packet was determined to have been reliably received by the LLC layer sender. Note that in the block acknowledgment protocol, messages are acknowledged in groups, and therefore the calculation of latency per message is an

average based on the current window size and the latency incurred in transmitting the entire group.

- calculated total latency, based on the two times given above.
- calculated average latency, calculated as total latency averaged over the current window size.
- number of LLC layer retransmissions.
- an indication of failure of the protocol due to excessive retransmissions; set to 1 if number of retransmissions was exceeded, 0 otherwise.

This data is provided in an output file, formatted as a header detailing the values of the simulation parameters for each simulation pass, followed by the data collected in that pass. A sample of the output is provided below:

```
Pkts=100 Recs=3 Data=1518 Ack=64 FailRate=0 Reps=1 Proto=BAP BAPWin=1 BAPPktT=115680
BAPAckT=95000 BAPrand=6000 Date=Sun Aug  4 12:01:27 1996
    1      1518      1            0        15780        15780     15780.00      0  0
    2      1518      1        15780        33893        18113     18113.00      0  0
    3      1518      1        33893        52491        18598     18598.00      0  0
    4      1518      1        52491        69372        16881     16881.00      0  0
    5      1518      1        69372        84225        14853     14853.00      0  0
    6      1518      1        84225        99785        15560     15560.00      0  0
    7      1518      1        99785       117367        17582     17582.00      0  0
    8      1518      1       117367       135567        18200     18200.00      0  0
    9      1518      1       135567       153258        17691     17691.00      0  0
   10      1518      1       153258       167145        13887     13887.00      0  0
   ...
```

3.6 Simulation Parameters

A simulation is performed by executing the simulator with an input command file which specifies the user-configurable parameters of the simulation. The simulator produces an output file, which contains some header information for each simulation pass followed by the timing data generated by the simulator. The simulator can also produce a logging file on request; the logging file contains a play-by-play description of the simulation as it executes, and can be used to explain protocol behaviour.

The command file allows users to specify many general and protocol-specific parameters of the simulation. A sample command file is given below:

```
# Sample Configuration File for Simulation Program
#
# Simulation is run with STARTRECS receivers up to ENDRECS receivers, using an increment
# of INCRRECS each time; using STARTPKTS packets up to ENDPKTS packets using an increment
# of INCRPKTS each time; with a failure rate of STARTFAILRT up to ENDFAILRT, using
# an increment of INCRFAILRT each time, etc.
#
# Syntax:
# # as first character in line - comment line
# VARIABLE = value - one space before and after the "="
#
# Name of log file; if empty, no logfile is produced
LOGFILE =
#
# Name of statistics file; default name is statsfile.
STATSFILE = bapwin.dat
#
# Protocol simulated: valid values are
# SAW (stop and wait)
# BAP (block acknowledgment).
#
PROTO = BAP
#
# Start of number of receivers interval
#     valid range 2 - 100
#    default:
```

```
STARTRECS = 3
#
# End of number of receivers interval
#    valid range 2 - 100
#    default:
ENDRECS = 20
#
# Increment for number of receivers interval
#    valid range 1 - 98
#    default:
INCRRECS = 1
#
# Size of data packets
#    valid range (64 - 1518 bytes)
#    default = 1518
#
STARTDATAPKTSIZE = 1518
ENDDATAPKTSIZE = 1518
INCRDATAPKTSIZE = 1
#
# Size of ack packets
#    valid range (64 - 1518 bytes)
#    default = 64
#
STARTACKPKTSIZE = 64
ENDACKPKTSIZE = 64
INCRACKPKTSIZE = 1
# SAW timeout semantics
#    Valid Values: constant, formula
#
SAWTIMEOUTMETHOD = formula
#SAWTIMEOUTMETHOD = constant
#
# SAW timeout values
#
STARTSAWTIMEOUT = 20000
ENDSAWTIMEOUT = 40000
INCRSAWTIMEOUT = 5000
#
# SAW Random Wait values
#
STARTSAWRANDWAIT = 6000
ENDSAWRANDWAIT = 6000
INCRSAWRANDWAIT = 500
#
```

```
# SAW retransmit limits
#
STARTSAWRETRANSLIM = 10
ENDSAWRETRANSLIM = 10
INCRSAWRETRANSLIM = 1
#
# SAW retransmission method
#     unicast, multicast
#
SAWRETRANS = multicast
#SAWRETRANS = unicast
#
# BAP timeout semantics
#    Valid Values: constant, formula
#
BAPPKTTIMEOUTMETHOD = formula
#BAPPKTTIMEOUTMETHOD = constant
#
# BAP pkt timeout values
#
STARTBAPPKTTIMEOUT = 1000
ENDBAPPKTTIMEOUT = 1000
INCRBAPPKTTIMEOUT = 100
#
# BAP ack delay timeout values
#
STARTBAPACKTIMEOUT = 95000
ENDBAPACKTIMEOUT = 95000
INCRBAPACKTIMEOUT = 10000
#
# BAP window sizes
#
STARTBAPWIN = 1
ENDBAPWIN = 20
INCRBAPWIN = 1
#
# BAP Random Wait values
#
STARTBAPRANDWAIT = 6000
ENDBAPRANDWAIT = 6000
INCRBAPRANDWAIT = 500
#
# BAP retransmit limits
#
STARTBAPRETRANSLIM = 20
```

```
ENDBAPRETRANSLIM = 20
INCRBAPRETRANSLIM = 1
#
# Start of number of packets interval
#    valid range 1 - 5000
#    default:
STARTPKTS = 100
#
# End of number of packets interval
#    valid range 1 - 5000
ENDPKTS = 100
#    default:
#
# Increment for number of packets interval
#    valid range 1 - 5000
INCRPKTS = 1
#    default:
#
# Start of data failure rate interval
#    valid range 0 - 100
#    default:
STARTFAILRT = 0
#
# End of data failure rate interval
#    valid range 0 - 100
#    default:
ENDFAILRT = 0
#
# Increment for data failure rate interval
#    valid range 1 - 100
#    default:
INCRFAILRT = 2
#
# Number of repetitions of simulation
#    valid range 1 - 100
#     default: 10
REPETITIONS = 1
```

Many of the parameters allow for a range of values with a specified stepping value; one or more simulation passes are performed over the entire range of parameter values. The data thus produced can then be summarized over the range of values, illustrating

the effect of varying the parameter on protocol performance. Using several ranges for different parameters can result in a wealth of data useful for studying the interactions between parameters on protocol performance and scalability.

General simulation parameters are given below:

- `LOGFILE`: The name of the output log file when logging is enabled. The logging output includes all the actions performed by the simulator while transmitting packets. It is useful to show why a protocol behaves as it does; it is also used in debugging.

- `STATSFILE`: The name of the output data file produced by the simulator.

- `REPETITIONS`: The number of times each simulation pass is performed. Useful when a large number of data points is desired for statistical analysis.

- `STARTRECS`, `ENDRECS`, and `INCRRECS`: These parameters specify the range of receivers used in the simulation.

- `STARTDATAPKTSIZE`, `ENDDATAPKTSIZE`, and `INCRDATAPKTSIZE`: These parameters specify the range of data packet sizes used in the simulation. The values must fall within the range of 64 bytes to 1518 bytes as specified by Ethernet. The parameters are included to analyze the protocol performance under higher level applications with varying data payload requirements. For example, a file transfer application used by a distributed database system can maximize data payloads; while an interactive terminal session application used by a conferencing system could not.

- STARTACKPKTSIZE, ENDACKPKTSIZE, and INCRACKPKTSIZE: These parameters specify the range of acknowledgment packet sizes used in the simulation. Again, the valid range is 64 bytes to 1518 bytes. It is likely that most protocols will use an acknowledgment packet size of 64 bytes, the minimum allowable packet size under Ethernet. However, the flexibility to use large acknowledgment packet sizes is provided for protocols that may want to "piggyback" other information onto the packets.

- STARTFAILRT, ENDFAILRT, and INCRFAILRT: These parameters specify the range of error rates used in the simulation. The error rate is the rate at which packets are randomly dropped by the MAC layer interface of all nodes. The error rate is expressed as an integer percentage.

- PROTO: specifies the protocol to be used in the simulation. Valid values are SAW for the stop and wait protocol, and BAP for the block acknowledgment protocol.

User-configurable parameters for the stop and wait protocol include values for sender timeouts, type of retransmission method, and the use of randomly timed acknowledgments. These are described below:

- STARTSAWTIMEOUT, ENDSAWTIMEOUT, and INCRSAWTIMEOUT: These parameters specify the range of SAW timeout values used in the simulation. and are expressed in bit times. The SAW timeout is the amount of time the sender waits for acknowledgments once a data packet has been sent before deciding that the

packet has been lost and retransmitting. The semantics of the SAW timeout differ when unicast or multicast retransmissions are used.

When the protocol uses unicast retransmissions, this value is used to set one timer per receiver; each timer expires individually if an acknowledgment from the corresponding receiver has not been received by the sender. When the protocol uses multicast retransmissions, only one timer is used; if any of the acknowledgments have not been received before expiry, the sender retransmits the data packet in a multicast transmission.

SAW timeouts occur due to errors, such as dropped packets. In fact, the purpose of the SAW timeout is to detect such errors. If a SAW timeout occurs because the timeout value is not large enough to accommodate the time required for the sender to receive and process acknowledge data packets from receivers, then the resulting "unnecessary" retransmissions can cause congestion in the network, increase message latencies and reduce overall throughput. Simulating the protocol under varying conditions with a range of timeout values can help determine the best timeout value under specific conditions.

- `SAWTIMEOUTMETHOD`: This parameter specifies the method used to calculate the SAW timeout. If the method is `constant`, then the range of timeouts specified in the command file is used as the timeout value; if the method is `formula`, the range of values given in the command file are used as an increment to a base timeout value calculated based on the number of receivers, the transmission time

of data and acknowledgment packets, and the interval used for randomly timed acknowledgments.

- `SAWRETRANS`: This parameter specifies the retransmission method, unicast or multicast, to be used by the protocol. Valid values are `multicast` or `unicast`. Unicast retransmissions are directed at receivers which the sender believes to have lost packets; multicasting retransmissions are sent in multicast packets to all nodes of the multicast group whenever the sender believes that any of the receivers has lost a data packet. The first method reduces the processing overhead on nodes which do not require retransmissions; the second reduces network congestion when more than one node requires a retransmission. The simulator can be used to determine under what circumstances one method results in better throughput value than the other.

- `STARTSAWRETRANSLIM`, `ENDSAWRETRANSLIM`, and `INCRSAWRETRANSLIM`: These parameters specify the range of maximum retransmissions used by the SAW protocol in a simulation. The SAW protocol uses the maximum retransmission value to bound the latency of any one message; it is used to ensure that the protocol terminates under all conditions. By making this parameter user-configurable, the simulator permits the user to determine how much protocol performance is allowed to degrade.

- `STARTSAWRANDWAIT`, `ENDSAWRANDWAIT`, and `INCRSAWRANDWAIT`: These parameters specify the range of random wait intervals used by the SAW protocol in a

simulation, and are expressed in bit times. A receiver randomly chooses the amount of time to delay the transmission of an acknowledgment from within this interval. This interval therefore bounds the amount of delay incurred when using randomly timed acknowledgments. As the interval gets larger, the probability of collisions among the receivers decreases; however, the probability of significant delays increases.

The simulator's implementation of the block acknowledgment protocol allows user configuration of the maximum BAP window size, various timeouts including the sender timeout and receivers' acknowledgment delay timeout, the retransmission limit, and the interval for randomly timed acknowledgments. These are described below:

- `STARTBAPWIN`, `ENDBAPWIN`, and `INCRBAPWIN`: These parameters specify the range of maximum window sizes used by the simulation for the BAP protocol. The maximum window size specifies the maximum number of messages which may be sent by the sender at one time; this value is static over a single simulation pass. As the window size is increased, the number of acknowledgments required is reduced; however, the number of messages which must be retransmitted when an error occurs also rises. The simulator can be used to determine which window size provides the better performance in environments with specified error rates.

- `STARTBAPACKTIMEOUT`, `ENDBAPACKTIMEOUT`, and `INCRBAPACKTIMEOUT`: These parameters specify the range of acknowledgment timeout values used by the simulation for the BAP protocol, and are expressed in bit times. This timeout

value is the amount of time a receiver will wait without receiving additional data packets before deciding that the sender is either not sending more packets or has lost a previously sent acknowledgment packet. The timeout causes the receiver to retransmit an acknowledgment of currently received packets to the sender. This timeout can occur when packets are lost, but also when the sender does not have enough data to transmit to fill its maximum window. Thus, this parameter affects performance even under error-free conditions. The simulator can be used to study the effect of modifying this parameter under varying BAP window sizes and error rates.

- `STARTBAPPKTTIMEOUT`, `ENDBAPPKTTIMEOUT`, and `INCRBAPPKTTIMEOUT`: These parameters specify the range of sender timeout values used by the simulation for the BAP protocol. The BAP sender timeout specifies the amount of time the sender waits for acknowledgments before retransmitting the current window. This parameter affects how quickly the protocol responds to errors. Again, a tradeoff is involved: low values for sender timeout may result in "unnecessary" retransmissions.

- `BAPPKTTIMEOUTMETHOD`: This parameter specifies the method used to calculate the BAP packet timeout. If the method is `constant`, then the range of timeouts specified in the command file is used as the timeout value; if the method is `formula`, the range of values given in the command file are used as an increment to a base timeout value calculated based on the number of receivers, the

transmission time of data and acknowledgment packets, the interval used for randomly timed acknowledgments, and the BAP acknowledgment timeout.

- `STARTBAPRETRANSLIM`, `ENDBAPRETRANSLIM`, and `INCRBAPRETRANSLIM`: These parameters specify the range of maximum retransmissions used by the simulation for the BAP protocol. As in the SAW protocol, the retransmission limit parameter bounds maximum message latency; it is used to ensure the protocol simulation will always terminate.

- `STARTBAPRANDWAIT`, `ENDBAPRANDWAIT`, and `INCRBAPRANDWAIT`: These parameters specify the range of intervals to be used in the simulation for the implementation of randomly timed acknowledgments in the BAP protocol and are expressed in bit times. As in the SAW protocol, a good choice for this interval should reduce the probability of collisions and the resulting network congestion.

3.7 Adding new Protocols

Extending the simulator to include other protocols requires the implementation of new LLC layers implementing the additional protocols. A new LLC layer must use existing data structures to communicate with the lower layer MAC protocols, and follow a few conventions to work within the existing simulation framework.

The simulator uses the convention of a global variable of enumerated type `ProtoType` to distinguish among the different protocols when necessary. Current valid values are `SAW` and `BAP`. For each new protocol to be added, a new value should be chosen to

represent the protocol, and the definition of `ProtoType` modified appropriately.

The suggested overall structure of a LLC layer consists of a receiver and sender module together with supporting functions. For each module, a list of possible event types should be defined. These events should stored in an event queue for processing by the modules. The sender module should also have a transmit queue to store impending transmit requests. Both modules should define a global structure to store information about the current state of the simulation; this may include items like packet buffers, window parameters, etc, but must include an LLC event queue, which is used by the lower MAC layer to inform the LLC layer about events pertaining to it.

The LLC/MAC layer interface consists of the MAC layer function `transmitMAC`, and the LLC event queue. The `transmitMAC` function is used by the LLC layer to pass along transmit requests to the underlying MAC layer, and is declared as follows:

```
void transmitMAC(PktInfo *pkt, SimTime time);
```

This function takes the given packet and request time and adds it to the MAC layer transmit request queue for processing.

The LLC event queue is the mechanism by which the MAC layer informs the upper LLC layer of relevant events, such as the arrival of a data or acknowledgment packet. The type of events supported may differ among sender/receiver modules and protocol implementations. Therefore the MAC layer differentiates between the various protocols when passing event information up to the higher protocol-dependent LLC layer. Thus the implementor is required to add code to the MAC layer module which

adds the LLC event to the event queue referenced by relevant module's LLC global data structure.

The simulator monitors protocol performance by tracking the amount of time required for a sender to transmit a packet and verify its transmission. The simulator uses a global structure for this purpose. The `PktsStatsStruct` structure is defined as follows:

```
typedef struct {
    int pktID;
    int size;
    int fail;
    SimTime llcStart;
    SimTime llcEnd;
    int llcRetrans;
    int llcWin;
} PktStatsStruct;
```

For protocols with single message buffers such as stop and wait, the global variable `pktStats` of type `PktStatsStruct` is used to store the timing data until transmission of the packet has been confirmed by the sender. For protocols with several outstanding messages, a global vector `winStats` of type `PktStatsStruct` is used, containing one record for each outstanding message. The function `printPktStats` is provided to write the timing data to the output data file in the correct format.

It is the responsibility of the LLC layer to initialize the global structures when sending a packet, add the timing data when the transmission has been confirmed, and call the `printPktStats` function to write the results to the output data file.

Timers in all layers (LLC and MAC) are implemented by timer variables, which contain expiry times. The timers are normally stored in the global data structure for

the module which uses it. All timers are checked by the `checkTimers` function every time the simulation time is advanced; only the physical layer and a special utility function which advances the simulation time in idle periods are permitted to change the overall simulation time. The `checkTimers` function should be modified to include all timers required by the new protocol; the code should check if a timer has expired, and place the correct event in the event queue of the appropriate module.

It is recommended that future extensions to the simulator use the logging feature to log the progression of the new protocol through the simulation. This is not only useful for debugging the protocol, but also for verifying that the protocol is working as expected and for closely studying how the protocol behaves in specific circumstances. The following `printf`-like function is provided for this purpose:

```
void plog(char *format, ...);
```

The simulator uses the global variables, `contLLC`, `contMAC`, and `contPhys`, to determine whether a simulation pass has completed. A simulation pass is complete if all these variables are set to 0. The variables are initialized to 0 at the beginning of each iteration through the execution of LLC, MAC and physical layers. The layers are responsible for setting the appropriate value to 1 whenever the layer completes execution with outstanding requests and/or work to be done. The LLC layer, therefore, must determine whether subsequent invocations are required to complete the simulation pass, and set the `contLLC` variable appropriately.

Another set of global control variables are `LLCactivity`, `MACactivity`, and `physActivity`. These variables are set by the layers whenever any activity occurs in

the corresponding module, and are used by the simulation to advance the simulation time whenever all layers are idle. The time is advanced to the time of the first expired timer; the event triggered by the timer will cause activity in at least one of the layers. It is the responsibility of the new LLC layer to set the `LLCactivity` variable whenever the execution of a sender/receiver module produces activity in this layer.

An important feature of the simulator is the configurability of simulation parameters via a command file. Any new protocol added to the simulator is likely to have a few user-configurable parameters. These parameters are specified in the command file, which is parsed by the simulator during initialization. The parsing of the command file is performed by the `parseConfigFile` function, which returns a large data structure containing all the user-configurable parameters. This structure is used by the main function of the simulator to perform simulation passes. Parameters can be added by adding code to parse and check valid values of the parameters in the `parseConfigFile` function, and extending the `simParams` data structure to accomodate the new parameters.

Many of the parameters in the command file specify a range of values for specific protocol variables to assume in separate simulation passes. The main body of the simulator uses looping constructs which execute a simulation pass to implement these ranges. New protocols utilising new parameter ranges will have to add new looping constructs to the main function.

A simulation pass is executed using the function `runSim` which will have to be modified slightly to include calls to the new LLC layer sender/receiver modules. This

function uses the global variable `proto` to determine which LLC layer modules to execute for a simulation run.

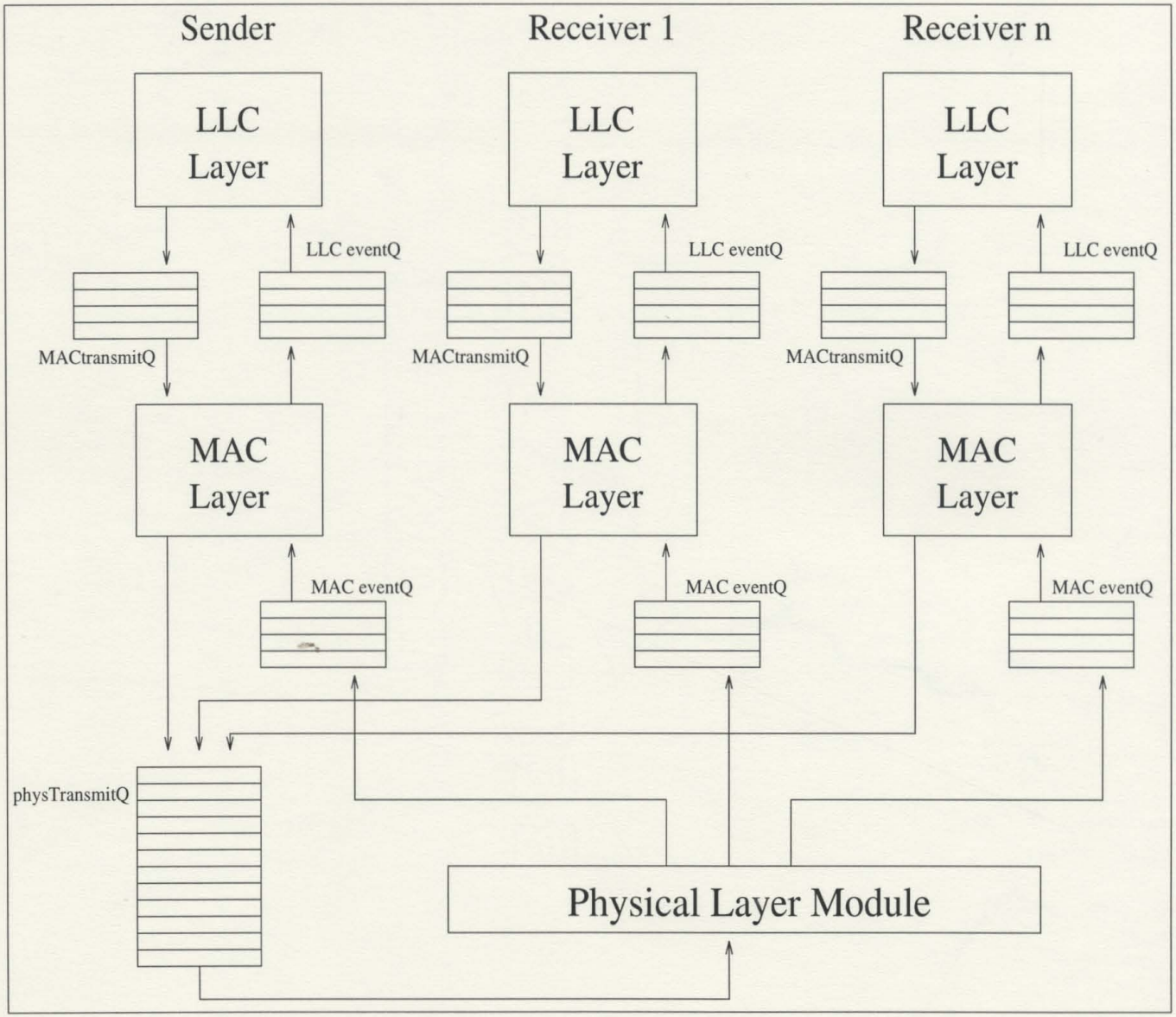

Figure 3.1: Simulator Design

Chapter 4

Simulation Results

The simulator is used to study the workings and performance of the multicast stop and wait and block acknowledgment protocols. Simulation runs using varying parameter values are performed on both protocols and the results analysed and compared. The raw simulation results are processed by programs written using the SAS System [26] [27] of statistical analysis software, which calculate the following performance measures: a throughput ratio relative to an equivalent unicast protocol, an absolute throughput value and an absolute message latency. The programs also produce the summary tables and the plots of relative throughput, absolute throughput and message latency presented in this chapter. A sample program is given in Appendix A. The results of this analysis are presented and the observed behaviour of the protocols is explained.

