MULTICAST FLOW CONTROL ON LOCAL AREA NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Guerney Douglass Holloway Hunt

May 1995

MULTICAST FLOW CONTROL ON LOCAL AREA NETWORKS

Guerney Douglass Holloway Hunt, Ph.D.
Cornell University 1995

The primary LAN technologies in use today — ethernet, FDDI, and token-ring — all provide hardware support for broadcast and multicast capability. However, distributed systems have traditionally used unicast messaging exclusively, even when multicast communication patterns arise. As the number of distributed applications grows, the load on networks caused by unnecessary unicasts will increase. In addition, for some applications performance and group size are limited by using unicast technologies for multicast. If multicast technologies are exploited, the network load caused by "redundant" packets will be reduced. Exploiting multicast will also improve the performance and scalability of some distributed applications. However, as distributed systems move towards exploiting multicast, multicast flow control protocols are becoming more important. Finding a general, effective solution for multicast flow control will help facilitate the exploitation of the multicast primitives provided in LANs and Wide Area Networks (WANs).

This dissertation presents the results of an investigation into multicast performance on local area networks. An analysis of multicast flow control is presented which distinguishes between *rate reservation* and *rate control*, followed by a discussion of the major design issues associated with multicast flow control and a presentation of a proposed protocol family. The proposed protocol family is based on send-rate control. An unreliable and a reliable multicast flow control protocol based on the proposed family are presented. A study of the performance of these protocols is also presented. This dissertation concludes with an investigation into how well the proposed reliable multicast flow control protocol performs when used to disseminate messages. This dissertation argues that direct rate control has merit as a technique for multicast flow control on local area networks.

Biographical Sketch

Guerney Douglass Holloway Hunt was born January 1, 1952, in Nashville, Tennessee. His parents' divorce five years after his birth led to numerous moves (more than twenty) before he entered 9th grade. As a result his childhood was spent in Nashville, Tennessee; Orlando, Florida; and Ann Arbor, Michigan, where he attended several primary and middle schools. After attending 9th, 10th and 11th grades in Marquette, Michigan, he graduated from Marquette Senior High School in the spring of 1969. He went on to attend Michigan Technological University in Houghton, Michigan. At MTU he worked his way through college primarily through employment in the digital computer lab, and this was the beginning of a long term interest in computers and computer science. He graduated from Michigan Tech in the spring of 1973 with a bachelor of science in Mathematics. In the fall of 1973 he enrolled as a Ph.D. candidate in the Computer Science Department at Cornell University studying theoretical computer science (complexity and automata). In the spring of 1975 he received a master of science in Computer Science from Cornell after successfully negotiating the qualifying examinations, took a leave of absence from his graduate studies at Cornell, and coincidentally was introduced to Harriet Haas. Between June of 1975 and April

1982 he worked for the NCR corporation at 950 Danby Road in Ithaca, New York. He held several positions as a programmer and project engineer while working for NCR, the most interesting of which was as a corepresentative for NCR on the ANSI basic language committee for one year. His time at NCR provided a pragmatic and practical view into the state of computer programming as contrasted with computer science as studied at Cornell. He was married to Harriet Haas (the love of his life) in August of 1981. In April of 1982 he resigned from NCR and started working for the IBM corporation in Endicott, New York. He started at IBM as a Senior Associate Engineer and eventually became an Advisory Programmer. While at IBM his most interesting assignments were a year as VM's communications architect, followed by an assignment as technical team leader for 370 LAN server development. He also received some awards from IBM, including an Endicott Programming Lab Team award in 1987 and an IBM Outstanding Innovation award in 1990. In the spring of 1990 he received an IBM Resident Study Fellowship. With his IBM fellowship, wife, and four children he re-enrolled as a graduate student in the Computer Science Department at Cornell University in the fall of 1990 to pursue doctoral studies in fault-tolerant distributed computing. In July of 1993 he resigned from IBM, shortly before the end of his resident study appointment, in order to continue his doctoral studies. He is successfully completing those studies as of January 1995.

To those on whose shoulders I stand, and to my parents.

Acknowledgements

First of all, I want to thank Professor Kenneth P. Birman for serving as the chair of my committee and challenging me with questions which eventually precipitated this work. Ken's support and unique perspective on computer science research has been invaluable in the progress of this work.

I would also like to thank Professors Anil Nerode and Dexter Kozen, who also served on my committee and provided additional motivation, insight, and wisdom.

This work was supported in part by the National Science Foundation, under grant number CDA-9024600, and by ONR grant N00014-92-J-1866. The NSF grant enabled the completion of this dissertation. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or of the Office of Naval Research.

I gratefully acknowledges the use of the facilities of the Program of Computer Graphics at Cornell University. The Program of Computer Graphics receives support from the NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219), and equipment grants from the Hewlett Packard Corporation, on whose workstations part of this research was conducted.

I also gratefully acknowledges equipment grants from the Hewlett Packard Corporation to the Isis project and the Computer Science Department Undergraduate Computing Labatory. This equipment was used for some experiments during the course of this work. I also very gratefully acknowledge the IBM Corporation's equipment grant to the Isis project. The IBM workstations were my home base during my graduate work, and the initial test system and algorithms were developed using them.