4.1 Theoretical Bounds

Before discussing the results of the simulations, we first attempt to define theoretical bounds on throughput and message latency values for both the stop and wait and block acknowledgment protocols. We will derive these results for an Ethernet network with no other network traffic.

We have previously defined the relative throughput of a multicast protocol to be the ratio of the absolute throughput of the protocol to the absolute throughput of the equivalent unicast protocol. The absolute throughput of the unicast stop and wait protocol is given as follows:

$$\frac{rd}{rd + ra + \mathcal{S}} \tag{4.1}$$

where r is the number of receivers, d is the transmission time of a data packet, a is the transmission time of an acknowledgment packet, and $\mathcal{S}$ is a synchronisation factor (all times are expressed in bit times). The numerator is the amount of time required to send useful data, while the denominator is the total transmission time, including acknowledgments and synchronisation.

The synchronisation factor is present to allow for transmission delays incurred by the underlying network environment. In the case of Ethernet, this synchronisation factor includes the amount of time spent dealing with media contention and collision resolution. For the unicast stop and wait protocol in ideal (no other network traffic) conditions, no collision resolution or media contention is required, resulting in a syn-

chronisation factor of zero. The absolute throughput for the unicast stop and wait protocol is, therefore,

$$\frac{rd}{rd + ra} \tag{4.2}$$

Despite the absence of other network traffic, the synchronisation factor when using multicast stop and wait protocol is significant. Several nodes may attempt to acknowledge multicast packets simultaneously, causing one or more collisions. (One of the factors which may skew the times the receivers attempt to send acknowledgments is variances in processing times among receivers; however, these variances are unlikely to eliminate collisions entirely.) The synchronisation factor used in the calculation of the absolute throughput of the multicast stop and wait protocol is, therefore, the amount of time spent in collision resolution. This is highly dependent on the Ethernet adaptive backoff algorithm and is a function of the number of nodes attempting to simultaneously access the network medium.

The absolute throughput of the multicast stop and wait protocol is given by:

$$\frac{rd}{d + ra + \mathcal{C}(r)} \tag{4.3}$$

where r, d, and a are as before and $\mathcal{C}(r)$ is the collision resolution time. Note that only one data packet is sent, as opposed to the r packets sent using the unicast protocol.

Using equations 4.2 and 4.3, we obtain the following equation for the relative throughput of the multicast stop and wait protocol with respect to the unicast stop and wait protocol for a given number of receivers r:

$$\frac{rd + ra}{d + ra + \mathcal{C}(r)} \tag{4.4}$$

To calculate the upper bound for this relative throughput value, we first observe that the best throughput will be achieved when the collision resolution function $\mathcal{C}(r) = 0$, meaning that all packets are transmitted one after another, perfectly scheduled, with no collisions or unnecessary delays. Thus, we arrive at the following equation for the upper bound of the relative throughput of the multicast stop and wait protocol for a given number of receivers r:

$$\frac{rd + ra}{d + ra} \tag{4.5}$$

Note that this equation is dependent on the number of receivers r; increasing the number of receivers will increase the upper bound. However, this increase in relative thoroughput does not continue indefinitely, as Figure 4.1, which plots upper bound relative throughput versus number of receivers for a data packet size of 1518 bytes and an acknowledgment packet size of 64 bytes, clearly shows. We find the maximum relative throughput value for multicast stop and wait by calculating the limit of equation 4.5 as $r \to \infty$:

$$\lim_{r \to \infty} \frac{rd + ra}{d + ra} \tag{4.6}$$

Solving this limit:

$$\frac{d+a}{a} \tag{4.7}$$

This result is significant because it places a limit on the improvements which can be achieved by merely increasing the number of receivers involved in a multicast communication using the multicast stop and wait protocol.

Thus, the maximum relative throughput of the multicast stop and wait protocol with a data packet size of 1518 bytes and an acknowledgment packet size of 64 bytes is:

$$\frac{1518+64}{64} = 24.719 \tag{4.8}$$

The block acknowledgment protocol increases throughput by requiring only one acknowledgment for a block of messages, thus reducing overhead due to acknowledgments. In a multicast environment where the number of acknowledgments required is directly related to the number of receivers, this improvement can be significant. We compare the throughput performance of the multicast block acknowledgment protocol to the equivalent unicast block acknowledgment protocol. We assume that we always have enough data available to transmit entire blocks.

We calculate the throughput of the unicast block acknowledgment protocol as follows:

$$\frac{rnd}{rnd+ra+\mathcal{S}} \tag{4.9}$$

where r, d, a and $\mathcal{S}$ are as defined previously, and n is the size of the window.

Under ideal conditions (no other network traffic) in an Ethernet environment, the synchronisation factor is zero, and the throughput of the unicast block acknowledgment protocol is given by equation 4.10:

$$\frac{rnd}{rnd + ra} \tag{4.10}$$

The throughput performance of the multicast block acknowledgment protocol, like the multicast stop and wait protocol, must include the effect of collision resolution, even in ideal (no other network traffic) conditions. The absolute throughput of the multicast block acknowledgment protocol is given by:

$$\frac{rnd}{nd + ra + \mathcal{C}(r)} \tag{4.11}$$

where $\mathcal{C}(r)$ is the collision resolution time, which is a function of the number of receivers r.

Using equations 4.10 and 4.11, we derive the following equation for the relative throughput of the multicast block acknowledgment protocol with respect to the unicast block acknowledgment protocol:

$$\frac{rnd + ra}{nd + ra + \mathcal{C}(r)} \tag{4.12}$$

We note that the upper bound of this equation occurs when $\mathcal{C}(r) = 0$. Thus the upper bound relative throughput of the multicast block acknowledgment protocol is:

$$\frac{rnd + ra}{nd + ra} \tag{4.13}$$

Calculating the limit of equation 4.13 as $r \to \infty$ will result in an equation for the maximum relative throughput achievable by the multicast block acknowledgment protocol for a given window size n:

$$\frac{nd + a}{a} \tag{4.14}$$

Thus, the upper bound on the relative throughput of the multicast block acknowledgment protocol with a window size of 5, a data packet size of 1518 bytes and an acknowledgment packet size of 64 bytes is:

$$\frac{5 \times 1518 + 64}{64} = 119.594 \tag{4.15}$$

Similarly, the upper bound on the relative throughput of the multicast block acknowledgment protocol with the same parameters as above but a window size of 10 is:

$$\frac{10 \times 1518 + 64}{64} = 238.187 \tag{4.16}$$

Increasing the window size to 15 results in an upper bound relative throughput of:

$$\frac{15 \times 1518 + 64}{64} = 356.781 \tag{4.17}$$

These asymptotes are clearly visible in Figure 4.2 which plots upper bound throughput against number of receivers for window sizes 5, 10, and 15.

When we attempt to define an upper bound for message latency, we find that the statistical nature of collision resolution of the Ethernet environment on which these multicast protocols are based prevents such a calculation. When two or more nodes in an Ethernet collide, both nodes must randomly choose to retransmit within a specified time interval, which grows exponentially with successive collisions. The likelihood of successive collisions is, therefore, reduced significantly with each collision, but is never eliminated. It is possible, although unlikely, that a packet will never be transmitted due to collisions. Thus, since the underlying Ethernet environment cannot guarantee packet delivery within a specific time frame, the multicast protocols using Ethernet as the network medium cannot guarantee it either. Since none of the multicast protocols can provide an upper bound of message latency, they are not the best choice for applications with stringent real-time requirements, such as process control. Note, however, that this applies to any protocol using Ethernet as the underlying network, not just multicast protocols.

Although not as useful, we can arrive at a lower bound for absolute message latency for both the stop and wait and block acknowledgment protocols. The lower bound can be used as a check for the simulation results, and provides some basis for comparison to average latency results. It can be used to calculate how much of the average latency is due to overhead such as collision resolution and retransmissions.

For the multicast stop and wait protocol, the lower bound for message latency is

calculated based on the minimum time required to send one multicast message and all associated acknowledgments. The following equation represents a lower bound for message latency:

$$d + ra \tag{4.18}$$

where d is the transmission time of a data packet, r is the number of receivers, and a is the transmission time of an acknowledgment packet.

In the case of the multicast block acknowledgment protocol, however, messages are sent and acknowledged as a block, thus the latency of a single message is the latency of the entire block of messages. The lower bound latency is therefore calculated for a block of messages as follows:

$$dn + ra \tag{4.19}$$

where n is the maximum window size permitted. Note that this calculation for lower bound latency assumes that the multicast block acknowledgment protocol always has enough data available to fill a window; otherwise the lower bound latency is dependent on the current size of the window.

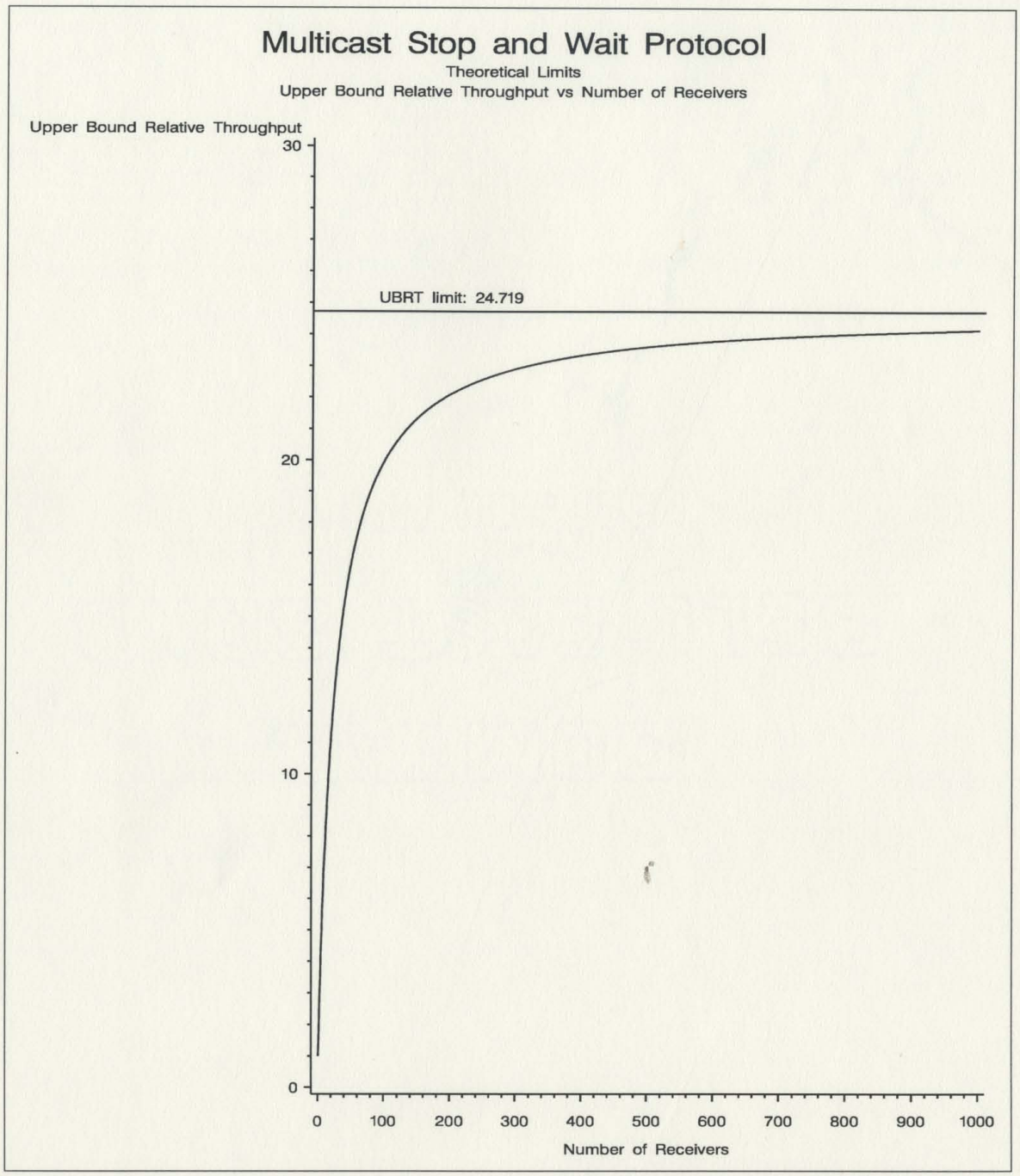

Figure 4.1: MSAW Performance: Upper Bound Relative Throughput

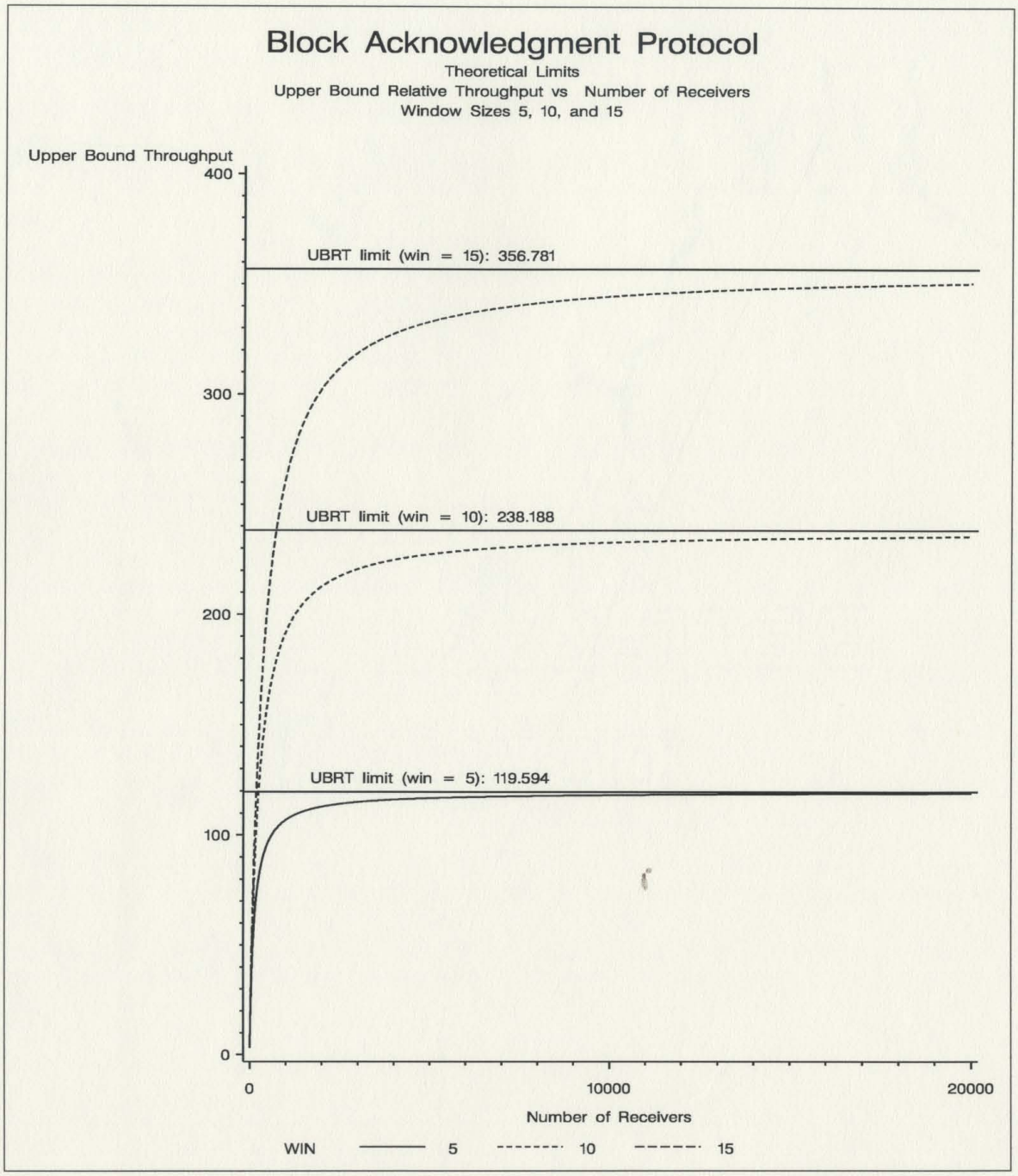

Figure 4.2: MBAP Performance: Upper Bound Relative Throughput for Window Sizes 5, 10, and 15

4.2 Stop and Wait Protocol Performance

A major motivation for the design of multicast protocols is to utilize the hardware capabilities of the underlying network technology to increase the rate at which identical data can be transmitted to multiple sites. Thus, it is reasonable to expect that a multicast protocol should perform better than the equivalent multiple point-to-point transmissions. We test this premise using the two simple multicast protocols we have studied, the stop and wait protocol and the block acknowledgment protocol.

In general, we perform simulations by sending 500 packets to 3 to 60 receivers, using a data packet size of 1518 bytes, an acknowledgment packet size of 64 bytes, multicast retransmissions, and no randomly timed acknowledgments. The Ethernet environment is simulated with no errors. Both the stop and wait and block acknowledgment sender timeout values are chosen deliberately high, to prevent premature timeouts at large number of receivers; we do not want to consider the effects of retransmissions at this stage. Changes from this general scheme are noted as appropriate.

We first simulate the multicast stop and wait protocol. The data file produced by the simulator is then processed through a SAS program, which produced Figure 4.4, Figure 4.3, and Table 4.1. Figure 4.4 plots absolute message latency against number of receivers; the entire range of message latency values for a given number of receivers is plotted as a line with the boxed area representing the 25th to the 75th percentile ranges. The dotted line plots the lower bound message latency values as derived in Section 4.1. Figure 4.3 plots relative throughput against number of receivers; the

entire range of relative throughput values for a given number of receivers is indicated on the plot, with the joined line representing the mean relative throughput values. The upper bound relative throughput values are plotted as a dotted line. Table 4.1 summarizes the simulation results, and includes the derived lower bound message latency and upper bound relative throughput values.

The results presented confirm that the multicast stop and wait protocol performs better than the equivalent unicast stop and wait protocol on average. The mean relative throughput results are better for the entire range of number of receivers simulated, from a low of 3.504 times better at $r = 5$ to a high of 6.191 times better at $r = 50$ than those calculated for the equivalent unicast stop and wait protocol.

The simulated multicast stop and wait protocol results also follow the same general trend as the upper bound theoretical results: rapid improvement eventually levels at an asymptote. However, the simulated multicast stop and wait protocol reaches its asymptote much earlier and at a much lower relative throughput value than the upper bound theoretical results; levelling at 20 receivers and a relative throughput value of approximately 6.0, while the theoretical results indicate a maximum relative throughput of 24.719 at $r > 1000$.

The results also show large variability in message latency values as the number of receivers increases, due mainly to large increases in maximum message latency values. However, minimum message latency values follow the lower bound message latency values fairly closely. Despite this variability, 50% of the message latency values fall within a small range, as indicated by the plotted 25th and 75th percentile message

latency values.

The better performance of multicast stop and wait is due to the fact that only one data transmission, as opposed to n separate transmissions, must be sent for data to be delivered. Thus, the amount of "effective" data sent by the multicast packet increases linearly with respect to the number of receivers. Moreover, the time required to send the "effective" data using the stop and wait protocol does not include the amount of time required for multiple transmissions, as it does when using the equivalent unicast protocol. At first glance, it might be reasonable to expect the relative throughput of the stop and wait protocol to grow linearly with respect to the number of receivers. However, the upper bound theoretical results show this not to be the case; the increase in the number of acknowledgments required as the number of receivers increases tempers the gains realized by the increase of "effective" data, eventually reaching a maximum asymptotic value of 24.719. The simulation results also exhibit this pattern, albeit at a much lower relative throughput value of approximately 6.

The lower throughput values achieved by the multicast stop and wait protocol from the upper bound theoretical results are the effect of the time required to resolve collision among receivers sending acknowledgments. As the number of receivers increases, the number of acknowledgments required also increases, causing the probability of multiple collisions to rise. Moreover, the amount of time required to resolve these collisions also grows significantly. This effect is clearly demonstrated by the increase in maximum message latency as the number of receivers grows.

We conclude that the simple multicast stop and wait protocol has satisfactory performance within the range of receivers simulated, but that improvements in throughput taper at about $r = 20$.

The results also showed large variability in the maximum and minimum throughput values throughout the range of receivers. This large variability can be explained by the statistical nature of collision resolution in an Ethernet environment. The randomness of this process signifies that the time required to resolve collisions among multiple nodes can vary greatly from one instance to another, thus causing large variabilities in message latencies and relative throughput calculations. These variabilities are most noticeable among larger numbers of receivers, where the effect of collision resolution on message latencies is quite pronounced. Moreover, the statistical nature of collision resolution also signifies that, on average, collisions will be resolved within a reasonable time frame, with extreme values occurring more rarely. Figure 4.4 demonstrates this effect quite clearly, as 50% of the message latency values fall within a relatively small range.

The above simulation was done using a fixed data packet size of 1518 bytes, the largest MTU allowed by Ethernet. We next investigate the effect of smaller data packet sizes on the protocol's performance. We can use the upper bound result derived in Section 4.1 to calculate what the maximum expected relative throughput will be for various data packet sizes. For example, for the simple stop and wait protocol, the upper bound on relative throughput when using a data packet size of 1436 is 23.438; for a data packet size of 848, it is 14.25, and for a data packet size of 64, the upper

bound drops to just 2.0. Given that the theoretical upper bound throughput declines significantly as the packer sizes get smaller, we would expect the protocol's simulated throughput to drop dramatically as the data packet size decreases. To determine how much the size of the packets transmitted affects the protocol performance, we performed simulation runs as before but varying the data packet sizes to be 64, 848, and 1436 bytes.

Plots of relative throughput versus number of receivers for data packet sizes of 64, 848, and 1436 bytes are given in Figures 4.5, 4.6, and 4.7. Summary results of relative and absolute throughput are given in Table 4.2. The summary results include the upper bound relative (UBRT) and absolute throughput (UBAT) calculated for each r.

The results show a marked decrease in performance as the data packet size decreases. For data packet size of 1436 bytes, the throughput reaches a high of 5.910 for $r = 50$, but for a data packet size of 848, the throughput reaches a high of 3.849 for $r = 25$, subsequently dropping to a maximum throughput of 0.869 at $r = 5$ for the minimum data packet size of 64 bytes. It is interesting to note that the maximum throughput value is found at lower number of receivers as the data packet sizes decrease.

The overall decline in throughput when transmitting smaller data packets is easily explained by the decline in "effective" data transmitted, while the overhead, such as transmissions of acknowledgments and collision resolution, stays relatively constant. However, another effect is also present as the number of receivers increase: the colli-

sion resolution time, growing due to the rising collision rate as the number of receivers increases, dominates over the effect of the linear increases in "effective" data transmitted much earlier. This is due to the fact that the multiplicative factor for the increase in "effective" data as the number of receivers increase is the size of the data packet itself; as the data packet size decreases, the increase in "effective" data is affected accordingly.

We can conclude that the multicast stop and wait protocol's performance declines significantly when transmitting smaller packets; its performance as the number of receivers increases also degrades much earlier for smaller data packets. In fact, for the minimum data packet size of 64 bytes and receivers $r > 3$, the multicast stop and wait protocol is worse than using multiple unicast transmissions.

Both of the above analyses show that the most significant factor affecting the performance of the stop and wait protocol is the high rate of collisions caused by acknowledgments. Reducing this high rate of collisions should enhance performance significantly. Randomly timed acknowledgments were added to the stop and wait protocol in order to reduce these collisions. The next simulation attempts to measure the effectiveness of this technique. Simulation runs were performed using `SAWrand` values of 12,000 bit times, 22,000 bit times, and 32,000 bit times.

Plots of relative throughput for `SAWrand` values of 12,000, 22,000 and 32,000 bit times are given in Figures 4.8, 4.9 and 4.10; a summary of both relative and absolute throughput for these `SAWrand` values is given in Table 4.3. A summary plot showing the mean relative throughput values for each r with `SAWrand` values of 0, 12,000,

22,000 and 32,000 is given in Figure 4.11. This plot also shows the upper bound relative throughput achievable by the multicast stop and wait protocol.

The results show that using randomly timed acknowledgments improves throughput significantly throughout the range of receivers. Even the smallest `SAWrand` value used, 12,000 bit times, improved throughput for $r = 30$ receivers from 5.999 using the simple stop and wait protocol to 8.625, an increase of 43%. The best throughout achieved was 10.560 for $r = 45$ and a `SAWrand` value of 32,000 bit times which is an increase of 72% over the throughput of 6.156 achieved by the simple multicast stop and wait protocol for $r = 45$.

The results also showed that while the throughput gains increased as the `SAWrand` values increased, where in the range of receivers those gains were most felt also changed as the `SAWrand` values changed. The greater the `SAWrand` value, the greater the number of receivers at which the maximum throughput value was observed.

The overall better performance of the protocol when using randomly timed acknowledgments is explained by the reduced collision rate. Since nodes must now wait for some random period of time within an interval specified by the `SAWrand` parameter before transmitting acknowledgments, the possibility of collisions, and the number of nodes involved in a collision when one occurs, is reduced dramatically. In turn, this reduces the amount of time spent resolving collisions, resulting in a stronger influence from increasing "effective" data rates and less from collision resolution. The overall effect is significant throughput gains throughout the range of receivers.

As noted earlier, the maximum throughput gains appear at larger number of re-

ceivers as the SAWrand value increase. Moreover, the throughput gains noticed at smaller numbers of receivers seem to dwindle as the SAWrand values increase. These effects are clearly shown by Figure 4.11. For example, the throughput at $r = 15$ for a SAWrand value of 12,000 is 7.347; by the time the SAWrand value is 32,000 bit times, the throughput value for $r = 15$ has dropped to 4.444, which is even less than the throughput value of 5.418 achieved by the stop and wait protocol without randomly timed acknowledgments. This reduction in throughput is caused when the increased delay incurred when transmitting randomly timed acknowledgments overshadows the effect of the reduced collision rate. Thus, the usefulness of the SAWrand parameter in improving protocol performance is linked to the number of receivers participating in a multicast communication and must be fine-tuned accordingly. It would be advantageous to explore this relationship further; if a formula for calculating the best SAWrand value for a given number of receivers could be found, it would improve the scalability of the protocol greatly.

Note, however, the SAWrand value is bounded from above by the SAWtimeout value. Increases in the SAWrand value must be matched by corresponding increases in the SAWtimeout value; otherwise, the sender may time out prematurely, causing unnecessary retransmissions and network congestion. It is also not advisable to increase the SAWtimeout value indiscriminately; high SAWtimeout values lengthen the time which must elapse before the sender can determine that a packet has been lost.

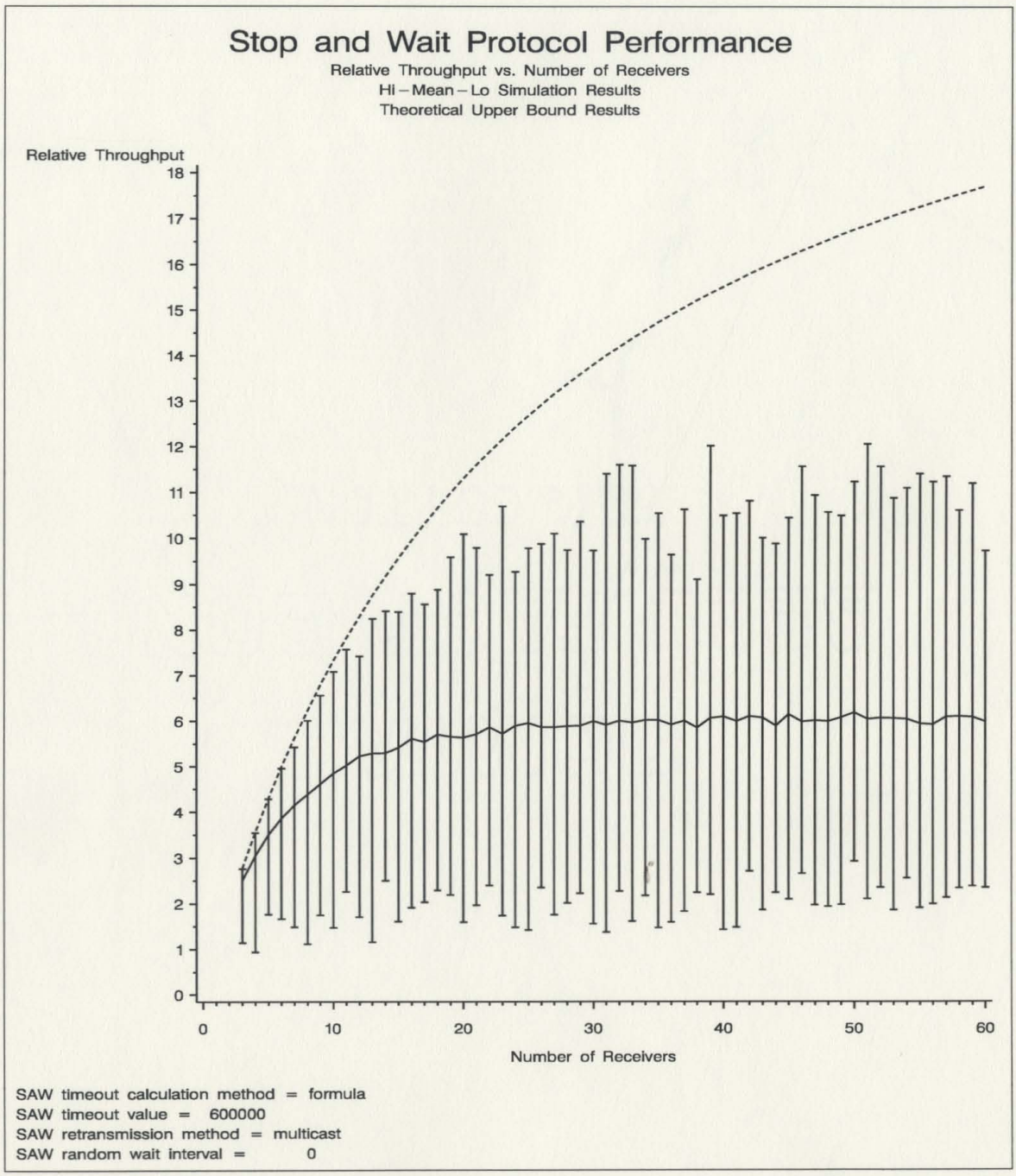

Figure 4.3: MSAW Performance: Relative Throughput (Mean)

Recs	Mean RT	Min RT	Max RT	UBRT	Mean Lat	Min Lat	Max Lat	LBL
5	3.504	1.765	4.285	4.304	18481	14768	35856	14704
10	4.849	1.471	7.081	7.331	27357	17872	86032	17264
15	5.418	1.611	8.397	9.576	37346	22608	117840	19824
20	5.639	1.599	10.096	11.308	48296	25072	158320	22384
25	5.961	1.423	9.785	12.684	56910	32336	222320	24944
30	5.999	1.557	9.729	13.805	68067	39024	243824	27504
35	6.026	1.471	10.547	14.734	80480	42000	301168	30064
40	6.102	1.433	10.494	15.517	89998	48240	353392	32624
45	6.156	2.098	10.441	16.187	99171	54544	271408	35184
50	6.191	2.927	11.226	16.766	108054	56368	216208	37744
55	5.954	1.914	11.404	17.271	126155	61040	363632	40304
60	5.990	2.357	9.727	17.716	135282	78064	322224	42864

Table 4.1: MSAW Performance: Relative Throughput and Message Latency Summary Results

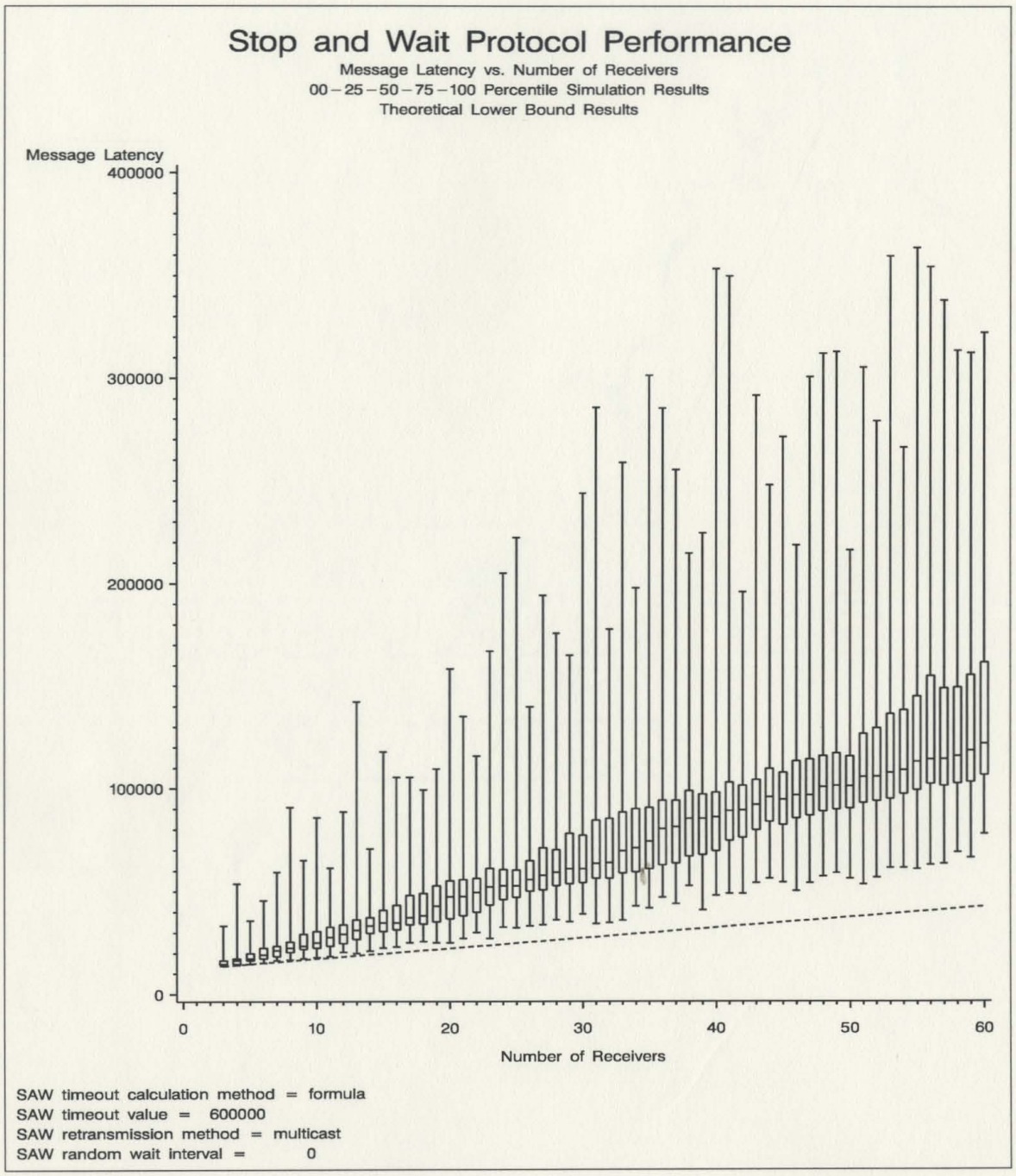

Figure 4.4: MSAW Performance: Message Latency (Percentiles)

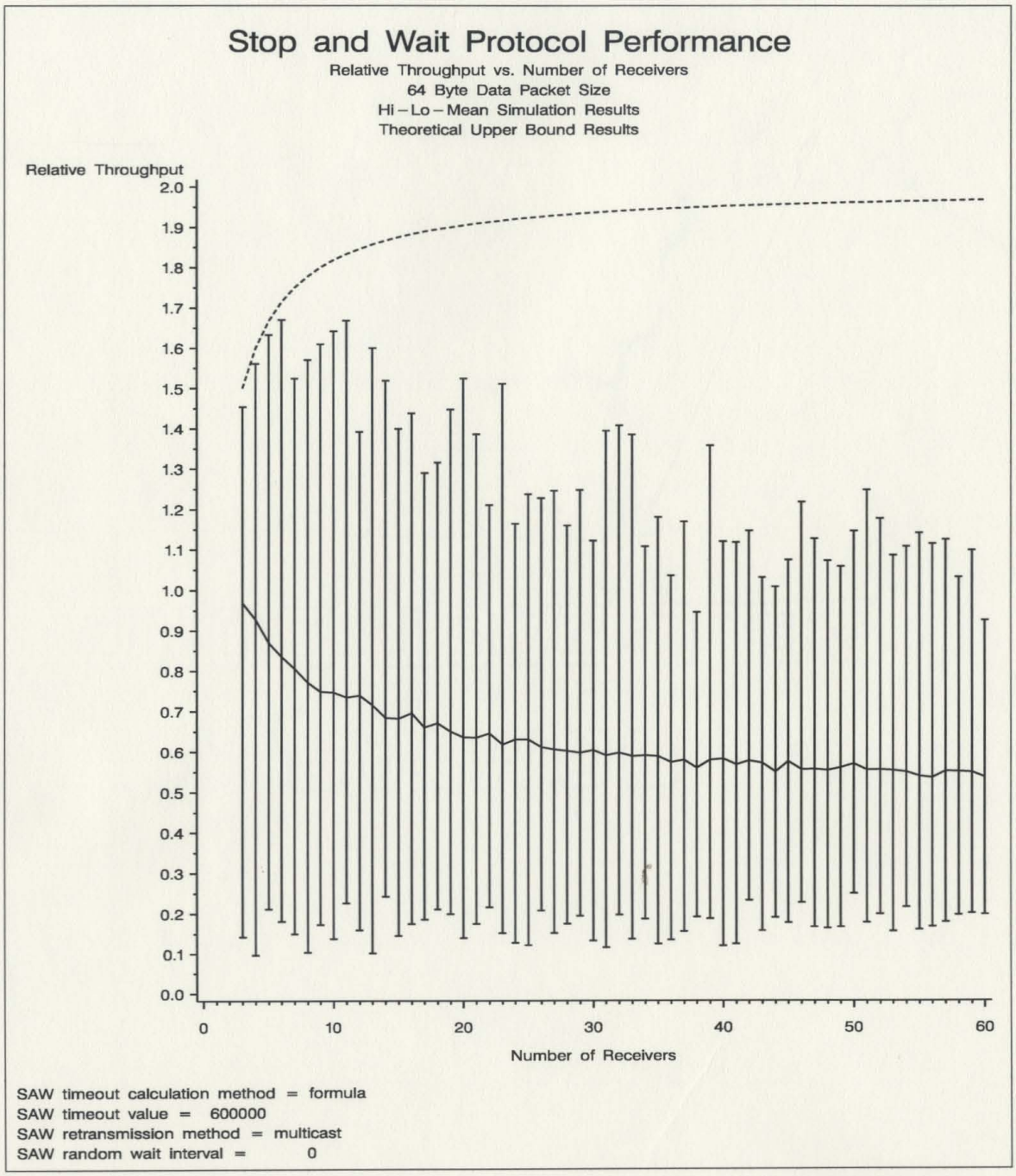

Figure 4.5: MSAW Performance: Relative Throughput for 64-Byte Data (Mean)

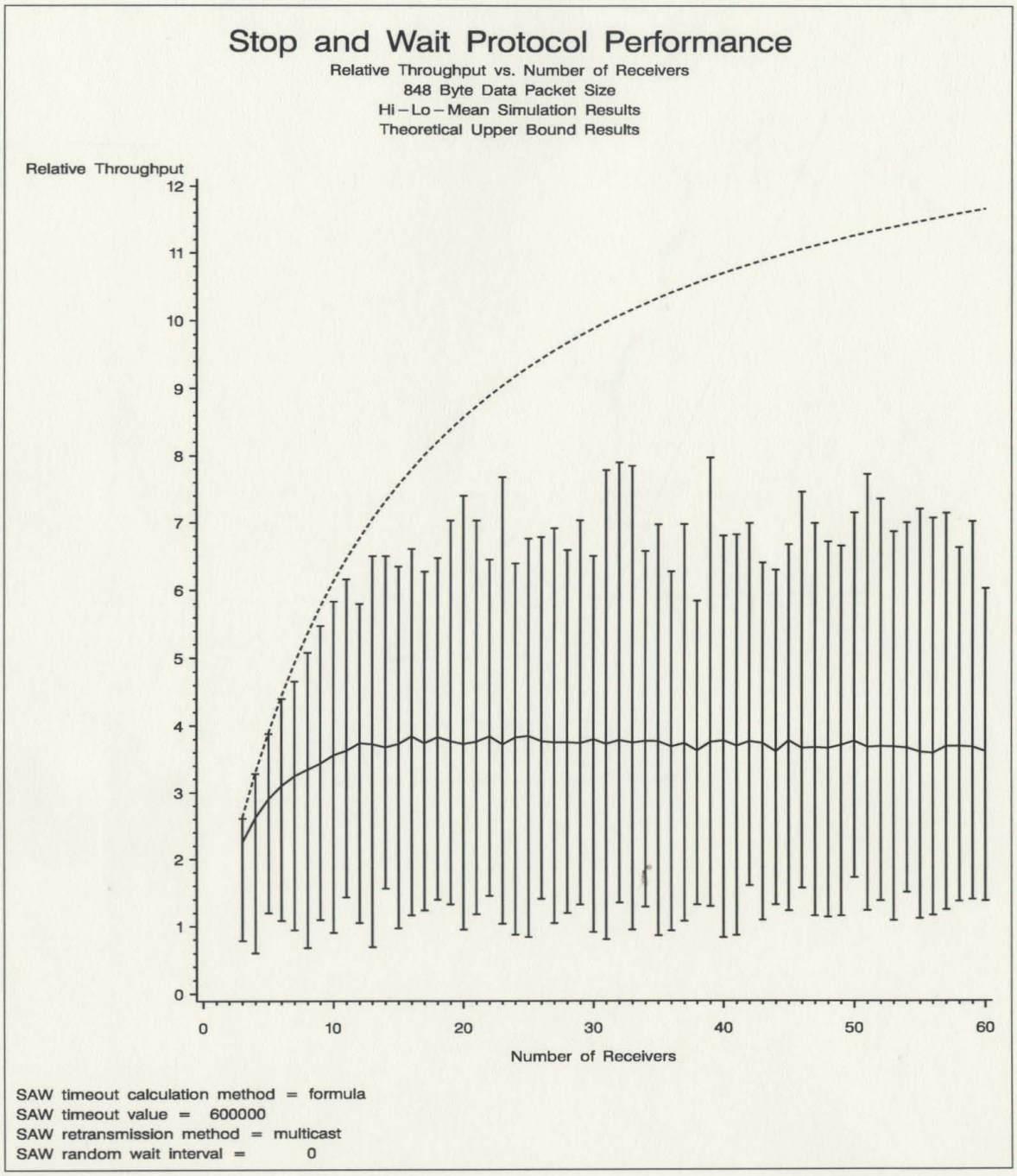

Figure 4.6: MSAW Performance: Relative Throughput for 848-Byte Data (Mean)