I want to thank the IBM corporation for their resident study fellowship and their Endicott Transition Opportunity Program, which provided support for a large part of my doctoral studies. I especially want to thank Richard E. Kizis, who championed my application for the resident study program. Without his efforts the initial funding for this degree would not have occurred.

Sir Isaac Newton said "If I have seen further than other men it is by standing on the shoulders of giants." His remark is also true of our lives. I therefore acknowledge the vision and work of Fredrick Douglass, Booker T. Washington, George Washington Carver, and Martin Luther King Jr., whose vision and efforts contributed greatly to the opportunities available to me.

I also owe a debt to Cullen King Hunt, who moved from Hendersonville, N.C., to Raleigh, N.C., in part so his children would have an opportunity for a college education. Four of his six children graduated from Saint Augustine's College, including my father Robert Walker Hunt, who at 20 went to Columbia University in New York City to work on a Ph.D. in mathematics. After making some progress in mathematics at Columbia, he enrolled in Meharry Medical College in Nashville, Tennessee, from which he graduated with a D.D.S.

I am also indebted to Henry Hartsfield Holloway, who broke the bonds of slavery through education and encouraged his son John Wesley Holloway in his educational pursuits. John Wesley Holloway graduated from Fisk University in Nashville, Tennessee, and went on to become a preacher, linguist, and author. He also instilled a desire for education in his children, all of whom also graduated from Fisk University. One of his sons, Guerney Douglass Holloway, went on to get a Masters degree from Harvard University and to do postgraduate work at the University of Chicago before graduating from Meharry Medical College with an M.D. He taught at Meharry for forty years, eventually becoming head of the Department of Clinical Pathology. During that time he also nurtured the natural inquisitiveness of his grandson Guerney Douglass Holloway Hunt (III) and instilled in him a love for detailed rigorous investigation. Dr. Holloway and his wife inspired in their children an appreciation and desire for education. His oldest daughter, Nareda Winifred Holloway, after graduating from Fisk University at the age of 16 and doing graduate work in library science, went on to work at Meharry, where she met and married a young man from Raleigh, N.C., Robert Walker Hunt.

I want to thank and acknowledge the input of Mark Wood, Dawson Dean, Michael Kalantar, Cliff Krumvieda, and Thomas Bressoud, with whom I have discussed various parts of this work during its development.

No man is an island. I also wish to thank my friends, associates, and mentors who have provided support, wisdom, and insight during the progress of my career and education. In no particular order from this group I want to especially thank Hubert

L. Hunzinger, James W. Coar, Gary and Mae Ellen Fick, Shigeki Morimoto, Earl and Miriam Hilton, Tim Metcalf, Ray and Esther Zimmerman, the Isis research group, the EE and CS prayer group, the Thursday noon prayer group, the LAN sharks, Ray and Gretchen Crognale, Tom and Sally Brown, Paul Simonet, George and Margie Hillenbrand. Russ and Lisa Qualls, Jim and Nancy Crawford, and Peg Hardesty.

I want to thank Joseph Martin and the Writing Workshop for instruction in writing and increasing my confidence as a writer. I also want to thank Penny J. Beebe and the Engineering Communications Program for counsel and guidance in technical writing. I owe a special debt of gratitude to Lucy Cunnings, who edited my rough manuscript and as a result improved my overall writing skills.

I also wish to thank my wife and five children, who have provided vision, wisdom, and motivation for the completion of this degree.

Finally and most importantly I want to thank God. Without His mercy, grace, and work in my life this dissertation would not have been begun let alone been completed.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ATM Asynchronous Transfer Mode

CDDI Copper Distributed Data Interface

CSMA/CD Carrier Sensed Multiple Access with Collision Detection

DLC Data Link Control

FDDI Fiber Distributed Data Interface

GDLC Generic Data Link Control

IP Internet Protocol

ISO International Standards Organization

LAN Local Area Network

LLA Link Level Access

MAC Media Access Control

OSI Open Systems Interconnect

SAP Service Access Point

SNA System Network Architecture

UDP User Datagram Protocol

WAN Wide Area Network

Chapter 1

Introduction

Computers and computer networks have become an increasingly important part of our society over the last forty years. Interconnecting computers for the direct exchange of information is more efficient than exchanging information through secondary media. Traditionally these interconnections have been made with point-to-point links, but point-to-point networks require a separate connection for each system which is directly interconnected. In addition, messages sent to systems for which there is no direct connection require one or more intermediate systems.

As the number of computers needing to be interconnected has increased, the benefits of directly interconnecting computers and the inefficiencies of point-to-point networking have driven the development of broadband technologies for computer interconnection. Networks based on broadband technology are called local area networks or LANs. In this type of network each computer is directly connected to the network and can use the network to pass a message to any other computer on the network without requiring an intermediate system. With broadband technology two other capabilities have been added, broadcast and multicast. Broadcast is sending a message to every computer on the network, and multicast is sending a message to a subset of the computers on the network.