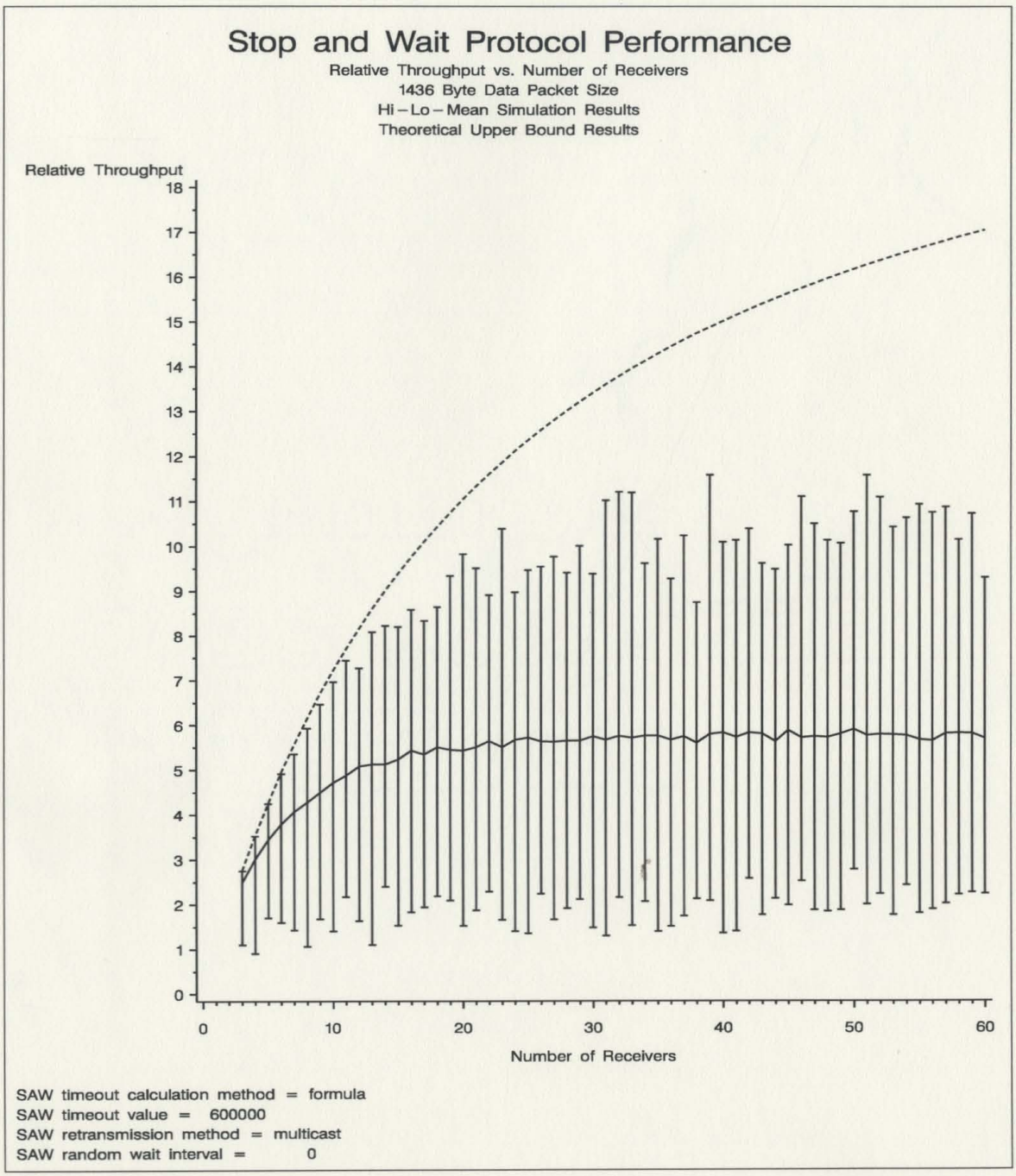

Figure 4.7: MSAW Performance: Relative Throughput for 1436-Byte Data (Mean)

Data Packet Size = 64								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	4.347	1.057	8.163	8.333	0.869	0.211	1.633	1.667
10	3.734	0.688	8.205	9.091	0.747	0.138	1.641	1.818
15	3.413	0.723	6.997	9.375	0.683	0.145	1.399	1.875
20	3.177	0.698	7.619	9.524	0.635	0.140	1.524	1.905
25	3.149	0.608	6.182	9.615	0.630	0.122	1.236	1.923
30	3.019	0.662	5.607	9.677	0.604	0.132	1.121	1.935
35	2.942	0.619	5.901	9.722	0.588	0.124	1.180	1.944
40	2.909	0.599	5.594	9.756	0.582	0.120	1.119	1.951
45	2.878	0.887	5.369	9.783	0.576	0.177	1.074	1.957
50	2.847	1.251	5.722	9.804	0.569	0.250	1.144	1.961
55	2.695	0.800	5.699	9.821	0.539	0.160	1.140	1.964
60	2.685	0.989	4.624	9.836	0.537	0.198	0.925	1.967
Data Packet Size = 848								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	27.009	11.123	36.054	36.301	2.905	1.196	3.878	3.904
10	33.109	8.409	54.220	56.989	3.561	0.904	5.831	6.129
15	34.675	9.047	58.998	70.354	3.729	0.973	6.345	7.566
20	34.674	8.870	68.831	79.699	3.729	0.954	7.403	8.571
25	35.785	7.817	62.871	86.601	3.849	0.841	6.762	9.314
30	35.330	8.535	60.456	91.908	3.800	0.918	6.502	9.884
35	35.048	8.027	64.803	96.114	3.769	0.863	6.969	10.337
40	35.145	7.797	63.284	99.531	3.780	0.839	6.806	10.704
45	35.167	11.475	62.069	102.361	3.782	1.234	6.675	11.009
50	35.126	16.087	66.499	104.743	3.778	1.730	7.152	11.265
55	33.551	10.414	67.011	106.777	3.608	1.120	7.207	11.484
60	33.612	12.846	55.986	108.532	3.615	1.382	6.021	11.672
Data Packet Size = 1436								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	33.034	16.318	40.703	40.888	3.451	1.705	4.252	4.271
10	45.195	13.456	66.729	69.171	4.721	1.406	6.970	7.225
15	50.173	14.705	78.499	89.900	5.241	1.536	8.200	9.391
20	51.996	14.573	94.102	105.744	5.431	1.522	9.830	11.046
25	54.827	12.957	90.657	118.248	5.727	1.353	9.470	12.352
30	55.059	14.173	89.825	128.367	5.751	1.480	9.383	13.409
35	55.224	13.380	97.252	136.725	5.769	1.398	10.159	14.282
40	55.863	13.027	96.570	143.744	5.835	1.361	10.087	15.015
45	56.300	19.093	95.932	149.722	5.881	1.994	10.021	15.639
50	56.579	26.648	103.102	154.875	5.910	2.784	10.770	16.178
55	54.366	17.407	104.637	159.362	5.679	1.818	10.930	16.646
60	54.667	21.435	89.045	163.306	5.710	2.239	9.301	17.058

Table 4.2: MSAW Performance: Absolute and Relative Throughput Summary Results for Varying Data Packet Sizes

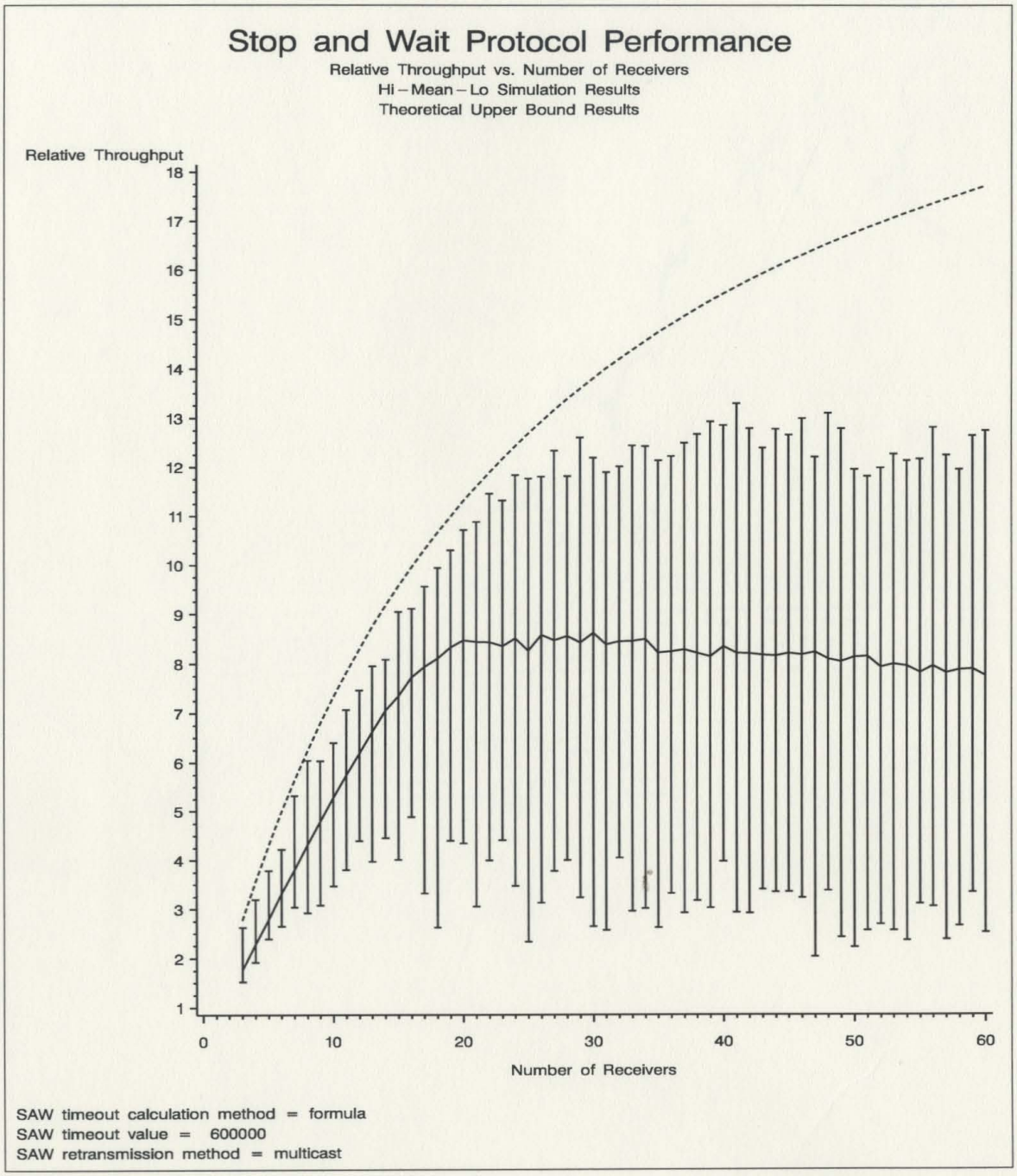

Figure 4.8: MSAW Performance: Relative Throughput using a SAWrand Value of 12,000 (Mean)

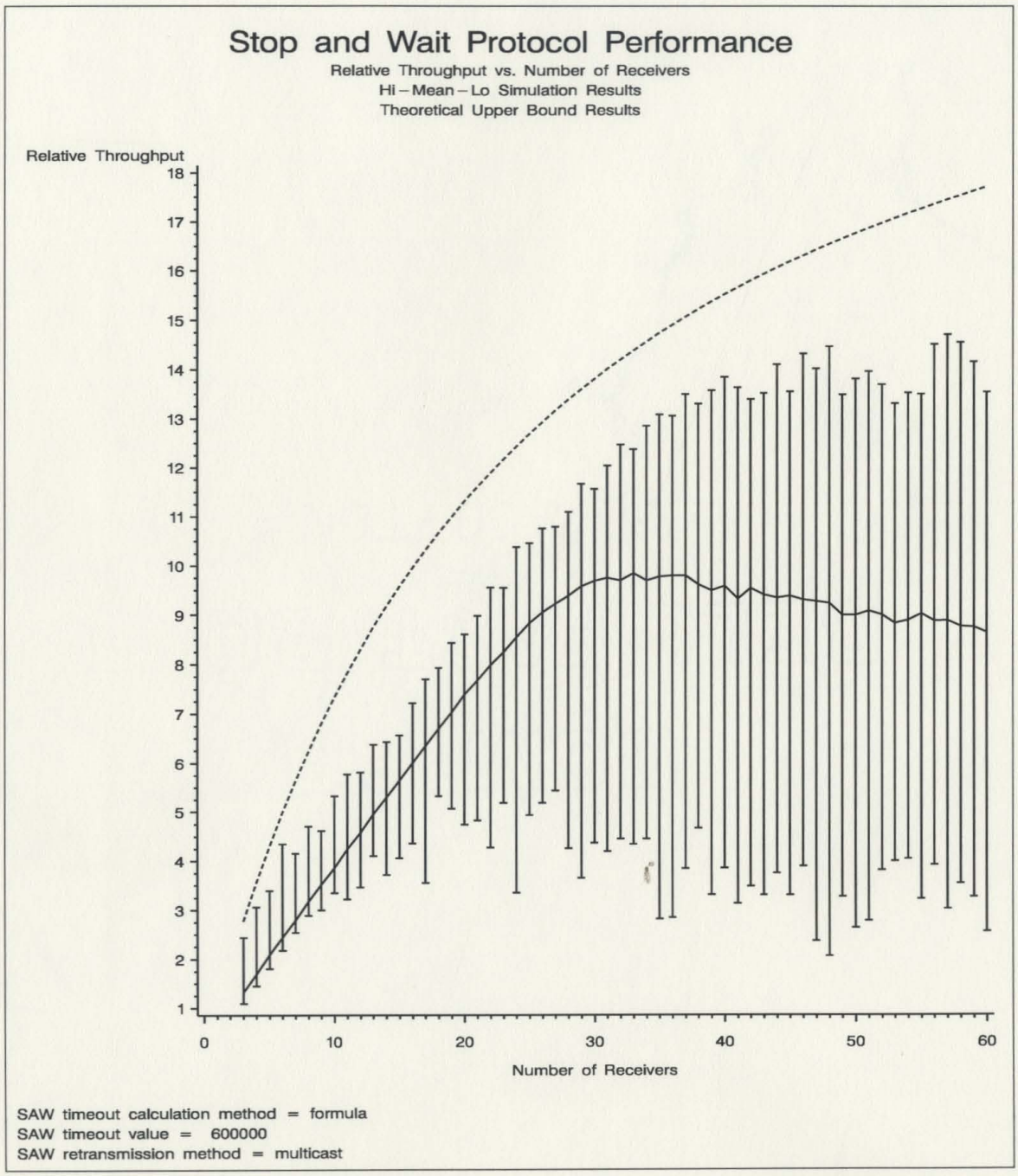

Figure 4.9: MSAW Performance: Relative Throughput using a SAWrand Value of 22,000 (Mean)

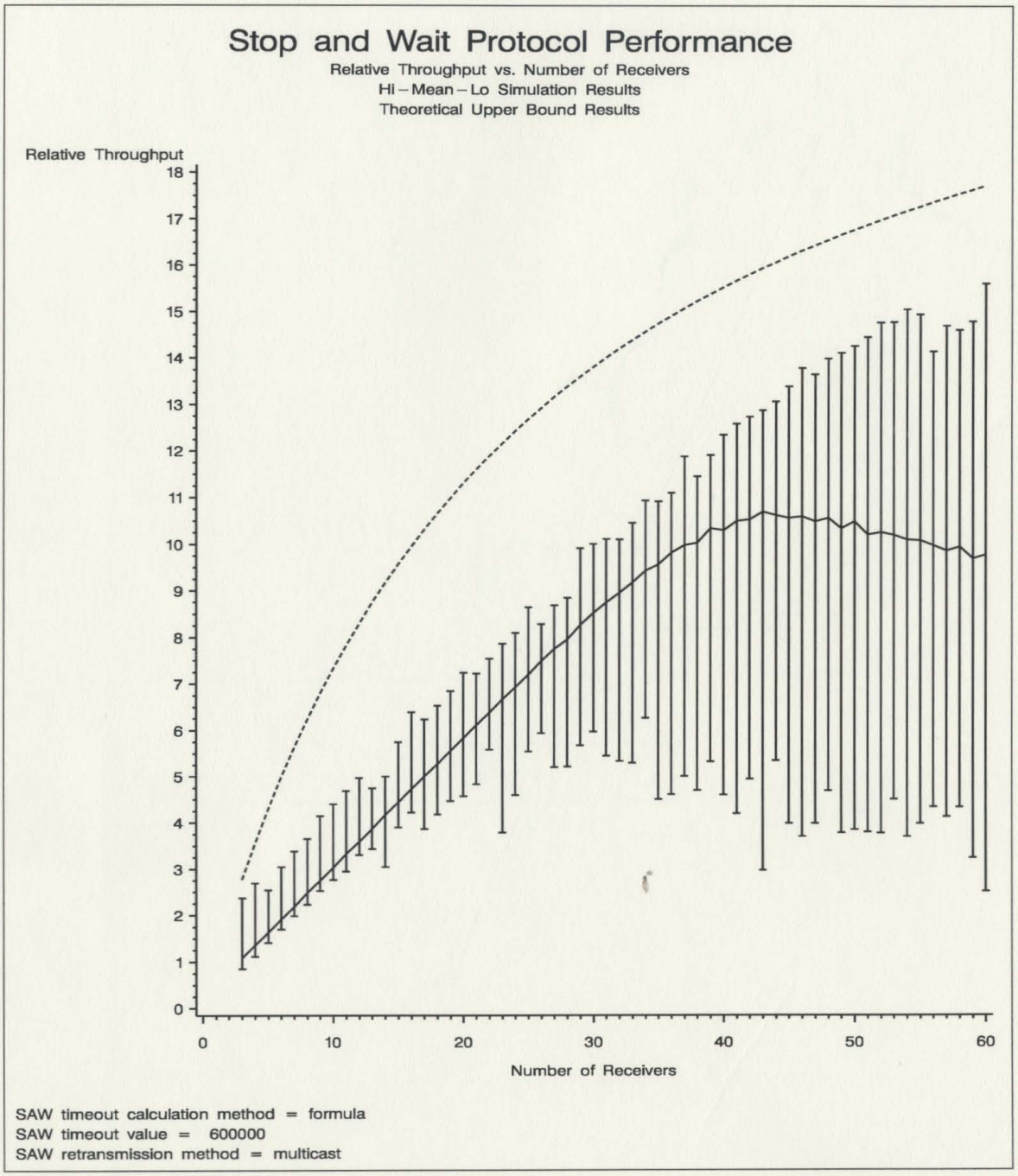

Figure 4.10: MSAW Performance: Relative Throughput using a SAWrand Value of 32,000 (Mean)

SAW randwait = 12000								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	26.766	22.991	36.346	41.295	2.789	2.396	3.788	4.304
10	50.726	33.310	61.414	70.343	5.286	3.471	6.400	7.331
15	70.498	38.513	86.875	91.889	7.347	4.014	9.054	9.576
20	81.347	41.740	102.902	108.506	8.478	4.350	10.724	11.308
25	79.334	22.431	112.951	121.713	8.268	2.338	11.771	12.684
30	82.761	25.439	116.975	132.461	8.625	2.651	12.191	13.805
35	78.953	25.228	116.459	141.378	8.228	2.629	12.137	14.734
40	80.152	38.260	123.277	148.897	8.353	3.987	12.847	15.517
45	78.872	32.273	121.402	155.321	8.220	3.363	12.652	16.187
50	78.145	21.375	114.705	160.873	8.144	2.228	11.954	16.766
55	75.164	29.886	116.743	165.721	7.833	3.115	12.166	17.271
60	74.432	24.257	122.267	169.989	7.757	2.528	12.742	17.716
SAW randwait = 22000								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	20.043	17.320	32.601	41.295	2.089	1.805	3.398	4.304
10	37.048	32.136	51.126	70.343	3.861	3.349	5.328	7.331
15	54.187	38.984	62.951	91.889	5.647	4.063	6.560	9.576
20	70.868	45.544	82.601	108.506	7.386	4.746	8.608	11.308
25	84.747	47.429	100.304	121.713	8.832	4.943	10.453	12.684
30	92.955	41.930	110.941	132.461	9.687	4.370	11.562	13.805
35	93.792	27.046	125.510	141.378	9.775	2.819	13.080	14.734
40	91.974	37.047	132.816	148.897	9.585	3.861	13.842	15.517
45	90.023	31.768	130.006	155.321	9.382	3.311	13.549	16.187
50	86.337	25.362	132.516	160.873	8.998	2.643	13.810	16.766
55	86.629	30.986	129.517	165.721	9.028	3.229	13.498	17.271
60	82.985	24.597	129.903	169.989	8.648	2.563	13.538	17.716
SAW randwait = 32000								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	15.613	13.506	24.436	41.295	1.627	1.408	2.547	4.304
10	29.049	26.515	42.214	70.343	3.027	2.763	4.399	7.331
15	42.645	37.441	55.212	91.889	4.444	3.902	5.754	9.576
20	55.990	43.926	69.446	108.506	5.835	4.578	7.237	11.308
25	69.080	53.223	83.010	121.713	7.199	5.547	8.651	12.684
30	81.809	57.357	95.975	132.461	8.526	5.978	10.002	13.805
35	91.754	43.274	104.770	141.378	9.562	4.510	10.919	14.734
40	98.858	44.202	118.519	148.897	10.303	4.607	12.352	15.517
45	101.330	38.275	128.457	155.321	10.560	3.989	13.387	16.187
50	100.640	37.032	136.775	160.873	10.488	3.859	14.254	16.766
55	96.801	38.309	143.352	165.721	10.088	3.992	14.940	17.271
60	93.818	24.141	149.692	169.989	9.777	2.516	15.600	17.716

Table 4.3: MSAW Performance: Absolute and Relative Throughput Summary Results for Varying SAWrand Values

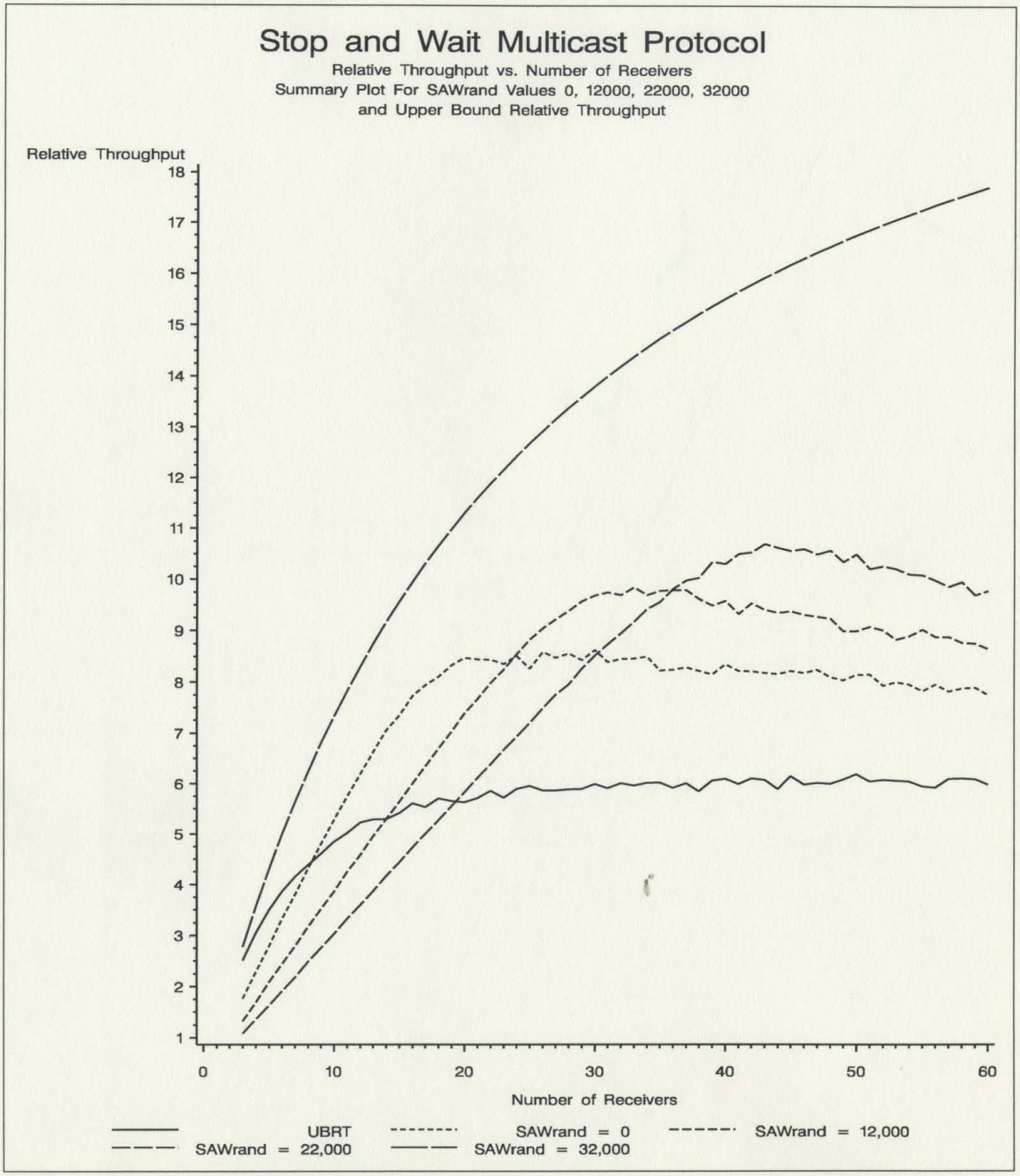

Figure 4.11: MSAW Performance: Summary of Mean Relative Throughput for SAWrand Values 0, 12,000, 22,000 and 32,000

4.3 Block Acknowledgment Protocol Performance

The multicast block acknowledgment protocol attempts to improve on the performance of the multicast stop and wait protocol by sending multiple multicast data packets at a time and acknowledging all the outstanding packets with one acknowledgment per receiver. This technique reduces the number of acknowledgments required, and therefore should increase the throughput obtained by the multicast block acknowledgment protocol over the multicast stop and wait protocol. To test this premise, we simulate the multicast block acknowledgment protocol with window sizes of 5, 10 and 15 from 3 to 60 receivers using 450 packets per simulation. The number of packets is chosen so that the multicast block acknowledgment protocol can always send full windows for window sizes of 5, 10, and 15.