The advantage of having computers exchange data directly precipitated the first networks. As the number of networks grew it became more important to interconnect these networks, which led to the development of the Internet. Today many computers (workstations, PCs, and others) are connected to one another via local area networks and internet. Many are also connected via proprietary isolated networks.

1.1 Flow Control

One of the main problems of interconnecting computers is unreliable networks. Computer networks can corrupt or lose data, and they generally do not guarantee that messages arrive. A second main problem is that each computer has finite resources; therefore, even with a reliable network it is possible for one computer to send more messages than the intended receiver can handle. Finally, as more and more interconnected computers send data over the network, which is a finite resource, the network can become congested or overloaded.

Flow control was initially developed to address these problems in point-to-point networks. As computer and network speeds increase and multicast technologies become more widely available, flow control becomes a more difficult and challenging problem.

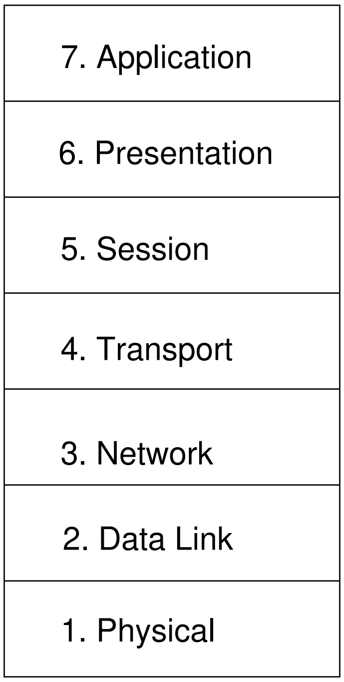

Figure 1.1: Seven-layer OSI model of communications developed by ISO.

1.2 Models of Communication

The seven-layer OSI model has become the standard one for modeling communication systems. These layers are shown in figure 1.1. In the case of two communicating computer systems, the model exists inside each system. Conceptually, each layer of this model exchanges information with the corresponding layer in the other system. Most existing communication systems have been mapped onto the OSI model, as it provides a useful framework for understanding computer communications. For example, in the discussion on losses below, the application is layer 7, layers 6 to 4 are the operating system, layers 3 and 2 are the adapter, and layer 1 is the network,

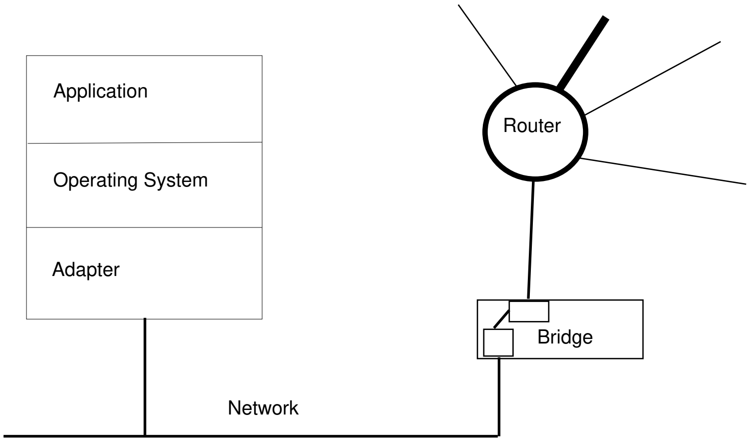

Figure 1.2: Overview of a computer network illustrating where messages can be lost.

bridge, etc.

Flow control can be applied on any level of this model. The semantics of flow control change depending on the level at which flow control is implemented. Even though the semantics change, a given technique of flow control can be applied at different levels, although, depending on the system in which it is implemented, it might be most appropriate at a single level. With respect to the OSI model, flow control is usually placed at level 2, 3, 4, or 5.

1.3 Where Losses Occur

Looking at an abstraction of one side of a communicating application, figure 1.2 illustrates that there are several places where communication losses can occur. Within each part of a communicating application there are three distinct places where errors can occur—within the application, in the operating system, or in the communications adapter and network. The application can fail to send or fail to receive a message. Messages can be lost by the operating system or the communications adapter. Even if all of these components function properly, messages can be corrupted (and therefore lost) in transit on the wire.

Operating systems generally lose packets when the system is overloaded and there are no internal buffers to receive packets from the adapter. The communications adapter can lose packets because of either hardware or software problems. Because the lower level of communications is implemented as microcode (software) on many adapters, there are buffer management problems similar to those of operating systems within the adapter. If an adapter's buffers are full when an incoming packet arrives, then a packet will be lost. Packets can also be corrupted on the wire (with low probability). Finally, there are also bridges and routers which can become overloaded and drop packets.

1.4 Distributed Systems

The development of microprocessors and microprocessor-based computers has made wide-scale distributed computing more economically feasible. As microprocessor-based computers have proliferated, they have been interconnected with LANs. However, the underlying communication systems which were developed for point-to-point networks have continued to use the point-to-point paradigm, even though multicast technology is available.

One of the technologies for building distributed systems relies on the group communication paradigm. In this paradigm communicating computers are collected in groups which multicast messages to one another. The proliferation of group computing has placed an increased load on LANs, because many of these systems rely on point-to-point communication for their multicast.