Absolute and relative throughput figures for the multicast block acknowledgment protocol with window sizes of 5, 10 and 15 are summarized in Table 4.4. Plots of relative throughput versus the number of receivers for window sizes 5, 10 and 15 are given in Figures 4.12, 4.13, and 4.14. These plots also include the upper bound relative throughput for the appropriate window size as derived in Section 4.1. Finally, a summary plot of simulated and upper bound relative throughput results for window sizes of 5, 10, and 15 is given in Figure 4.15.

It is important to note that the relative throughput values presented in this section are relative to an equivalent unicast block acknowledgment protocol. This allows us to compare the performance of the block acknowledgment protocol relative to an equivalent unicast version, and demonstrates the effect of using multicasting over unicast

transmissions. As a result, in order to compare the multicast block acknowledgment protocol with the multicast stop and wait protocol, we must use absolute throughput figures.

The results show that the multicast block acknowledgment protocol does perform better than the multicast stop and wait protocol, as expected. The multicast stop and wait protocol has an absolute throughput of 51.992 Mbps for 15 receivers, while the block acknowledgment protocol has absolute throughput results of 107.216 Mbps, 125.526 Mbps, and 132.387 Mbps for window sizes 5, 10, and 15 respectively for the same number of receivers. (Note that the absolute throughput values exceed the theoretical 10 Mbps throughput value of Ethernet; this is due to the fact that one packet of size d effectively carries rd bits of information when r receivers are participating in a multicast communication).

The results also demonstrate that the relative throughput of the multicast block acknowledgment protocol with respect to the unicast version improves as the number of receivers increases. For a window size of 10, relative throughput increases from a low of 4.781 at 5 receivers to a high of 30.698 at 60 receivers. The results also show that these gains eventually level off, but at different values for different window sizes.

The relative throughput performance of the block acknowledgment protocol also improves as the window sizes increase. For 30 receivers, the relative throughput increases from a value of 16.289 with a window size of 5, to a value of 20.622 with a window size of 10, and finally, a value of 22.913 with a window size of 15. Unfortunately, this higher performance does come at a price; the measure of message latency

in the block acknowledgment protocol includes the latency of all messages in a window — as the window sizes increase, the message latency increases dramatically.

The improved performance of the multicast block acknowledgment protocol over the multicast stop and wait protocol is due mainly to a dramatic reduction in the number of required acknowledgments. For every packet that does not require an explicit acknowledgment, the number of acknowledgments transmitted is reduced by the number of receivers r. For example, for 10 receivers and a window size of 5, the block acknowledgment protocol requires one acknowledgment packet per 5 packets per receiver, resulting in the transmission of 10 acknowledgment packets as opposed to the 50 acknowledgment packets required by the multicast stop and wait protocol. This is a reduction of 40 acknowledgment packets. In general, the multicast block acknowledgement protocol reduces the number of acknowledgments required from r acknowledgments per message to $\frac{r}{n}$ acknowledgments per message, where r is the number of receivers and n is the BAP window size. The effect of fewer acknowledgments is two-fold: the amount of time required for transmissions of acknowledgments is reduced, and the lesser network traffic reduces contention on the network, reducing the amount of time spent on collision resolution. The effect of requiring fewer acknowledgments is more pronounced in the higher range of receivers, where collision contention is a major factor reducing protocol throughput.

The multicast block acknowledgement protocol's performance, like the multicast stop and wait protocol, also improves as the number of receivers increases. The increase in the number of receivers improves the "effective" data transmitted per

multicast data packet; however, this effect is countermanded by corresponding increase in acknowledgments required. The overall effect is the asymptotic curve shown in Figures 4.12, 4.13, and 4.14. These results follow the trend exhibited by the upper bound relative throughput results derived in Section 4.1 and shown in Figure 4.15. The simulated multicast block acknowledgment protocol, however, does not achieve the magnitude of the upper bound performance results, due mainly to the effect of collision resolution.

The multicast block acknowledgment protocol also increases its performance when larger window sizes are used. This improvement is due mainly to the reduction in the number of acknowledgments required as the window size increases. For each single packet increase in the window size, the number of acknowledgments required by the multicast block acknowledgment protocol is reduced by r, where r is the number of receivers. As mentioned previously, any reduction in the number of acknowledgments required not only reduces overhead but also lowers the probability of collisions, thus reducing the amount of time spent resolving collisions. The combined effect on throughput performance is quite significant. The effect of increasing window size on protocol performance is clearly demonstrated in Figure 4.15.

However, increasing the window size does add a significant amount of latency to message delivery. Since messages are now acknowledged as a block, the message latency of any one message is the latency of the entire block; large window sizes cause corresponding increases in message latencies. Thus, the multicast block acknowledgment protocol is not suited to applications where quick response is important, for

example, interactive multi-user chat sessions. It is, however, particularly well-suited for high data traffic applications without tight time restraints, such as making multiple copies of distributed database files.

The failure of the multicast block acknowledgment protocol to achieve performance close to the upper bound relative throughput results is mainly due to the amount of time spent resolving collisions as the number of receivers grows larger. As in the multicast stop and wait protocol, we attempt to overcome this problem by adding randomly timed acknowledgments to the multicast block acknowledgment protocol. To test the effectiveness of this technique, we simulate the multicast block acknowledgment protocol for a fixed window size of 10, and `BAPrand` values of 12,000, 22,000, and 32,000 bit times.

Plots of relative throughput versus number of receivers for a window size of 10 using `BAPrand` values of 12,000, 22,000, and 32,000 bit times are given in Figures 4.16, Figure 4.17, and Figure 4.18. Summary relative throughput and message latency results are given in Table 4.5. A summary plot showing mean relative throughputs for `BAPrand` values of 0, 12,000, 22,000 and 32,000, as well as the derived upper bound relative throughput, is given in Figure 4.19.

The results demonstrate noticibly enhanced throughput performance; for example, for a window size of 10 and 40 receivers, relative throughput is 27.962 at 12,000, 29.671 at 22,000 and 30.681 at 32,000 bit times. This improvement is achieved for the same reasons as in stop and wait: using randomly timed acknowledgments reduces the possibility of collisions; thus reducing the amount of time spent resolving collisions. Note

also the close relationship between `BAPrand` values, number of receivers, and relative throughput: as `BAPrand` values increase, the relative throughput of higher number of receivers increases, while the relative throughput of lower number of receivers declines. This effect can be seen in Figure 4.19.

BAPwin = 5								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	45.666	37.552	47.929	47.977	4.605	3.787	4.833	4.838
10	81.620	61.818	90.681	92.224	8.231	6.234	9.145	9.300
15	107.216	69.770	128.008	133.158	10.812	7.036	12.909	13.428
20	128.030	95.268	152.013	171.139	12.911	9.607	15.329	17.258
25	147.904	90.366	183.973	206.474	14.915	9.113	18.552	20.822
30	161.531	112.578	197.553	239.432	16.289	11.353	19.922	24.145
35	166.481	90.771	222.823	270.244	16.789	9.154	22.470	27.252
40	179.041	103.343	233.790	299.113	18.055	10.421	23.576	30.164
45	185.964	83.496	250.513	326.218	18.753	8.420	25.263	32.897
50	188.509	98.464	274.562	351.715	19.010	9.929	27.688	35.468
55	205.325	102.051	281.035	375.743	20.706	10.291	28.341	37.891
60	204.267	100.490	292.561	398.425	20.599	10.134	29.503	40.178
BAPwin = 10								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	47.606	44.848	48.930	48.968	4.781	4.504	4.914	4.917
10	87.002	71.753	93.588	95.954	8.737	7.206	9.398	9.636
15	125.526	109.619	137.103	141.078	12.606	11.008	13.768	14.167
20	156.465	100.172	176.676	184.447	15.713	10.059	17.742	18.522
25	179.611	124.133	209.022	226.162	18.037	12.466	20.990	22.712
30	205.353	169.118	240.749	266.316	20.622	16.983	24.176	26.744
35	223.472	147.240	268.117	304.994	22.441	14.786	26.925	30.628
40	253.681	179.794	296.890	342.277	25.475	18.055	29.814	34.372
45	260.112	199.597	312.260	378.239	26.121	20.044	31.358	37.983
50	281.642	183.671	356.472	412.949	28.283	18.444	35.797	41.469
55	276.607	171.719	347.238	446.471	27.777	17.244	34.870	44.835
60	305.694	188.073	377.925	478.864	30.698	18.887	37.952	48.088
BAPwin = 15								
Recs	Mean AT	Min AT	Max AT	UBAT	Mean RT	Min RT	Max RT	UBRT
5	48.380	44.574	49.281	49.307	4.852	4.470	4.942	4.945
10	91.725	81.015	96.671	97.266	9.198	8.124	9.694	9.754
15	132.387	105.436	141.546	143.932	13.276	10.573	14.194	14.434
20	164.588	129.574	179.645	189.356	16.505	12.994	18.015	18.989
25	198.388	161.069	220.043	233.586	19.895	16.152	22.066	23.424
30	228.489	183.009	257.754	276.671	22.913	18.352	25.848	27.745
35	256.949	214.615	292.116	318.653	25.767	21.522	29.294	31.955
40	275.757	211.332	324.521	359.574	27.653	21.193	32.543	36.058
45	311.199	237.484	353.938	399.474	31.207	23.815	35.493	40.060
50	321.268	245.610	371.694	438.390	32.217	24.630	37.274	43.962
55	347.082	277.818	390.310	476.360	34.806	27.860	39.141	47.770
60	364.956	273.579	423.260	513.416	36.598	27.435	42.445	51.486

Table 4.4: MBAP Performance: Absolute and Relative Throughput Summary Results for Window Sizes 5, 10, and 15

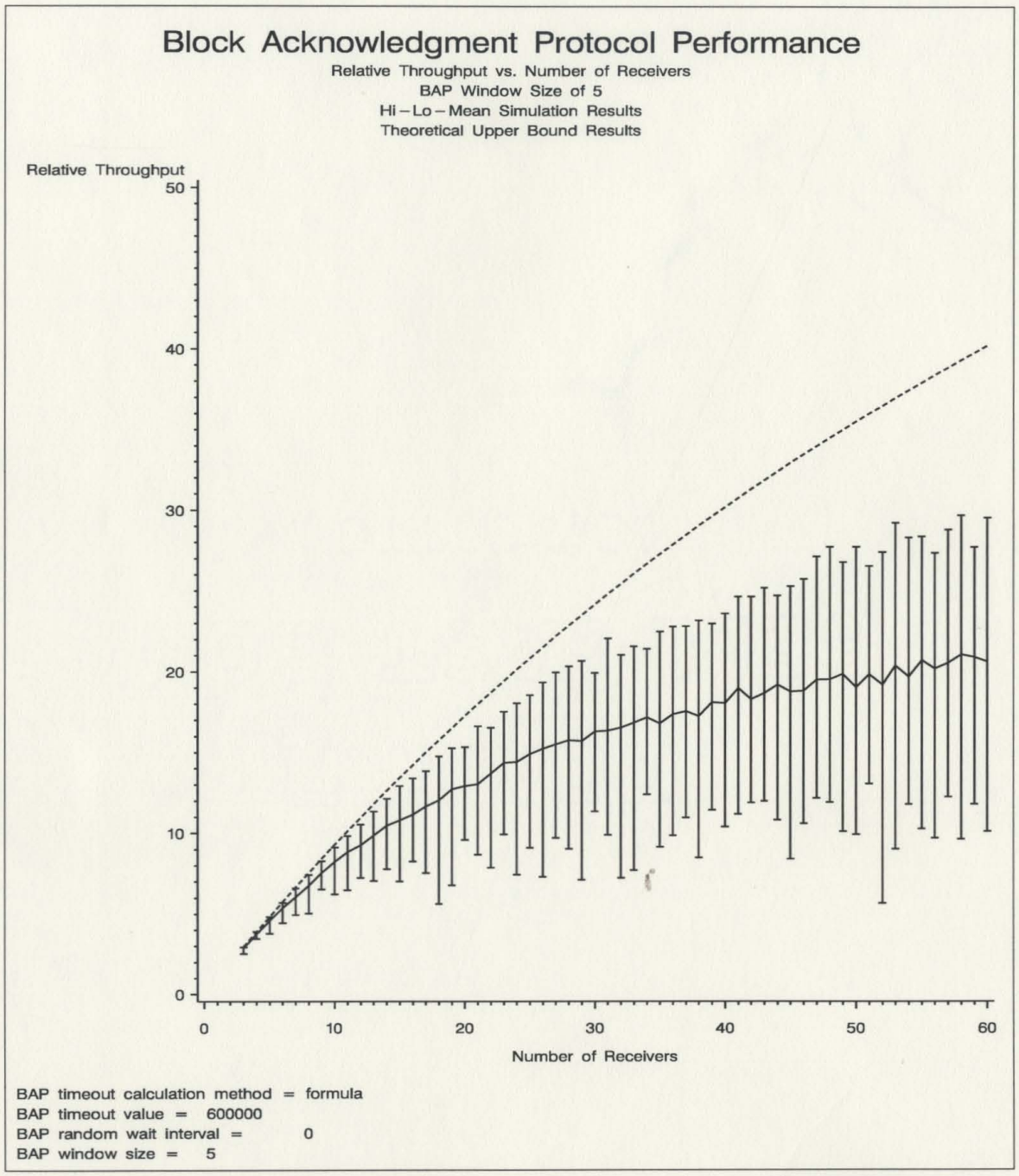

Figure 4.12: MBAP Performance: Relative Throughput for a Window Size of 5 (Mean)

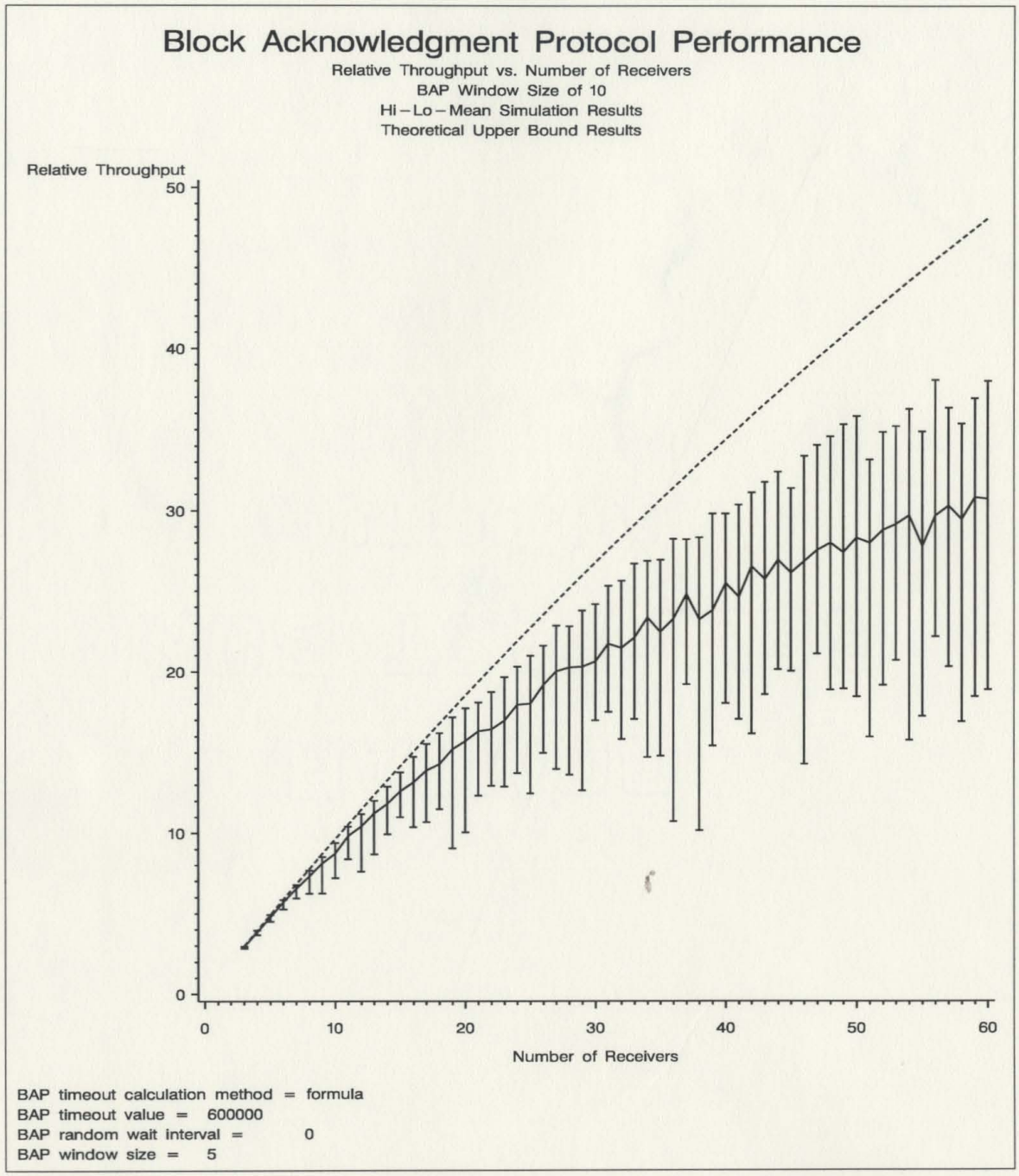

Figure 4.13: MBAP Performance: Relative Throughput for a Window Size of 10 (Mean)

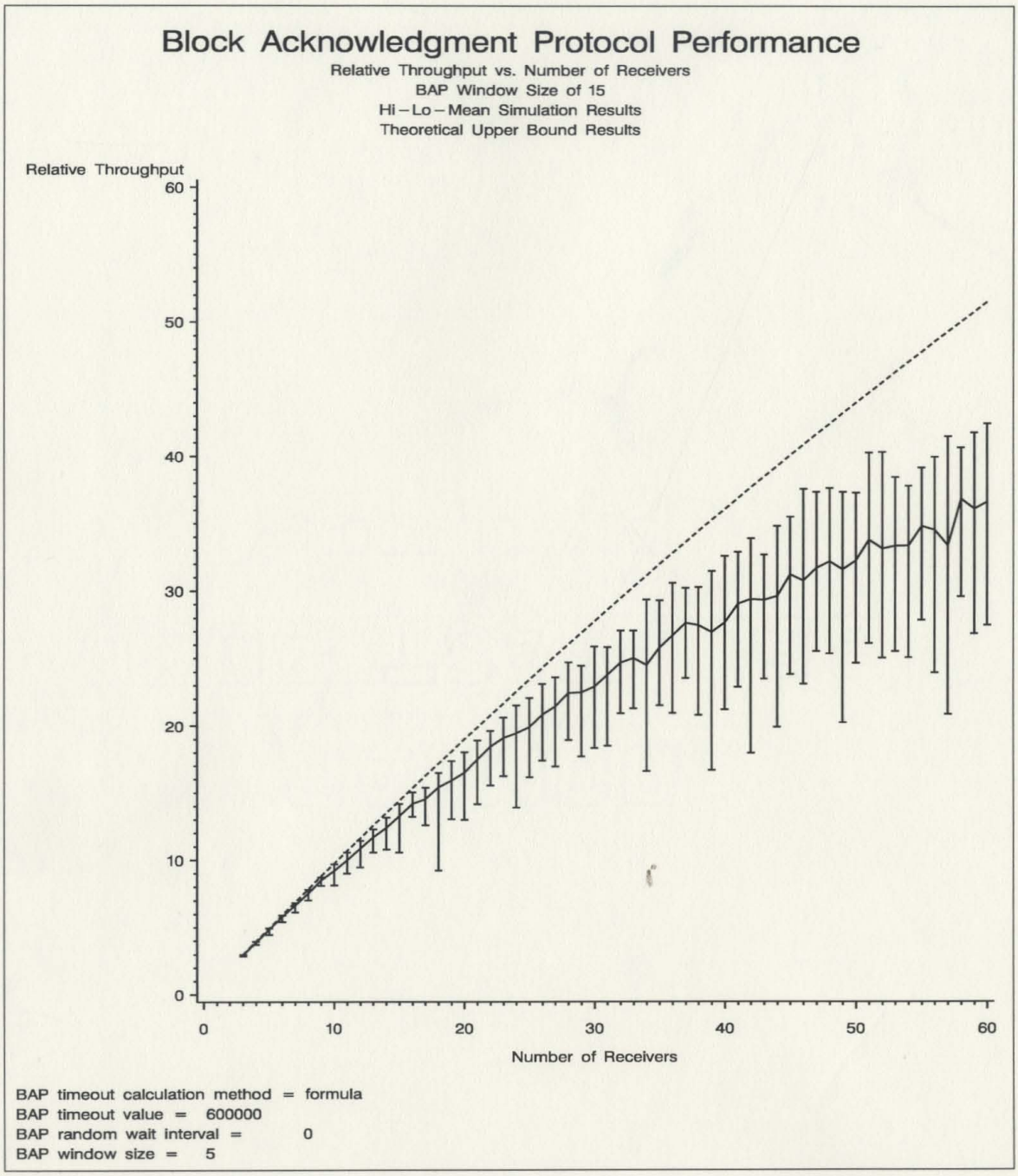

Figure 4.14: MBAP Performance: Relative Throughput for a Window Size of 15 (Mean)

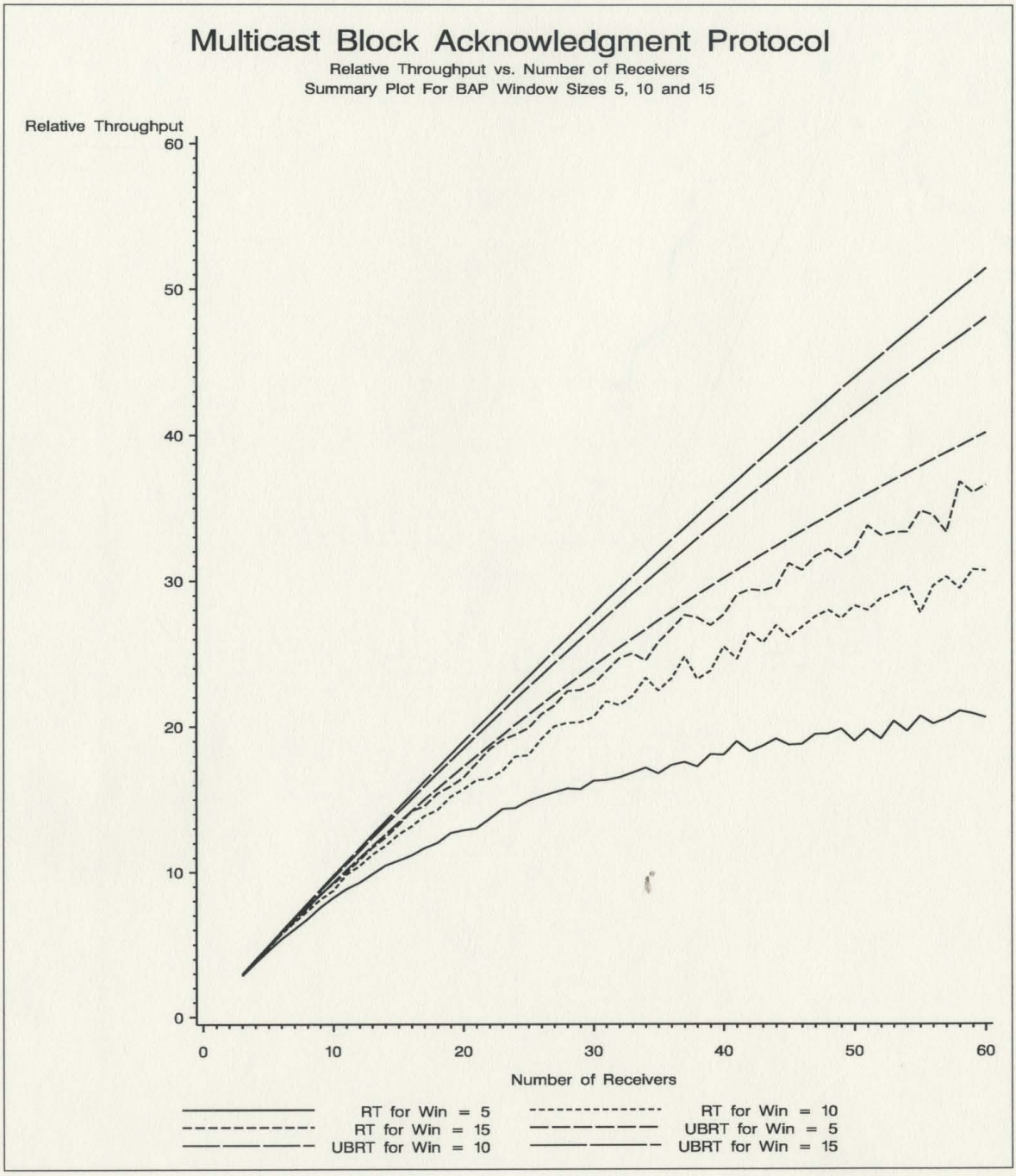

Figure 4.15: MBAP Performance: Summary of Mean Relative Throughput and Upper Bound Relative Throughput for Window Sizes 5, 10 and 15

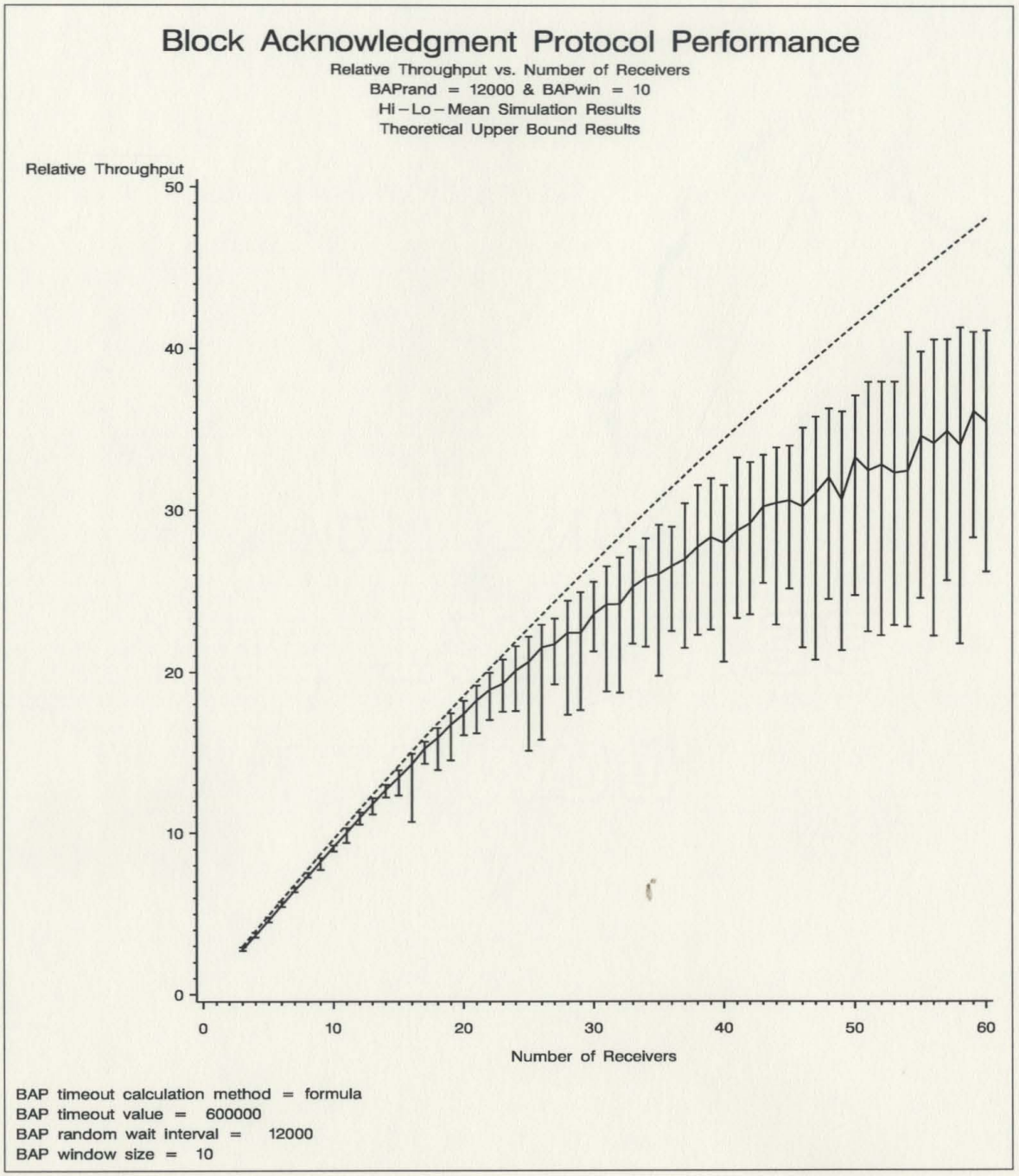

Figure 4.16: MBAP Performance: Relative Throughput using a BAPrand Value of 12,000 and a Window Size of 10 (Mean)

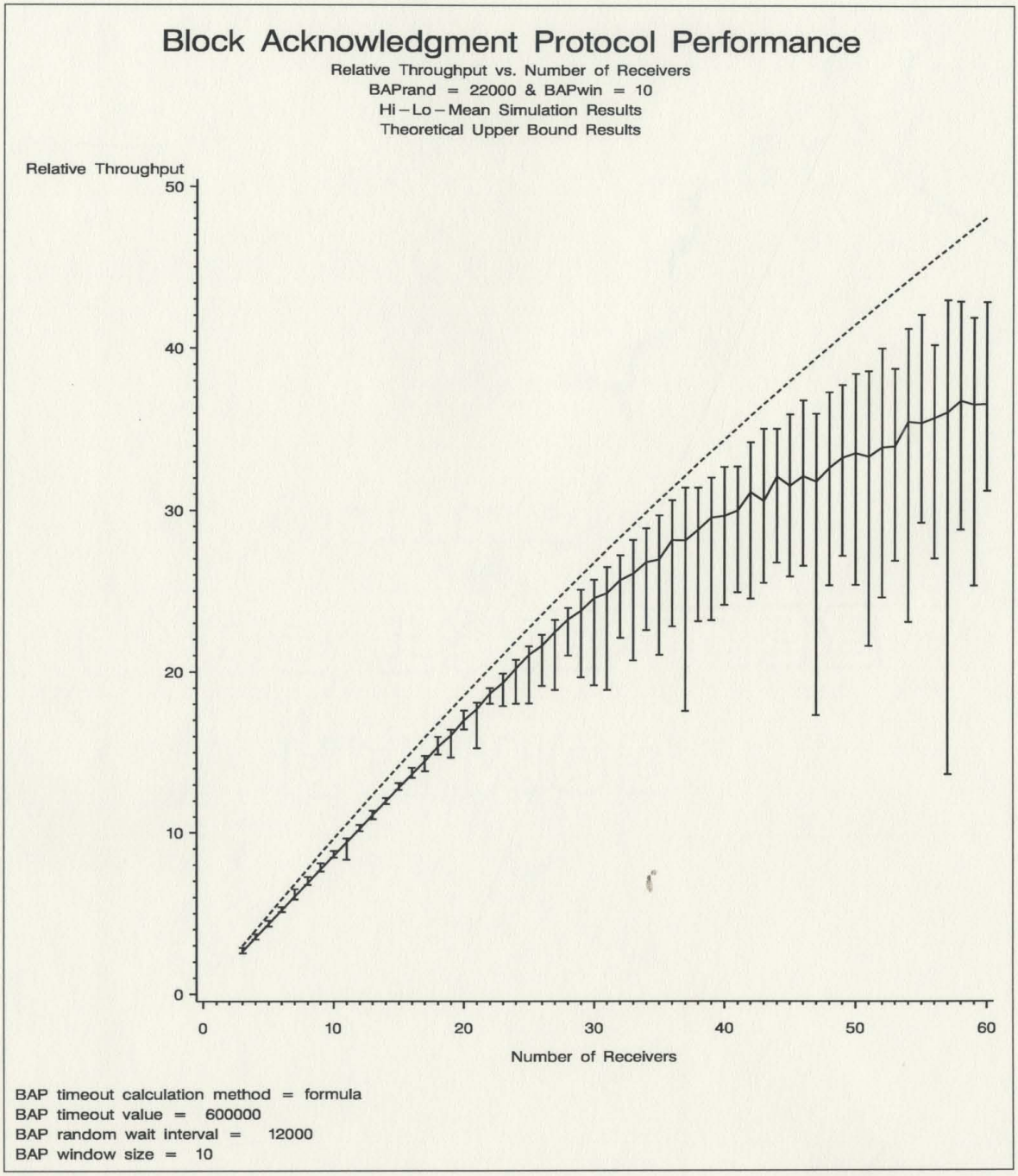

Figure 4.17: MBAP Performance: Relative Throughput using a BAPrand Value of 22,000 and a Size of 10 (Mean)

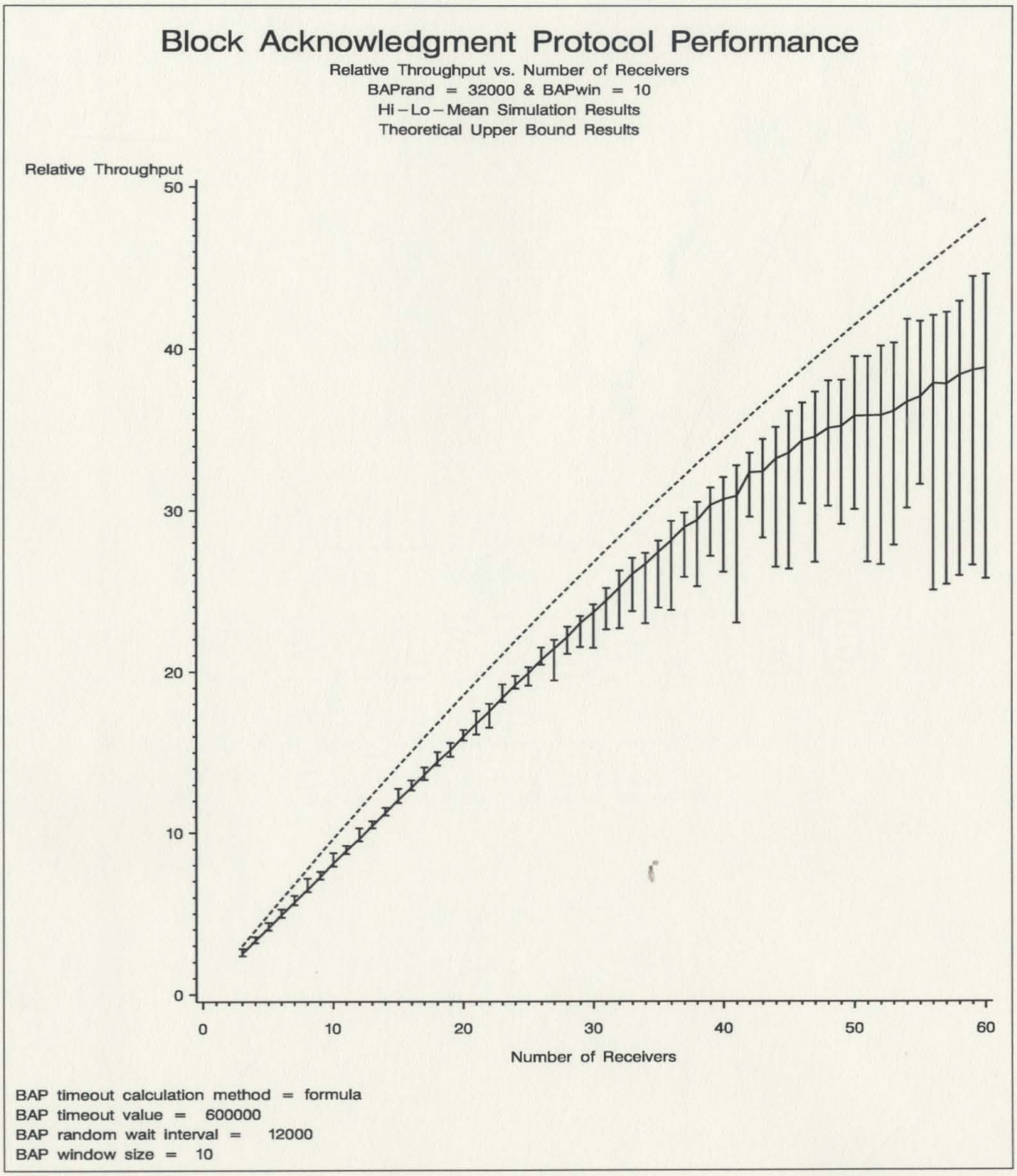

Figure 4.18: MBAP Performance: Relative Throughput using a BAPrand Value of 32,000 and a Window Size of 10 (Mean)

BAPwin = 10 & BAPrand = 12000								
Recs	Mean RT	Min RT	Max RT	UBT	Mean Lat	Min Lat	Max Lat	LBL
5	4.610	4.494	4.803	4.917	132300	126958	135682	124000
10	9.153	8.878	9.248	9.636	133240	131865	137370	126560
15	13.473	12.322	13.886	14.167	135865	131732	148456	129120
20	17.394	16.055	18.213	18.522	140363	133915	151920	131680
25	20.627	15.102	22.188	22.712	148628	137406	201882	134240
30	23.587	21.273	25.572	26.744	155565	143069	171980	136800
35	26.037	19.755	29.066	30.628	165008	146847	216064	139360
40	27.962	20.637	31.511	34.372	176013	154808	236375	141920
45	30.581	25.135	33.946	37.983	180512	161663	218337	144480
50	33.221	24.750	37.074	41.469	184907	164471	246365	147040
55	34.557	24.571	39.816	44.835	195939	168460	272981	149600
60	35.417	26.188	41.111	48.088	208405	177986	279408	152160
BAPwin = 10 & BAPrand = 22000								
Recs	Mean RT	Min RT	Max RT	UBT	Mean Lat	Min Lat	Max Lat	LBL
5	4.348	4.202	4.537	4.917	140299	134402	145095	124000
10	8.576	8.462	8.884	9.636	142215	137278	144121	126560
15	12.820	12.670	13.104	14.167	142701	139601	144374	129120
20	16.993	16.451	17.607	18.522	143550	138527	148265	131680
25	21.059	18.054	21.566	22.712	144890	141371	168874	134240
30	24.546	19.149	25.699	26.744	149536	142364	191053	136800
35	26.933	21.053	29.684	30.628	159443	143794	202742	139360
40	29.671	24.128	32.692	34.372	165178	149212	202171	141920
45	31.541	25.912	35.916	37.983	175005	152798	211787	144480
50	33.539	25.376	38.442	41.469	183118	158619	240286	147040
55	35.401	29.247	42.076	44.835	190905	159410	229337	149600
60	36.598	31.218	42.858	48.088	201186	170728	234386	152160
BAPwin = 10 & BAPrand = 32000								
Recs	Mean RT	Min RT	Max RT	UBT	Mean Lat	Min Lat	Max Lat	LBL
5	4.107	3.963	4.418	4.917	148622	138011	153870	124000
10	8.125	7.920	8.750	9.636	150193	139379	153972	126560
15	12.061	11.870	12.745	14.167	151692	143527	154109	129120
20	15.978	15.721	16.397	18.522	152665	148745	155146	131680
25	19.893	19.138	20.262	22.712	153273	150468	159310	134240
30	23.648	21.457	24.162	26.744	154783	151417	170508	136800
35	27.396	23.933	28.110	30.628	155926	151841	178342	139360
40	30.681	26.157	32.047	34.372	159284	152218	186495	141920
45	33.568	26.345	36.142	37.983	164171	151839	208307	144480
50	35.846	30.054	39.517	41.469	170846	154303	202887	147040
55	37.070	31.629	41.716	44.835	181897	160785	212065	149600
60	38.838	25.757	44.630	48.088	190146	163951	284083	152160

Table 4.5: MBAP Performance: Relative Throughput and Message Latency Summary Results for Varying BAPrand Values at a Window Size of 10

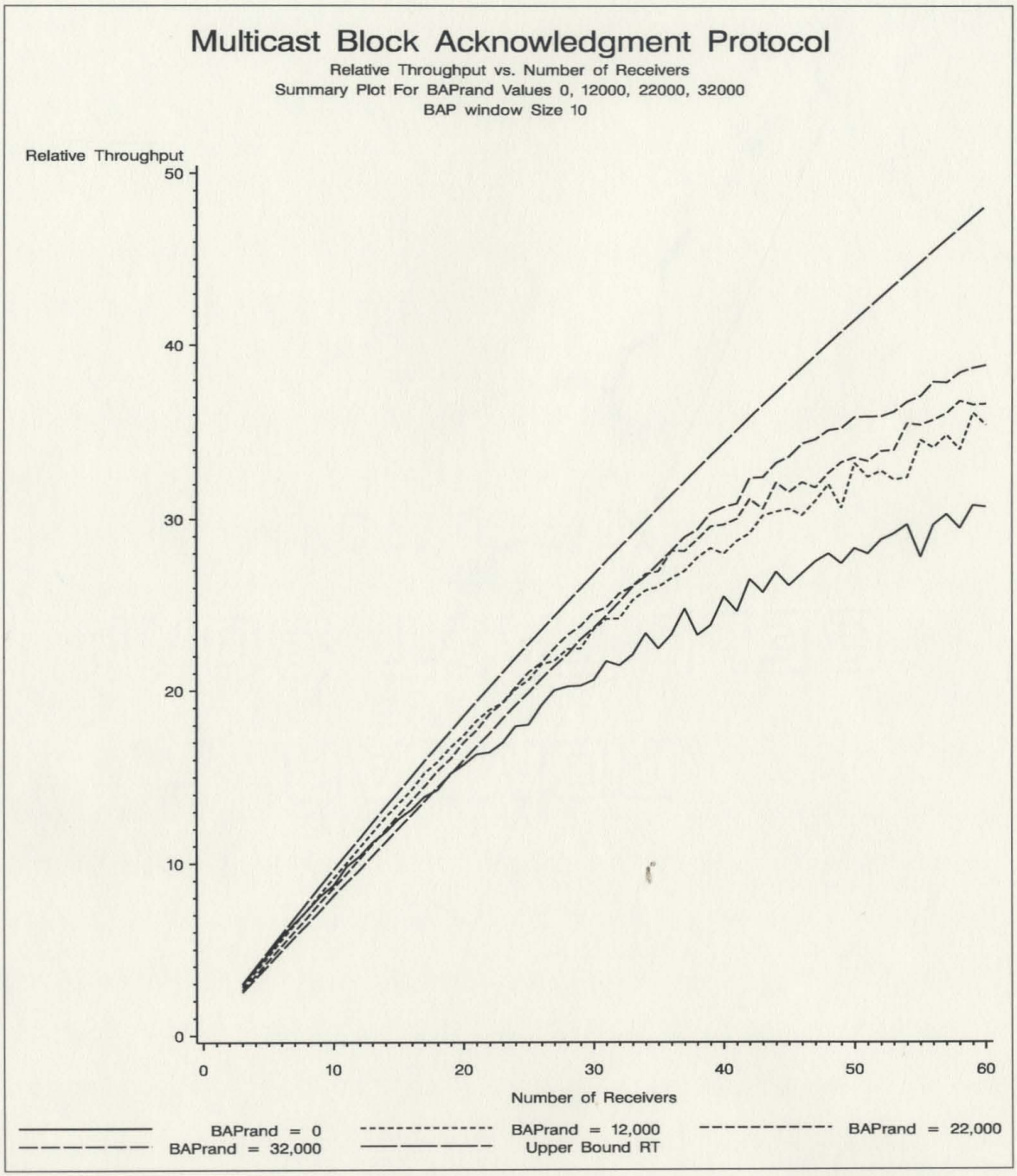

Figure 4.19: MBAP Performance: Summary of Mean Relative Throughput for BAPrand values 0, 12,000, 22,000 and 32,000

4.4 SAW and BAP Protocol Performance under Error Conditions

Previous simulation studies were performed in error-free conditions. We next investigate the effect of errors on the performance of both the multicast stop and wait and block acknowledgment protocols by simulating the protocols using failure rates of 0-5%. First, though, we note that the measure of relative throughput we have been using cannot be used for error analysis. The relative throughput measures the throughput of the multicast protocol with respect to the throughput of the equivalent unicast protocol in error-free conditions. Thus, we will use absolute message latency and number of retransmissions as our measures of protocol performance under error conditions.