The desire to build distributed systems beyond the scope of a single LAN, the desire to use the internet as a broadcast media, and the desire to make optimal use of the internet have precipitated the development of multicast protocols for internet. However, managing losses and overrun is more complex for multicast than for unicast communications, and as distributed systems convert to the direct use of multicast (on LANs and WANs), multicast flow control protocols are becoming increasingly important. Understanding multicast and multicast flow control on LANs is a key step in optimally exploiting our existing networks.

1.5 Summary

Multicast flow control is an important aspect of enabling communication systems to exploit multicast technology efficiently and effectively. Chapter 3 of this dissertation starts with a study of the performance of multicast from the application level. The results of this performance study provide the basic motivation for the approach to flow control that is proposed in this dissertation. In chapter 4 the issues surrounding multicast flow control are discussed in more detail and an approach to multicast flow control based on the performance study of chapter 3 is presented. Finally, in chapter 5 this dissertation studies the performance of a reliable and an unreliable multicast flow control protocol based on the proposal of chapter 4. Chapter 5 concludes with a study of how well the reliable protocol performs when used for message dissemination. A summary of the results presented in this dissertation is presented in chapter 6. The appendices at the end of the dissertation provide some additional information which may be useful to the reader.

Chapter 2

Related Work

This chapter presents a few basic definitions and an overview of previous work in the general area of multicast and group communications.

There are three basic types of communication used between computers connected by LANs or WANs: point-to-point, broadcast, and multicast. *Point-to-point* is one-to-one communication, one sender to one receiver. Broadcast and multicast are both one-to-many communication. The difference between these two is that *broadcast* is one to everyone and *multicast* is one to a proper subset of everyone.

Group communication is a environment where individual processes join and leave groups. A group communication system provides primitives for sending and receiving messages, failure detection, and membership. Additional services such as ordered, reliable, or virtually synchronous messages may be provided. Messages which are sent are sent to every member of the group, and messages which are received are assumed to have been sent to every member of the group [Bir93,Mul93,BvR94].

Multicast and group communication has been an active area of research for some time[CNL89]. In some of the earlier work, multicast was not differentiated from group communication, but more recently it has become common to distinguish group communication (a logical abstraction) from multicast (a technology used for communications). Multicast is a communications technology and therefore can be used outside the context of group communication. For example, the publish-subscribe paradigm used in internet radio uses unreliable multicast without all of the structure of a group communication system. Multicast and group communication were initially linked because group communication naturally exploits multicast technology. Even though much work has been done on multicast and group communication, not many authors have addressed the general multicast flow control issue.

2.1 Link Level Protocols

A number of authors have written about link level response protocols for broadcast networks[DEK+82,GJ84,Lee91,WS88,WS89,Yam90,WS91]. Most of this work is analytical in nature and concentrates on managing responses to multicast request at the link level. No general flow control issues have been addressed. The error correction techniques present in this work are similar to those analyzed by Mockapetris [Moc83].

2.2 Systems Supplying Multicast or Group Communication

There have been a few systems which implemented multicast or group communication interfaces. The Amoeba system[KT91a,KT91b] has a group communication interface, but it does not address flow control issues. Frank[FWB85] implemented a multicast communication mechanism without addressing flow control issues. Deering[Dee89] has proposed extensions to IP for multicasting which do not address flow control. Transis [ADKM92] contains a flow control mechanism which has been described as a generalization of sliding window for broadcast communications.

XTP[DFW90,Pro92] implements multicast with flow control and rate control as separate functions. Rate control is primarily used as a means of addressing multicast implosion and not as a method of direct flow control. Multicast flow control in XTP is accomplished with a window-based method that is the same for unicast. For multicast, a lower level in XTP coalesces information from members of the multicast set into a single response, thereby allowing multicast communication to be handled by their point-to-point flow control technique.

Isis [BvR94,BC94] supports IP multicast and has a multilevel window method for multicast flow control. Horus [vRHB94] also supports multicast and has a credit-based method for flow control.

2.3 Specialized Protocols

Some specialized multicast flow control protocols have been proposed. Naryan[Nar90] proposed a protocol which uses a window-based method with a single designated acknowledger. It is designed for operating in a single LAN environment and is directed at enabling multicast for replicated databases. The protocol assumes only one transmitter per group. Earlier, Crowcroft[CP88] proposed a window-based protocol aimed at a similar environment which provides reliable multicast between a client and a number of servers. The proposed protocol has been modeled using probabilistic analysis. Armstrong, Freier and Marzullo [AFM92,FM90] have proposed MTP, which is a flow controlled atomic transport protocol for the internet.

2.4 Multicast Communication

Some papers have addressed multicast communications on a wider scale[JQ91,Ngo91] without discussing any flow control issues. Singh[SE90] discusses design and modeling issues for protocols on networks which support broadcast or multicast communications. There has also been a paper[JSW91] proposing a protocol for large group multicast; it presents some simulation results of the proposed protocol.