The packet timeout parameter is a critical parameter when operating under error conditions in both the multicast stop and wait and block acknowledgment protocols. The timeout parameter is used by the sender node of both protocols to decide whether a packet has been lost and thus to initiate retransmissions. In both protocols, the value used for the timeout must account for the transmission time of the multicast packet, that of any acknowledgments, the interval used for randomly timed acknowledgments, if any, and the collision resolution time. If the timeout value is too short, the sender will retransmit packets prematurely, causing unnecessary delays and network congestion, even in error-free conditions. If the timeout value is too large, the amount of time required to detect a lost packet will be long, also causing unnecessary delays.

The simulator allows fine-tuning of the timeout parameters of both protocols in

two ways: by directly specifying the value to be used, or by specifying an increment to be added to a simple formula based on the number of receivers. For the multicast stop and wait protocol, the formula is:

$d + a * r + i$

where d is the transmission time of a data packet, a is the transmission time of an acknowledgment packet, r is the number of receivers, and i is the interval used for randomly timed acknowledgments, if any. This formula calculates the minimum amount of time required to transmit a multicast packet and receive all the acknowledgments from the receivers.

The formula for the multicast block acknowledgment protocol is:

$(d * n) + (a * r) + i + t$

where d is the transmission time of a data packet, n is the maximum BAP window size, a is the transmission time of an acknowledgment packet, r is the number of receivers, i is the interval used for randomly timed acknowledgments, if any, and t is the acknowledgment delay timeout value (used to force transmissions of acknowledgments when the maximum BAP window is not filled or acknowledgments are lost). This formula approximates the amount of time required to transmit the current window of multicast data packets and receive all acknowledgments from the receivers.

Note that neither of these formulas take into account the collision resolution time. As stated previously, the statistical nature of the collision resolution mechanism in Ethernet does not permit us to find an upper bound for the collision resolution time. Thus, we cannot find an exact formula which will account for the collision resolution

time and guarantee no retransmissions in error-free conditions.

We therefore select a timeout value by running several trial simulations, using the formula-based timeout calculation and choosing the smallest increment value that results in fewer than 2% of the packets being retransmitted under error-free conditions. Our previous simulation studies showed that the collision resolution time is highly sensitive to the number of receivers participating in the multicast protocol. Thus, we must choose an increment value separately for each number of receivers. We choose increment values of 45,000, 90,000 and 135,000 bit times for 20, 40 and 60 receivers for both the multicast stop and wait and block acknowledgment protocols.

The results for the multicast stop and wait protocol for 20, 40, and 60 receivers are tabulated in Table 4.6. The results for the multicast stop and wait protocol with a `SAWrand` interval of 22,000 bit times for the same numbers of receivers are in Table 4.7.

The results for the multicast block acknowledgment protocol for 20, 40, and 60 receivers for window sizes 5, 10, and 15 are found in Tables 4.8, 4.9, and 4.10. The same simulation was also performed for 20, 40 and 60 receivers with a `BAPRANDWAIT` interval of 22,000 bit times, and the results tabulated in Tables 4.11, 4.12, and 4.13.

For the multicast stop and wait protocol, using 20 receivers, a failure rate of 1% caused the mean message latency to jump to 98,793 bit times from 47,237 bit times in error-free conditions, an increase of 109%. However, each subsequent increase of one percentage point in failure rate only caused latencies to increase by 10-30% each time. Similar results were found for 40 and 60 receivers. Using randomly timed

acknowledgments with a `SAWrand` value of 22,000 bit time lowered absolute message latencies but exhibited the same general trend as the simple multicast stop and wait protocol.

The results for the multicast block acknowledgment protocol show an even more dramatic deterioration of performance under error conditions. For a window size of 10, with 20 receivers, a failure rate of 1% resulted in mean message latencies of 2,681,561 bit times, an increase of 1586% over the mean message latency of 159,006 bit times under error-free conditions. Error rates of 2-5% showed similar mean message latency values. Using randomly timed acknowledgments with a `BAPrand` value of 22,000 bit times did not change the overall trend of the results.

The low tolerance for errors in both protocols is due to the relationship between the sender timeout value, error detection, and the collision resolution time. The sender timeout value must include an estimation of the collision resolution time; otherwise premature timeouts can cause significant network congestion. However, as the number of receivers becomes larger, the estimate of the collision resolution grows, forcing larger timeout values. Larger timeout values lengthen the amount of time which must elapse before the sender can determine that an error has occurred and initiate retransmissions. Thus, response in the presence of errors, particularly with large number of receivers, is poor.

Receivers = 20						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	47237	26704	195584	0.010	0.000	2.000
1	98793	31248	377040	0.932	0.000	3.000
2	120507	33328	380360	1.296	0.000	4.000
3	148801	36464	475568	1.750	0.000	6.000
4	176844	47504	587280	2.204	0.000	8.000
5	196362	48048	509872	2.510	0.000	7.000
Receivers = 40						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	89609	50384	369824	0.018	0.000	2.000
1	215274	55984	677968	1.252	0.000	3.000
2	257787	122832	667344	1.682	0.000	5.000
3	327246	135888	1127760	2.276	1.000	9.000
4	384772	147184	1003632	2.790	1.000	8.000
5	452190	150288	1260656	3.362	1.000	10.000
Receivers = 60						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	137959	72304	701472	0.034	0.000	2.000
1	321829	135664	910328	1.336	0.000	5.000
2	420496	200096	950224	1.952	1.000	5.000
3	530055	197912	1462992	2.638	1.000	8.000
4	633635	241016	1498104	3.250	1.000	8.000
5	755114	380416	2030416	3.966	2.000	11.000

Table 4.6: MSAW Performance: Summary Results with 0–5% Failure Rates for 20, 40 and 60 Receivers

Receivers = 20 & SAWrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	34532	30374	51644	0.000	0.000	0.000
1	79167	28068	212566	0.534	0.000	2.000
2	112207	31220	293570	0.936	0.000	3.000
3	135357	32141	481015	1.202	0.000	5.000
4	152050	31304	482789	1.396	0.000	5.000
5	173318	32539	481184	1.636	0.000	5.000
Receivers = 40 & SAWrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	56116	37527	162197	0.000	0.000	0.000
1	176730	38338	597029	0.952	0.000	4.000
2	231888	55776	594388	1.368	0.000	4.000
3	268015	54780	822431	1.626	0.000	4.000
4	315032	164043	1043228	1.958	1.000	7.000
5	375575	162351	1042648	2.390	1.000	7.000
Receivers = 60 & SAWrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	90993	56388	258586	0.000	0.000	0.000
1	287254	121623	849213	1.178	0.000	4.000
2	375843	214466	1068943	1.644	1.000	5.000
3	463705	226497	1419619	2.104	1.000	7.000
4	553931	223855	1413077	2.572	1.000	7.000
5	631142	236067	1438749	2.972	1.000	7.000

Table 4.7: MSAW Performance: Summary Results with 0–5% Failure Rates for 20, 40 and 60 Receivers using a SAWrand value of 22,000

Receivers = 20 & GBNwin = 5						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	95466	77744	149456	0.000	0.000	0.000
1	519721	99280	1796960	2.086	0.000	9.000
2	655947	193904	1734480	2.828	0.000	9.000
3	737380	176640	2194784	3.490	0.000	10.000
4	705172	190112	1968720	3.310	1.000	10.000
5	729146	162536	1928224	3.490	0.000	10.000
Receivers = 20 & GBNwin = 10						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	159006	140544	230720	0.000	0.000	0.000
1	2681561	456352	3934640	8.760	1.000	11.000
2	2293372	381584	4113152	7.516	1.000	11.000
3	2395365	428192	3959184	8.050	1.000	12.000
4	2342186	226032	3446464	7.806	0.000	11.000
5	2537155	470720	3749408	8.816	1.000	13.000
Receivers = 20 & GBNwin = 15						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	216226	189672	253520	0.000	0.000	0.000
1	4216032	596192	4939696	9.694	1.000	11.000
2	3977723	1266464	5804320	9.470	3.000	12.000
3	4159637	994912	4806112	9.884	2.000	12.000
4	3891480	612240	5359552	9.754	1.000	12.000
5	3888046	619840	4851744	9.878	1.000	13.000

Table 4.8: MBAP Performance: Summary Results with 0–5% Failure Rates for 20 Receivers and Window Sizes of 5, 10 and 15

Receivers = 40 & GBNwin = 5						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	137103	105008	241200	0.000	0.000	0.000
1	1486439	279120	3290816	5.014	1.000	10.000
2	1567606	200016	3546912	5.388	0.000	12.000
3	1693525	235920	3656464	6.018	0.000	11.000
4	1597767	380488	2936720	5.818	1.000	10.000
5	1561641	416440	3216784	5.654	1.000	11.000
Receivers = 40 & GBNwin = 10						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	196719	163680	298912	0.000	0.000	0.000
1	4048725	523912	6251904	9.140	1.000	13.000
2	3912529	1352912	5694048	9.292	2.000	12.000
3	3819886	664168	6213120	9.254	1.000	12.000
4	3755537	918432	6434896	9.164	1.000	13.000
5	3757943	1182264	5855584	9.366	3.000	14.000
Receivers = 40 & GBNwin = 15						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	251020	204616	284304	0.000	0.000	0.000
1	6623085	3519056	8709744	10.066	9.000	13.000
2	5980839	2669392	9335952	9.668	3.000	12.000
3	5994812	753736	8943776	9.754	1.000	12.000
4	5660538	2819200	9812912	9.702	6.000	12.000
5	5743338	2995824	8372992	9.884	4.000	12.000

Table 4.9: MBAP Performance: Summary Results with 0–5% Failure Rates for 40 Receivers and Window Sizes of 5, 10 and 15

Receivers = 60 & GBNwin = 5						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	185298	126480	310512	0.000	0.000	0.000
1	1812103	489712	5237472	4.834	1.000	11.000
2	2017384	613472	4781728	5.616	1.000	11.000
3	2318213	512816	5677792	6.708	1.000	11.000
4	2346614	537776	4515392	6.960	1.000	11.000
5	2195509	622800	4602912	6.602	1.000	11.000
Receivers = 60 & GBNwin = 10						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	249589	188832	369504	0.000	0.000	0.000
1	5538971	686656	9426832	9.240	1.000	15.000
2	5236987	1109360	9744592	9.248	2.000	14.000
3	5727595	665760	9139520	9.636	1.000	14.000
4	5442980	2462224	9233680	9.512	3.000	14.000
5	5342846	2376848	9195200	9.440	2.000	13.000
Receivers = 60 & GBNwin = 15						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	306753	260784	367760	0.000	0.000	0.000
1	8871447	2079216	12291936	9.662	4.000	12.000
2	8488712	3019872	13120688	9.926	6.000	14.000
3	8499577	2806992	10976704	9.916	6.000	12.000
4	8326783	4693376	12891952	9.986	3.000	13.000
5	7997450	3930688	12747632	9.758	5.000	12.000

Table 4.10: MBAP Performance: Summary Results with 0–5% Failure Rates for 60 Receivers and Window Sizes of 5, 10 and 15

Receivers = 20 & GBNwin = 5 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	82886	78950	88525	0.000	0.000	0.000
1	452812	82021	2043572	1.754	0.000	8.000
2	559948	81673	1711696	2.290	0.000	9.000
3	639130	215126	1508323	2.866	0.000	8.000
4	658837	221072	2058455	2.986	0.000	10.000
5	667654	247472	1725603	3.146	1.000	8.000
Receivers = 20 & GBNwin = 10 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	143596	140030	150928	0.000	0.000	0.000
1	2223497	206246	3671104	6.988	0.000	11.000
2	2293055	581035	3452570	7.418	1.000	12.000
3	2369949	228610	3674969	7.598	0.000	12.000
4	2174899	497091	4002767	7.224	1.000	11.000
5	2303744	426416	3569504	7.814	1.000	12.000
Receivers = 20 & GBNwin = 15 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	204238	178072	215757	0.000	0.000	0.000
1	4230665	228441	4749904	9.678	0.000	11.000
2	4080021	218588	4830649	9.534	0.000	12.000
3	4038868	2375543	5388608	9.874	6.000	12.000
4	3848600	1859264	5534432	9.546	5.000	12.000
5	4027267	1046284	4489632	9.820	2.000	13.000

Table 4.11: MBAP Performance: Summary Results with 0–5% Failure Rates for 20 Receivers and Window Sizes of 5, 10 and 15 using a BAPrand value of 22,000

Receivers = 40 & GBNwin = 5 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	107757	86278	187928	0.000	0.000	0.000
1	1300706	123767	3263441	4.058	0.000	10.000
2	1252398	356481	3142971	4.040	1.000	11.000
3	1339005	257768	3306296	4.480	0.000	10.000
4	1405239	372813	3357995	4.714	1.000	10.000
5	1366955	430104	2889539	4.850	1.000	10.000
Receivers = 40 & GBNwin = 10 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	167565	149910	261615	0.000	0.000	0.000
1	3729939	1150813	6387439	8.652	2.000	11.000
2	3823477	1312768	6351679	8.978	3.000	11.000
3	3765843	627774	5807152	9.058	1.000	14.000
4	3753621	682182	6235376	9.128	1.000	12.000
5	3665466	1266245	5571758	9.320	3.000	13.000
Receivers = 40 & GBNwin = 15 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	224684	206268	262488	0.000	0.000	0.000
1	6344133	683288	8773600	9.822	1.000	12.000
2	5957945	364699	8841889	9.542	0.000	13.000
3	6108010	3618256	8123424	9.918	6.000	13.000
4	5739297	2163499	8519200	9.794	5.000	12.000
5	5656157	1922403	8438805	9.834	4.000	13.000

Table 4.12: MBAP Performance: Summary Results with 0–5% Failure Rates for 40 Receivers and Window Sizes of 5, 10 and 15 using a BAPrand Value of 22,000

Receivers = 60 & GBNwin = 5 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	136451	104964	221699	0.000	0.000	0.000
1	2067479	469184	4777728	5.602	1.000	11.000
2	1978096	566816	5054475	5.366	1.000	11.000
3	2152196	586940	4579936	5.984	1.000	11.000
4	2252952	590246	4208691	6.572	1.000	11.000
5	2298450	573398	4041902	6.802	1.000	11.000
Receivers = 60 & GBNwin = 10 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	197240	169896	266807	0.000	0.000	0.000
1	4997515	535726	8892976	8.936	1.000	13.000
2	5365264	731392	8448556	9.016	1.000	11.000
3	5326984	2740727	9351038	9.266	5.000	13.000
4	5158330	1460389	7778222	9.218	3.000	12.000
5	5143863	1754290	8031444	9.468	4.000	13.000
Receivers = 60 & GBNwin = 15 & GBNrand = 22,000						
FailRate	Mean Lat	Min Lat	Max Lat	Mean Retr	Min Retr	Max Retr
0	262493	227111	347189	0.000	0.000	0.000
1	9061660	876224	13693616	9.794	1.000	13.000
2	8409535	2090674	13586272	9.840	4.000	12.000
3	8245958	3336656	13513622	9.652	4.000	12.000
4	8212559	3358352	12139376	9.942	5.000	14.000
5	7864979	2623072	13318203	9.768	5.000	12.000

Table 4.13: MBAP Performance: Summary Results with 0–5% Failure Rates for 60 Receivers and Window Sizes of 5, 10 and 15 using a BAPrand Value of 22,000

Chapter 5

Conclusions and Further Work

We chose to study multicast protocols because of their relevancy to high-bandwidth distributed network applications, such as multi-user whiteboard systems and distributed databases. We designed and built a simulation system to study multicast protocols in an Ethernet environment. The simulation system eliminates the need for a dedicated testbed network.

The simulator follows a layered approach, the lower two layers implement the underlying Ethernet network, and the top layer implements one of several multicast protocols. The simulator was designed to be extensible; other multicast protocols can easily be added as alternate top layers. The simulator records data such as message latency and retransmissions, which can later be used to analyse protocol performance.

Users can control the simulation via a command file, which contains both general and protocol-specific parameters. General parameters specified in the command file

include the number of receivers, size of data and acknowledgment packets, number of packets to be sent, and the error rate of the underlying network. As well, the command file can be extended to include protocol-specific parameters; in the multicast stop and wait protocol, for example, this includes timeout values and the interval to be used for randomly timed acknowledgments.

The system simulates errors in an Ethernet environment by implementing a simple error model which randomly drops packets at interfaces based on an user-configurable error rate. This allows simulation of multicast protocols in varying error conditions.

The simulator was used to study the protocol performance of two simple multicast protocols, stop and wait and block acknowledgment. Simulations were performed under varying conditions, and the results analysed using the SAS System to compile summary results and plots. These results were compared against comparable unicast protocols, and derived theoretical bounds.

An analysis of the multicast stop and wait protocol gives a maximum relative throughput of 24.719 over equivalent unicast protocols when using a data packet size of 1518 bytes, and an acknowledgment packet size of 64 bytes. The simulation studies show that while the stop and wait protocol does not reach this maximum theoretical performance, it does show steady improvements in relative throughput for small number of receivers which eventually reach an asymptotic value. This behaviour is consistent with derived upper bound theoretical results.

The multicast stop and wait protocol is also affected by the size of the data payload. Theoretical results show that the relative throughput of the multicast stop and wait

protocol declines sharply as the size of the data packet decreases, eventually reaching a maximum relative throughput of 2.0 for a data packet size of 64 bytes. Simulation results also follow this trend; as the data packet size decreases, a dramatic decline in protocol performance occurs.

Randomly timed acknowledgments were added to the multicast stop and wait protocol to reduce the collision rate. This technique did show promise; not only did the relative throughput of the protocol increase, but the number of receivers at which the protocol performance gains started to level off also increased; improving the range of receivers for which protocol performance was significantly better than the equivalent unicast protocol. As well, the simulation studies clearly demonstrated that the optimal value of the interval used for randomly timed acknowledgment is closely related to the number of receivers.

The multicast block acknowledgment protocol was an attempt to improve on the performance of the multicast stop and wait protocol by reducing the number of acknowledgments required, thereby reducing the collision rate and improving relative throughput. An analysis of this protocol revealed that the theoretical limit on relative throughput was related to the window size of the protocol and would increase as the window size increased. On the other hand, the window size is bounded by the sender's timeout value, memory resources and by the requirement to keep message latency values within a reasonable limit.

Simulation studies showed that the multicast block acknowledgment protocol did indeed perform better than the multicast stop and wait protocol. This improvement

was achieved at the expense of message latency; message latencies in the multicast block acknowledgment protocol include the latency of the entire outstanding window, and were thus significantly longer than the latencies exhibited by the multicast stop and wait protocol. Moreover, the performance of the multicast block acknowledgment protocol improved as the BAP window size increased, again at the expense of message latency.

As in the multicast stop and wait protocol, the multicast block acknowledgment protocol's performance gains reach an asymptotic value as the number of receivers grows; due mainly to the increase in collisions and the resulting increase in message latency. This effect, however, was slowed by increasing BAP window sizes, improving the range of receivers for which the multicast block acknowledgment protocol's performance was significantly better than either multicast stop and wait or the equivalent unicast protocol.

A significant portion of the performance degradation exhibited by the multicast block acknowledgment protocol is due to rising collision rates and ensuing network congestion. The addition of randomly timed acknowledgments to the protocol reduce the rate of collisions, as expected, and therefore improves the relative throughput significantly. As in the stop and wait protocol, the simulation results show the optimal value of the interval used for randomly timed acknowledgments is closely related to the number of receivers, and can be used to improve the scalability of the protocol.

Simulation of both protocols under error conditions showed that both protocols were very sensitive to the presence of errors on the network; a failure rate of 1% was

enough to degrade both protocols' performance by over 100%.

The two simple protocols have limited application due to their poor performance in the presence of errors. Further work on techniques to improve the error-tolerance of these simple protocols is suggested. Specifically, the use of negative acknowledgments may reduce the performance deterioration in the presence of errors by allowing the sender to detect and respond to errors quicker.

Another approach to improve performance is to combine the features of both protocols into a multicast round robin acknowledgment protocol (RRA), in which receivers acknowledge messages in a round robin fashion. The sender sends a multicast message along with an indicator of which receiver should send the acknowledgment. The selected receiver acknowledges this message and all previous messages successfully received since it last sent an acknowledgment. By selecting each receiver in a round robin fashion, the sender will eventually receive acknowledgments from all receivers. This combines the stop and wait approach of sending a message and waiting for a response with the block acknowledgment approach of acknowledging multiple messages with one acknowledgment packet. The result is fewer acknowledgments, and the elimination of contention among receivers acknowledging messages, one of the biggest drawbacks of the two previous protocols.

Another suggested avenue of further study is implementation of the two protocols in a testbed network for comparison with the protocol simulations. The results might suggest ways of improving the simulator model to more accurately reflect real-world conditions. A careful study of errors in such a testbed might suggest refinements to

the error model used in the simulator, as well as give better estimations of typical error rates in real Ethernet networks.

Finally, implementing the three protocols described in the second chapter may provide some insight on other useful techniques for the design of multicast protocols. In particular, the protocol suggested by Erramilli and Singh [14] is heavily dependent on a few timing parameters and is therefore well-suited for study in the simulator.

Bibliography

[1] Decitre, P., Estublier, J., Khider, A., Rousset de Pina, X., Vatton, I., "An Efficient Error Detection Mechanism for a Multicast Transport Service on the DANUBE network", *Proceedings of the IFIP TC 6th International In-Depth Symposium on Local Computer Networks*, pp. 335–347, April 1982.