2.5 Related Studies

Van Jacobson's paper on congestion avoidance and control [Jac88] is similar to the work that was done for this dissertation. Jacobson analyzed congestion issues for point-to-point interconnected networks (Internet) using TCP/IP. His paper reports on a number of techniques that were successfully developed and utilized to solve flow control problems in TCP/IP. Jacobson's analysis did not include LANs, multicast communication, or any of the associated flow control issues. Nevertheless, the principles defined (for example, slow start and exponential backoff) remain the same and are exploited in this present work.

Pingali and Kurose [PTK94] have recently published an analytical analysis of sender-initiated (positive acknowledgment) versus receiver-initiated (negative acknowledgment) multicast protocols. The sender-initiated protocol they model is related to the multicast flow control protocol proposed in this dissertation. However, their asynchronous protocol does not dynamically adjust the send rate. This work is analytical in nature, and the proposed protocols were not actually implemented or calibrated to existing protocols. Nevertheless, under the assumptions of their model they show that receiver-initiated protocols will outperform sender-initiated protocols. The focus of their analysis is different from the focus of this dissertation. In addition, in order to get numerical results, they made simplifying assumptions that do not correspond to the environment that this dissertation found. However, their insight may be useful in improving the performance of the protocols proposed in this dissertation.

Previous work which examined rate-based flow control methods for wide area networks combined all rate-based flow control methods together under the term rate control and assumed they were all hop-by-hop methods [Jai92]. This present dissertation distinguishes between rate control and rate reservation (the distinction is made clear in section 4.2). The methods proposed in this dissertation are called send-rate control, and the hop-by-hop methods reviewed in the cited work are called rate reservation. Distinguishing between hop-by-hop rate reservation methods and other rate-based methods is important, because conclusions based on the analysis of hop-by-hop rate reservation do not necessarily apply to other rate-based methods. In addition, this dissertation focuses on the use of rate-based methods on local area networks, an area where the hop-by-hop analysis degenerates, in part because there are no hops.

Chapter 3

Multicast Performance

This chapter studies primitive multicast performance (with no superimposed flow control protocol) with the goal of gaining insight that will be used in building flow control protocols for multicast. A general purpose flow control protocol will not make assumptions about communication patterns or network topology. There are two basic approaches to this problem: extending an existing flow control protocol or designing a new protocol. Existing general purpose flow control protocols were developed for point-to-point communication networks. However, multicast is not point-to-point communication but is point-to-multipoint communication. Therefore it was not clear whether extending an existing flow control method would effectively/efficiently allow us to exploit multicast hardware, and consequently new protocol development was chosen.

Flow control protocols are designed based on a communications pattern and an understanding of what happens when messages are lost because the network is loaded

[Jac88]. One could gain insight into what happens when the network becomes loaded by collecting existing studies and analyzing the data, by building a mathematical model and simulating performance, or by measuring an existing system.

With regard to prior studies, no prior work on multicast was found which could be analyzed and/or used to direct the design of a protocol or calibrate a simulation for our purposes.

Simulation is a valuable technique which can be used to gain considerable insight into issues affecting computer and communication systems. In order to build a model of a real system the modeler should understand how the system works. When the parameters of a problem are understood, a model calibrated to an observable system can be built. Such a model can be used to determine how sensitive the problem is to changes in the individual parameters. It can also be used to find answers to questions about the system. However, unless a simulation is calibrated to an observable system, it can produce results which are of limited use. These results meet the parameters of the model, but since the model has no basis in a real system, they do not give additional insight into solving problems in real networks or distributed systems. Since there were no data with which to calibrate a model, simulation was ruled out because it could easily lead to unrealistic conclusions.

This study starts by measuring an existing system because no prior data was found that could be used as a basis for a design or used to calibrate a simulation.

Whether the network is a LAN or WAN, the basic problem is the same: messages are lost. Multicast hardware is more widely available on LANs, and LANs are simpler to work with than WANs. Therefore the initial investigation was done on a single

LAN with hardware multicast support and focused on loss characteristics.

The goal of this chapter is to gain insight into the raw performance of multicast hardware. This insight will be used in chapter 4 to design a flow control protocol for multicast. Raw performance is defined as how well an application can perform using multicast without flow control or error correction. Flow control is needed in point-to-point communications, so it is expected that it will be needed for multicast communication. Answering the following questions will give us the background information necessary to design a flow control protocol for multicast:

1. What is the performance of applications sending multicast?

2. What is the performance of applications receiving multicast?

3. What happens when the network is loaded and multicast packets are lost?

4. Do packets normally arrive at all destinations?

In order to answer these questions, performance will be measured in terms of messages per second and K bytes per second. These measurements will verify whether or not senders using multicast can overrun receivers. They will indicate when overrun occurs and perhaps why overrun occurs. Studying the raw performance of multicast will give insight into what types of flow control should be used for multicast.