[2] Mockapetris, P. V., "Analysis of reliable multicast algorithms for local networks", *Proceedings of the 8th data communications symposium (ACM)*, pp. 150–157, October 1983.

[3] Powell, Michael L., Presotto, David L., "PUBLISHING: A Reliable Broadcast Commnication Mechanism", *Operating Systems Review*, Volume 17, Number 5, pp. 100–109, Proceedings of the 9th ACM Symposium on Operating Systems Principles, October 1983.

[4] Wong, J.W., Gopal, G., "Analysis of reliable broadcast in local area networks", *Proceedings of the 8th data communications symposium (ACM)*, pp. 158–163, October 1983.

[5] Segall, A., Awerbuch, B., "A Reliable Broadcast Protocol", *IEEE Transactions on Communications*, Volume COM-31, Number 7, pp. 896–901, July 1983.

[6] Chang, J., Maxemchuk, N.F., "Reliable Broadcast Protocols", *ACM Transactions on Computer Systems*, Volume 2, Number 3, pp. 251–273, August 1984.

[7] Gopal, I.S., Jaffe, J.M., "Point-to-Multipoint Communications over Broadcast Links", *IEEE Transactions on Communications*, Volume COM-32, Number 9, pp. 1034–1044, September 1984.

[8] Gopal, G., Wong, J.W., "Two Protocols for Reliable Broadcast: A Performance Study", *Proceedings of the IEEE GLOBECOM '84*, pp. 11.1.1–11.1.6, November 1984.

[9] Ahamad, M., Bernstein, A.J., "Multicast Communication in UNIX 4.2BSD", *IEEE Proceedings of the 5th International Conference on Distributed Computing Systems*, pp. 80–87, May 1985.

[10] Deering, S.E., Cheriton, D.E., "Host Groups: A Multicast Extension to the Internet Protocol", *RFC-966*, December 1985.

[11] Cheriton, D.R., Deering, S.E., "Host Groups: A Multicast Extension for Datagram Internetworks", *Proceedings of the 9th Data Communications Symposium (ACM SIGCOMM)*, Computer Communications Review, Volume 15, Number 4, pp 172–179, September 1985.

[12] Cheriton, D.R., Zwaenepoel, W., "Distributed Process Groups in the V Kernel", *ACM Transactions on Computer Systems*, Volume 3, Number 2, pp. 77–101, May 1985.

[13] Birman, K.P., Joseph, T.A., "Reliable Communications in the Presence of Failures", *ACM Transactions on Computer Systems*, Volume 5, Number 1, pp. 47–76, February 1987.

[14] Erramilli, A., Singh, R.P., "A reliable and efficient multicast protocol for broadcast networks", *Frontiers in Computer Communications Technology: Proceedings of the ACM SIGCOMM '87 Workshop*, pp. 343–352, Stowe, Vermont, August 1987.

[15] Ramakrishnan, S., Jain, B., "A negative acknowledgment with periodic polling protocol for multicasts over LANs", *Proceedings of the IEEE INFOCOM '87*, pp. 502–511, 1987.

[16] Gopal, I,, Rom, R., "Multicasting to Multiple Groups over Broadcast Channels (Extended Abstract)", *Proceedings of the Computer Networking Symposium*, pp. 79–81, Washington, DC, April 1988.

[17] Hughes, L., "A Multicast Interface for Unix 4.3", *Software Practice and Experience*, Volume 18, Number 1, pp. 15–27, January 1988.

[18] Garcia-Molina, H., Kogan, B., Lynch, N., "Reliable Broadcast in Networks with Nonprogrammable Servers", *Proceedings of the 8th International Conference on Distributed Computing Systems*, San Jose, California, pp. 428–437, June 1988.

[19] Navaratnam, S., Chanson, S., Neufeld, G., "Reliable Group Communication in Distributed Systems", *Proceedings of the 8th International Conference on Distributed Computing Systems*, San Jose, California, pp. 439–446, June 1988.

[20] Garcia-Molina, H., Spauster, A., "Message Ordering in a Multicast Environment", *Technical Report CS-TR-161-88*, Princeton University, June 1988.

[21] Garcia-Molina, H., Kogan, B., "An Implementation of Reliable Broadcast Using an Unreliable Multicast Facility", *Technical Report CS-TR-170-88*, Princeton University, August 1988.

[22] Tanenbaum, Andrew S., *Computer Networks.* Second Edition, Prentice-Hall, 1988.

[23] Chanson, S.T., Neufeld, G.W., Liang, L., "A Bibliography on Multicast and Group Communications" *Operating Systems Review*, Volume 23, Number 4, pp. 20–25, October 1989.

[24] Kaashoek, M.F., Tanenbaum, A.S., Hummel S.F., Bal, H.E., "An Efficient Reliable Broadcast Protocol", *Operating Systems Review*, Volume 23, Number 4, pp. 5–19, October 1989.

[25] Tseung, L.C.N., "Guaranteed, Reliable, Secure Broadcast Networks", *IEEE Network Magazine*, Volume 3, Number 6, pp. 33–37, November 1989.

[26] SAS Institute Inc., *SAS Language: Reference, Version 6, First Edition*, Cary, North Carolina, SAS Institute Inc., 1990.

[27] SAS Institute Inc., *SAS/GRAPH Software: Reference, Version 6, First Edition, Volume 1 & 2*, Cary, North Carolina, SAS Institute Inc., 1990.

[28] Golding, R.A., Long, D.D.E., "Quorum-oriented Multicast Protocols for Data Replication", *Technical Report UCSC-CRL-91-21*, University of California, Santa Cruz, June 1991.

[29] Garcia-Molina, H., Spauster, A., "Ordered and Reliable Multicast Communication", *ACM Transactions on Computer Systems*, Volume 9, Number 3, pp. 243–271, August 1991.

[30] Birman, K., Schiper, A., Stephenson, P., "Lightweight Causal and Atomic Group Multicast", *ACM Transactions on Computer Systems*, Volume 9, Number 3, pp. 272–314, August 1991.

[31] Ellington, R.M., "Supporting Causal Multicast in Distributed Operating Systems: An Experment in Architectural Approaches", *Master's Thesis*, Oregon Graduate Institute of Science and Technology, January 1992.

[32] Lashkari, Y., Ramachandran, V., Malpani, S., Mehndiratta, S.L., "Vartalaap: a Distributed Multicast Communication System", *Software Practice and Experience*, Volume 23, Number 7, pp. 799–811, July 1993.

[33] Whetten, B., Kaplan, S., Montgomery, T., "A High Performance Totally Ordered Multicast Protocol", To be published, *Infocom 1995.*

Appendix A

Sample SAS program

```
* SAS program: saw_recs.sas                                    ;
*
*     Analyzes results of simulation program                    ;
*
*    Input File: SAW_sim1.dat                                   ;
*
*     Output Files:                                                        ;
*        SAW_sim1a_plots.eps (encapsulated PostScript file)               ;
*        SAW_sim1b_plots.eps (encapsulated PostScript file)               ;
*        SAW_sim1_plots.ps (PostScript file)                               ;
*        SAW_sim1a_table.tex (LaTeX source)                               ;
*        SAW_sim1b_table.tex (LaTeX source)                               ;
*        SAW_sim1c_table.tex (LaTeX source)                               ;
*
*    Author: Andrea Segovia                                     ;
*    Date: August 20, 1996                                      ;


* Save datasets;
libname save ".";

* Read input data;
*/
```

```
srecs data set (simulation results):

    Header info (for each simulation run):

    pkts - number of packets for this simulation run
    recs - number of receivers for this simulation run
    data - size of data packet for this simulation run
    ack - size of ack packet for this simulation run
    failrate - failure rate for this simulation run
    reps - number of repetitions performed (always 1)
    proto - protocol identifier, valid values are:
        SAW - stop and wait
        BAP - block acknowledgment
    SAWtimeM = timeout calculation identifier, valid values are:
        for - formula-based timeout calculation
        cons - constant timeout calculation
    SAWretr - base SAW retransmission timeout value
    SAWretrM = retransmission method identifier
        multi - multicast retransmission
        uni - unicast retransmission
    SAWrand - SAW random wait interval
    s_lblat - calculated theoretical lower bound for message latency
        for this simulation run
        formula: (data * 8 + recs * ack * 8)
    s_uathru - calculated theoretical upper bound for absolute
        throughput for this simulation run
        formula: ((recs * data * 8)/(data * 8 + recs * ack * 8)) * 10
    s_urthru - calculated theoretical upper bound for relative
        throughput wrt unicast SAW for this simulation run
        formula: (recs * (data * 8 + ack * 8))/(data * 8 + recs * ack * 8)

    Simulation info (for each multicast packet sent):
        pktID - unique packet ID
        pktSize - packet size
        winSize - size of current window
        startT - time packet sent
        endT - time packet reliably received by all receivers
        timeElap - time elapsed between startT and endT
        timePkt - transmission time per packet sent (timeElap/winSize)
```

```
            retrans - number of retransmissions required
            fail - 1 if packet transmission failed
            s_athru - absolute throughput of multicast SAW
                calculated per packet sent (in Mbps for 10Mbps Ethernet):
                formula: ((recs * data * 8)/timeElap) * 10
            s_rthru - relative throughput sent wrt unicast SAW
                calculated per packet sent:
                formula: (recs * (data * 8 + ack * 8))/timeElap
            lat - latency of packet sent

    srecs_s dataset (summary results - per simulation run):

        recs - number of receivers
        data - data packet size
        ack - ack packet size
        s_lblat - calculated theoretical lower bound for message latency
        s_uathru - calculated theoretical upper bound for absolute throughput
            in Mbps (for 10Mbps Ethernet)
        s_urthru - calculated theoretical upper bound for relative throughput
            wrt unicast SAW
        s_mathru - mean absolute throughput value
        s_mrthru - mean relative throughput value
        s_mlat - mean message latency value
        s_lathru - minimum absolute throughput value
        s_lrthru - minimum relative throughput value
        s_llat - mininum message latency value
        s_hathru - maximum absolute throughput value
        s_hrthru - maximum relative throughput value
        s_hlat - maximum message latency value

/*;

data srecs;
    infile "SAW_sim1.dat";

    input pkts= recs= data= ack= failrate= reps= proto= $CHAR3. @;
    if proto = 'SAW' then do;
        input SAWtimeM= $CHAR3. @;
        if SAWtimeM = 'for' then
```

```
        input SAWtimeI= SAWtime= @;
    else if SAWtimeM = 'con' then
        input SAWtime= @;
    input SAWretr= SAWretrM= $CHAR3. SAWrand=  @;
end;
else if proto = 'GBN' then do;
    input GBNwin= GBNpktM= $CHAR3. @;
    if GBNpktM = 'for' then
        input GBNpktI= GBNpktT= @;
    else if GBNpktM = 'con' then
        input GBNpktT= @;
    input GBNackT= GBNrand= @;
end;

input date= $CHAR24. ;

s_lblat = (data * 8) + (recs * ack * 8);
s_uathru = ((recs * data * 8)/s_lblat) * 10;
s_urthru = (recs * (data * 8 + ack * 8))/s_lblat;

do i = 1 to pkts;
    input pktID pktSize winSize startT endT timeElap timePkt retrans fail;
    drop i;
    s_athru = ((recs * data * 8)/timeElap) * 10;
    s_rthru = (recs * (data * 8 + ack *  8))/timeElap;
    lat = timeElap;
    output;
end;

if _n_ = 1 then do;
    if proto = 'SAW' then do;
        call symput("Lproto", "Stop and Wait Protocol Performance");
        if SAWtimeM = 'for' then do;
            call symput("LStimeM", "formula");
            call symput("LStime", put(SAWtimeI, 7.));
        end;
        else if SAWtimeM = 'con' then do;
            call symput("LStimeM", "constant");
            call symput("LStime", put(SAWtime, 7.));
```

```
            end;
            call symput("LSrand", put(SAWrand, 7.));
            if SAWretrM = 'uni' then
                call symput("LSretrM", "unicast");
            else
                call symput("LSretrM", "multicast");
        end;
    end;

    run;

* Sort input data;
proc sort data=srecs;
    by recs;

* Print first 20 observations - sanity check;
proc print data=srecs (obs=20);

* Calculate means;
proc means noprint data=srecs;
    by recs;
    id data SAWrand
        s_uathru s_urthru s_lblat;
    var s_athru s_rthru lat;
    output out=save.srecs_s
        mean=s_mathru s_mrthru s_mlat
        min=s_lathru s_lrthru s_llat
        max=s_hathru s_hrthru s_hlat;

* Print first 20 observations - sanity check;
proc print data=save.srecs_s(obs=20);

* Produce LaTeX table of summary results;
filename texout1 "SAW_sim1a_table.tex";

data;
    set save.srecs_s end=endtab;
    file texout1;
    if _n_ = 1 then do;
```

```
        put '\begin{tabular}{|r|r|r|r|r|r|r|r|r|}';
        put '\hline';
        put '\multicolumn{1}{|l}{Recs} &';
        put '\multicolumn{1}{|l}{Mean RT} &';
        put '\multicolumn{1}{|l}{Min RT} &';
        put '\multicolumn{1}{|l}{Max RT} &' ;
        put '\multicolumn{1}{|l}{UBRT} &' ;
        put '\multicolumn{1}{|l}{Mean Lat} &';
        put '\multicolumn{1}{|l}{Min Lat} &';
        put '\multicolumn{1}{|l}{Max Lat} &' ;
        put '\multicolumn{1}{|l|}{LBL} \\' ;
        put '\hline';
        put '\hline';
    end;
    if mod(recs, 5) = 0 then do;
        put recs          3.     ' & '
            s_mrthru     8.3     ' & '
            s_lrthru      8.3     ' & '
            s_hrthru      8.3     ' & '
            s_urthru     8.3     ' & '
            s_mlat          8.      ' & '
            s_llat          8.      ' & '
            s_hlat          8.      '&'
            s_lblat         8.   '\\';
        put '\hline';
    end;
    if endtab then do;
        put '\hline';
        put '\end{tabular}';
    end;
    run;

* Produce LaTeX table of summary results including absolute throughput;
filename texout2 "SAW_sim1b_table.tex";

data;
    set save.srecs_s end=endtab;
    file texout2;
    if _n_ = 1 then do;
```

```
        put '\begin{tabular}{|r|r|r|r|r|r|r|r|r|}';
        put '\hline';
        put '\multicolumn{1}{|l}{Recs} &';
        put '\multicolumn{1}{|l}{Mean AT} &';
        put '\multicolumn{1}{|l}{Min AT} &';
        put '\multicolumn{1}{|l}{Max AT} &' ;
        put '\multicolumn{1}{|l}{UBAT} &' ;
        put '\multicolumn{1}{|l}{Mean RT} &';
        put '\multicolumn{1}{|l}{Min RT} &';
        put '\multicolumn{1}{|l}{Max RT} &' ;
        put '\multicolumn{1}{|l|}{UBRT} \\' ;
        put '\hline';
        put '\hline';
    end;
    if mod(recs, 5) = 0 then do;
        put recs          3.     ' & '
            s_mathru      8.3    ' & '
            s_lathru       8.3    ' & '
            s_hathru       8.3    ' & '
            s_uathru      8.3    ' & '
            s_mrthru     8.3    ' & '
            s_lrthru      8.3    ' & '
            s_hrthru      8.3    ' & '
            s_urthru     8.3    ' \\ ';
        put '\hline';
    end;
    if endtab then do;
        put '\hline';
        put '\end{tabular}';
    end;
    run;

* Produce LaTeX table of summary results for AT, RT and lat;
filename texout3 "SAW_sim1c_table.tex";

data;
    set save.srecs_s end=endtab;
    file texout3;
    if _n_ = 1 then do;
```

```
        put '\begin{tabular}{|r|r|r|r|r|r|r|}';
        put '\hline';
        put '\multicolumn{1}{|l}{Recs} &';
        put '\multicolumn{1}{|l}{Mean AT} &';
        put '\multicolumn{1}{|l}{UBAT} &' ;
        put '\multicolumn{1}{|l}{Mean RT} &';
        put '\multicolumn{1}{|l|}{UBRT} &' ;
        put '\multicolumn{1}{|l}{Mean Lat} &';
        put '\multicolumn{1}{|l|}{LBL} \\' ;
        put '\hline';
        put '\hline';
        put '\hline';
    end;
    if mod(recs, 5) = 0 then do;
        put recs          3.     ' & '
            s_mathru      8.3    ' & '
            s_uathru      8.3    ' & '
            s_mrthru     8.3     ' & '
            s_urthru     8.3     ' & '
            s_mlat          8.      ' & '
            s_lblat         8.   '\\';
        put '\hline';
        put '\hline';
    end;
    if endtab then do;
        put '\hline';
        put '\end{tabular}';
    end;
    run;

* Produce PostScript and encapsulated PostScript plots;
title1 justify=center "&Lproto";

axis1 label=(font=swiss "Message Latency");
axis2 label=(font=swiss "Relative Throughput");
axis3 label=(font=swiss "Number of Receivers");
axis4 label=(font=swiss "Absolute Throughput");

footnote1 justify=left "SAW timeout calculation method = &LStimeM";
```

```
footnote2 justify=left "SAW timeout value = &LStime";
footnote3 justify=left "SAW retransmission method = &LSretrM";
footnote4 justify=left "SAW random wait interval = &LSrand";

symbol1 color=black interpol=hilotj;
symbol2 color=black interpol=boxt00;
symbol3 color=black line=1 interpol=join;
symbol4 color=black line=2 interpol=join;


filename psout "SAW_sim1_plots.ps";
goptions device=ps gsfname=psout gsfmode=replace ftext=swiss;

proc gplot data=srecs;

    title2 justify=center "Message Latency vs. Number of Receivers";
    title3 justify=center "Hi-Mean-Lo Simulation Results";
    title4 justify=center "Theoretical Lower Bound Results";
    plot lat * recs=1 s_lblat * recs=4 /
        overlay
        vaxis=axis1
        haxis=axis3;

    run;

    title2 justify=center "Message Latency vs. Number of Receivers";
    title3 justify=center "00-25-50-75-100 Percentile Simulation Results";
    title4 justify=center "Theoretical Lower Bound Results";
    plot lat * recs=2 s_lblat * recs=4 /
        overlay
        vaxis=axis1
        haxis=axis3;

    run;

    title2 justify=center "Relative Throughput vs. Number of Receivers";
    title3 justify=center "Hi-Mean-Lo Simulation Results";
    title4 justify=center "Theoretical Upper Bound Results";
    plot s_rthru * recs=1 s_urthru * recs=4/
```

```
        overlay
        vaxis=axis2
        haxis=axis3;

    run;


    title2 justify=center "Relative Throughput vs. Number of Receivers";
    title3 justify=center "00-25-50-75-100 Percentile Simulation Results";
    title4 justify=center "Theoretical Upper Bound Results";
    plot s_rthru * recs=2 s_urthru * recs=4 /
        overlay
        vaxis=axis2
        haxis=axis3;

    run;

    title2 justify=center "Absolute Throughput vs. Number of Receivers";
    title3 justify=center "Hi-Mean-Lo Simulation Results";
    title4 justify=center "Theoretical Upper Bound Results";
    plot s_athru * recs=1 s_uathru * recs=4/
        overlay
        vaxis=axis4
        haxis=axis3;

    run;

    title2 justify=center "Absolute Throughput vs. Number of Receivers";
    title3 justify=center "00-25-50-75-100 Percentile Simulation Results";
    title4 justify=center "Theoretical Upper Bound Results";
    plot s_athru * recs=2 S_uathru * recs=4 /
        overlay
        vaxis=axis4
        haxis=axis3;

    run;

filename epsout1 "SAW_sim1a_plot.eps";
```

```
goptions device=psepsf gsfname=epsout1 gsfmode=replace ftext=swiss;
proc gplot data=srecs;

    title2 justify=center "Message Latency vs. Number of Receivers";
    title3 justify=center "Hi-Mean-Lo Simulation Results";
    title4 justify=center "Theoretical Lower Bound Results";
    plot timeElap * recs=1 s_lblat * recs=4 /
        overlay
        vaxis=axis1
        haxis=axis3;

    run;

filename epsout2 "SAW_sim1b_plot.eps";

goptions device=psepsf gsfname=epsout2 gsfmode=replace ftext=swiss;
proc gplot data=srecs;

    title2 justify=center "Message Latency vs. Number of Receivers";
    title3 justify=center "00-25-50-75-100 Percentile Simulation Results";
    title4 justify=center "Theoretical Lower Bound Results";
    plot timeElap * recs=2 s_lblat * recs=4 /
        overlay
        vaxis=axis1
        haxis=axis3;

    run;

filename epsout3 "SAW_sim1c_plot.eps";

goptions device=psepsf gsfname=epsout3 gsfmode=replace ftext=swiss;
proc gplot data=srecs;

    title2 justify=center "Relative Throughput vs. Number of Receivers";
    title3 justify=center "Hi-Mean-Lo Simulation Results";
    title4 justify=center "Theoretical Upper Bound Results";
    plot s_rthru * recs=1 s_urthru * recs=4/
        overlay
        vaxis=axis2
```

```
        haxis=axis3;

    run;

filename epsout4 "SAW_sim1d_plot.eps";

goptions device=psepsf gsfname=epsout4 gsfmode=replace ftext=swiss;
proc gplot data=srecs;

    title2 justify=center "Relative Throughput vs. Number of Receivers";
    title3 justify=center "00-25-50-75-100 Percentile Simulation Results";
    title4 justify=center "Theoretical Upper Bound Results";
    plot s_rthru * recs=2 s_urthru * recs=4 /
        overlay
        vaxis=axis2
        haxis=axis3;

    run;


filename epsout5 "SAW_sim1e_plot.eps";

goptions device=psepsf gsfname=epsout5 gsfmode=replace ftext=swiss;
proc gplot data=srecs;

    title2 justify=center "Absolute Throughput vs. Number of Receivers";
    title3 justify=center "Hi-Mean-Lo Simulation Results";
    title4 justify=center "Theoretical Upper Bound Results";
    plot s_athru * recs=1 s_uathru * recs=4/
        overlay
        vaxis=axis4
        haxis=axis3;

    run;

filename epsout6 "SAW_sim1f_plot.eps";

goptions device=psepsf gsfname=epsout6 gsfmode=replace ftext=swiss;
proc gplot data=srecs;
```

```
title2 justify=center "Absolute Throughput vs. Number of Receivers";
title3 justify=center "00-25-50-75-100 Percentile Simulation Results";
title4 justify=center "Theoretical Upper Bound Results";
plot s_athru * recs=2 s_uathru * recs=4 /
    overlay
    vaxis=axis4
    haxis=axis3;

run;
```