3.1 Methodology

Measuring the performance of a distributed application is a challenging task. Ideally all one has to do is write a program that characterizes multicast communication patterns and then measure its performance. However, there is no single communication pattern that characterizes all multicast applications. Therefore fixed length messages will be used to measure multicast performance.[1]

The performance of a distributed application can also be affected by the following:

1. Network issues.

2. Performance of the processors used by the distributed application:

   (a) Load

   (b) Operation system

   (c) Communication protocols

3. Measurement software.

3.1.1 Network

Characterizing the performance of multicast is a communication-intensive task. For communication-intensive programs, performance decreases as the network load increases, because there is less bandwidth available for the measuring program to use. Therefore in order to accurately characterize multicast, it is necessary to distinguish between network load caused by multicast and network load cause by other traffic. To insure that the tests identified issues associated with multicast performance, the initial tests were run on isolated and very lightly loaded networks.

A second network-related issue that could possibly affect performance is whether the network is a ring or CSMA/CD. Both of these physical transport methods naturally support multicast. However, the architectures are different, and therefore it is

[1]This is a standard technique for measuring communications performance.

not clear if multicast performance is the same on both types of networks. If there is a transport-layer-independent characterization of multicast performance and failure, then flow control management is simpler (one less variable). Therefore our tests were run on both ethernet (CSMA/CD) and token-ring in order to differentiate between the general characteristics of multicast and properties that might be associated with a specific network architecture.

3.1.2 Processor Performance

The clock rate of the processor directly affects performance. Slower processors cannot send as many messages per unit of time as faster processors. In addition, faster processors can more easily overrun slower processors. The test system used to characterize multicast performance will be a distributed system. Running these tests on different configurations of processors and operating system releases, or on differently loaded processors, would complicate the task of isolating the attributes of multicast. Depending on the precise test configuration, the need for flow control could be accentuated or eliminated. Therefore, to simplify the testing and identification of multicast characteristics, the tests were run on dedicated matched systems and lightly loaded matched systems.

When dedicated systems were used, the test application was the only application running on the system other than the operating system and necessary support (file system, X, etc.), and there were no other users logged into the systems during the test. On lightly loaded systems, the tests were run during off-peak hours when there were generally no other users on the systems. In both environments the tests were run on matched systems; these were identical systems from the same vendor, and they were running the same operating system release. Systems from both IBM and HP were used to take the initial measurements. This helped us identify anomalies introduced by a particular vendor's hardware or software implementation. Using dedicated and lightly loaded matched systems eliminated processor performance as a variable in the testing environment.

3.1.3 Measurement System

In order to test multicast performance without flow control or error correction it is necessary to bypass existing communications protocols. On the IBM systems the GDLC[Int91a,Int91b] interface for token-ring was used. Our test packets were sent as Network Data. According to the IBM manual, "Network data must contain the entire MAC layer packet headers so that the packet can be sent without the data link control (DLC's) intervention. GDLC only provides a pass-through function for this type of write." [Int91c, page 3-80] This interface provides a fairly direct path to the device driver. Therefore the token-ring test builds MAC headers and sends them to the device driver for transmission over the network.

Token-ring hardware supports group and functional addressing, both of which provide a multicast capability. However, in the version of AIX which was used for these tests, applications did not have direct access to group addressing, and therefore group addressing was not used. Applications could use functional addressing, but it is specific to token-ring and therefore might have unnecessarily affected the comparability of our results. Using an interface that was fairly similar between the different networks made the results for the two networks more comparable. Therefore, for the initial tests multicast was simulated by using broadcast packets to different SAPs. The performance of broadcast should be identical to the performance of multicast except that broadcast affects all systems attached to a LAN whereas multicast affects only those systems "listening" to the particular multicast address.

On the HP systems the test used the LLA interface, which provided direct access to the device driver. The level of the interfaces was identical and the commands available were very similar, so it was easy to modify the test system and run the same test on HP 720's over ethernet. The LLA interface did not require the test system to build MAC headers.

Using LLA, multicast was simulated by broadcasting to different SAPs, as was done on the IBM systems. The protocol stacks below the interfaces were not identical; therefore the results are primarily comparable for trends and only somewhat comparable for the efficiency of the implementation of the underlying protocols.

In order to measure application level performance of multicast hardware from the end-to-end perspective, it is not necessary to remove absolutely everything between the test application and the multicast hardware. It is only necessary to use low level interfaces which allow direct access to the hardware. When this is done there will be no flow control or error correction imposed between the test system and the hardware. Therefore the test system will still be exposed to operating system losses and as a result will more accurately represent application level performance.

Our test system did not change much between the two environments, HP and IBM. The only parts that changed were the interface to the device driver and some constants related to the transport layer (ethernet or token-ring).

The goal of the measurement system is to take accurate measurements. To do this it must not introduce unrelated network activity and must not introduce unnecessary system load while measurements are being taken. The test system does not take any action which would cause messages to be transmitted while test messages are being generated. As mentioned earlier, the initial measurements were taken on dedicated systems and lightly loaded systems. On dedicated systems no additional messages were generated. For lightly loaded systems some X-11[2] network traffic was generated. This traffic was part of the background load on the network. In order to minimize the load that was placed on the systems participating in the test, every process in the test system was run on a separate computer. There is one process, called the initiator, which controls the test. All other processes are either senders or receivers.

Figure 3.1 is an overview of the main part of each test cycle. The initiator reads a control file which specifies the number of senders, number of receivers, number of messages, and message lengths. The receivers are started first and then the senders. The senders are started using a multicast message, so they all start approximately at the same time. Senders keep track of the time between the sending of the first and the last message. Receivers keep track of the time between the first message they receive and the last message they receive. As each sender completes a cycle it notifies the initiator. When notification has been received from all participating senders, the initiator checks the status at all participating systems. If all systems have successfully

[2]X-11 is a "network transparent window system" which is used to provide remote windows[SG92a]. In the test environment used for this dissertation a single window on the system acting as the initiator for the test was used. X-11 traffic was not generated by any of the machines participating in the test.

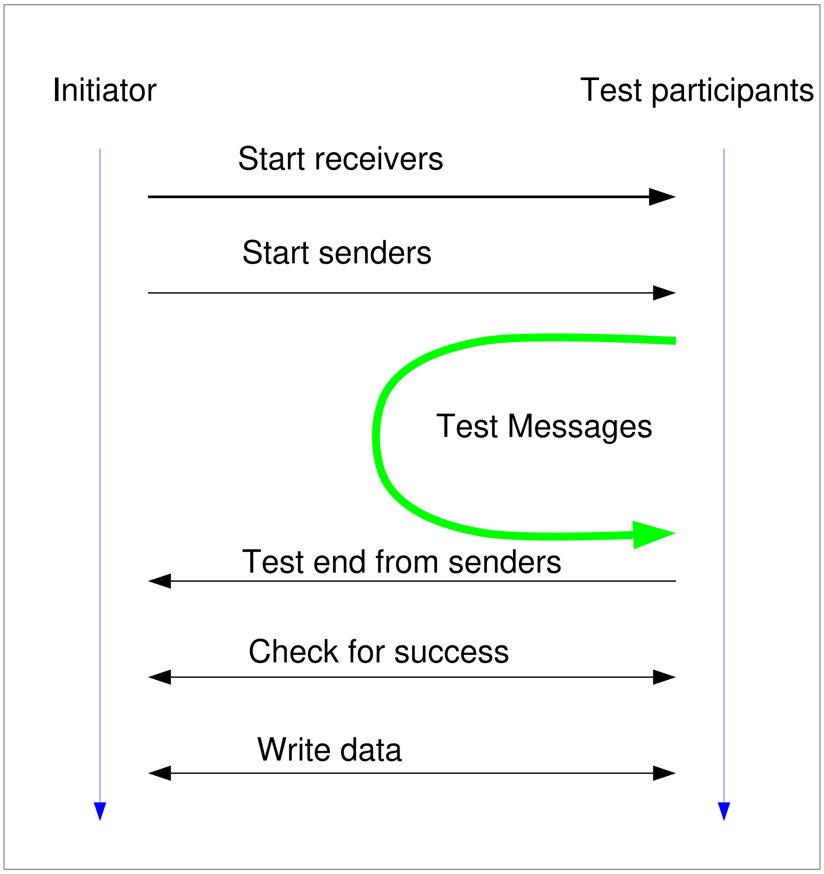

Figure 3.1: Overview of test protocol. Time goes downward in this overview.

completed the test, they write their data to files and wait for commands to start the next cycle. The details of the test protocol are specified in Appendix A.

Senders also kept track of the number of send errors and retried sending until messages were successfully sent. The senders did not receive any failure to send notification back from the device driver during our test. Receivers kept track of receive errors and lost messages. Receive errors were retried until data was successfully received. (There were not a lot of receive errors.)

3.2 Experimental Results

Our objective was to measure multicast performance from the application level. Performance was measured in terms of messages per second and K bytes per second for fixed length messages. For each message length in a test, a fixed number of messages was sent. For our initial testing this was typically 10,000 messages. The tests were generally run with two receivers. Different tests varied the number of senders to increase the load on the network. Each receiver kept track of the first message that was lost from each sender in the test. The message loss data that was collected showed that as errors began to occur, different receivers were losing different messages from the same sender. This, combined with the fact that no send errors were reported by the device driver to the senders, indicates that the early losses were receive omissions. As the loss rate increases it is not possible to distinguish between send omissions and receive omissions by this method. However, these findings are consistent with what has been reported for UNIX in the past and therefore indicate that the losses experienced during the test were receive omissions.

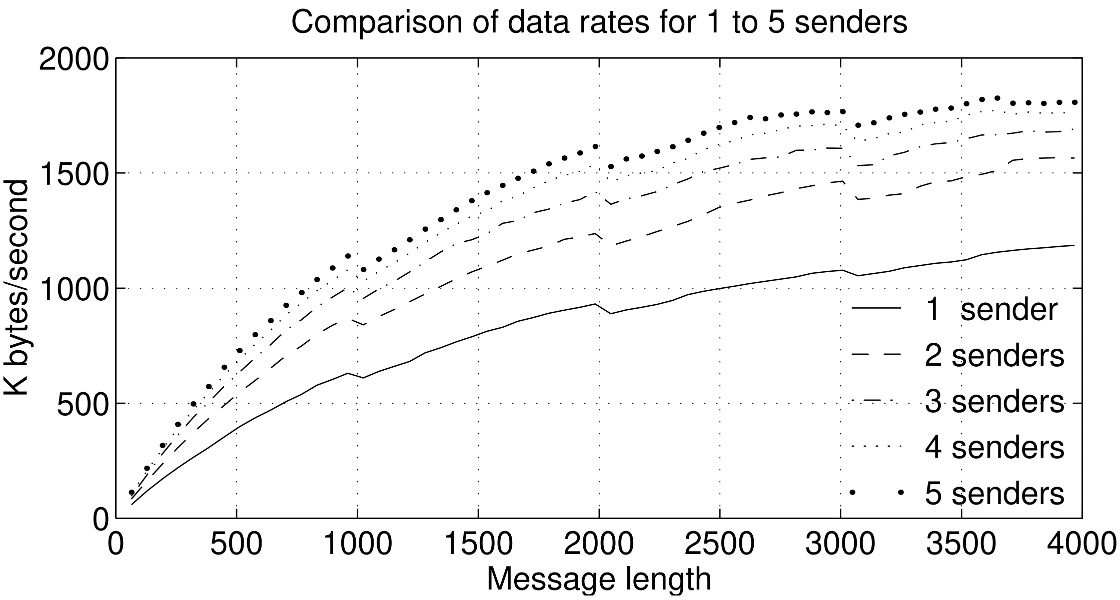

Figure 3.2: Token-ring network load (K bytes/second) for test with 1 to 5 senders and 1 receiver.

3.2.1 Initial Investigation

The initial testing was done on IBM RISC/6000 model 550 workstations using a 16 Mbit/second token-ring. The testing was done on isolated dedicated systems. The token-ring containing the test systems was physically disconnected from the rest of the network, so that the testing could be done without the possibility of affecting anyone else on the network. In addition, all of the user level processes running on all systems on the subnet that were not associated with our tests were killed. Therefore the data that was collected was associated only with the load placed on the systems and network by our multicast testing.

Increasing the number of senders during a test increased the load on the network and the load on receivers as expected. Figures 3.2 and 3.3 show the increased load in terms of K bytes per second and messages per second. As would be expected, the

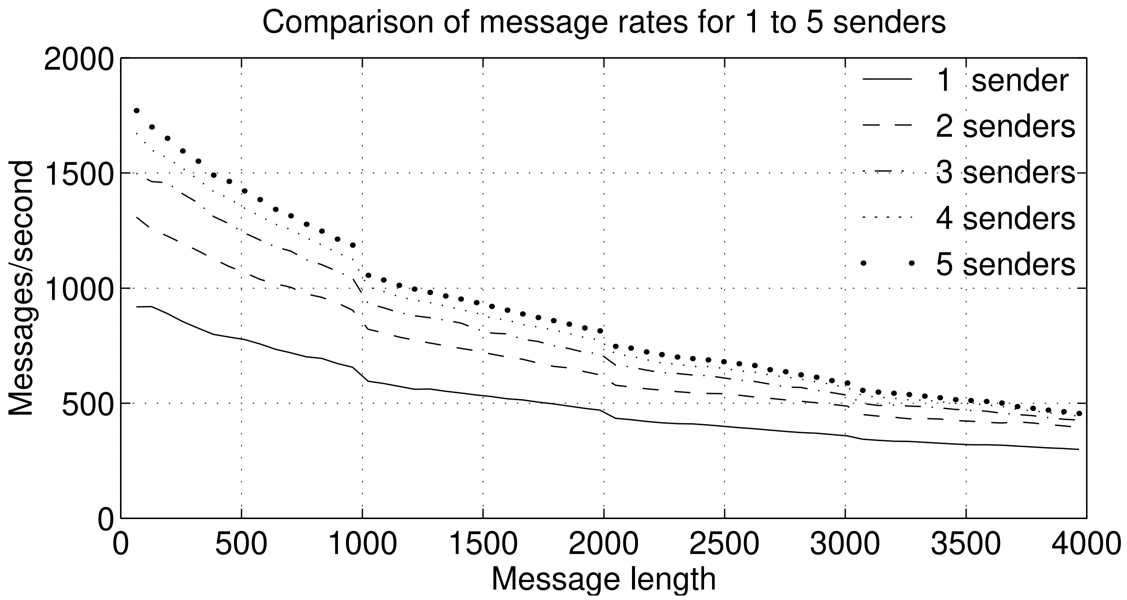

Figure 3.3: Token-ring network load (msg./sec) for test with 1 to 5 senders and 1 receiver.

total throughput (K bytes/second) increases as the message length increases. This is illustrated in figure 3.4.

Our tests verified that senders can overrun receivers, as expected. Figures 3.5 and 3.6 show the results in terms of messages per second and K bytes per second from a test in which there were 4 senders and 2 receivers. For these figures the solid line represents the total of all the senders and the dashed line represents the average of the 2 receivers. The space between the lines is the region where the senders' total throughput is exceeding the receivers and therefore flow control is needed. The receivers' rate rises to meet the senders' rate at approximately 1000 messages per second. In figure 3.5 (and many of the other figures in this section) there are a number of downward spikes after the receivers' rate initially matches the senders' rate. These downward spikes are caused by an anomaly that was found during the

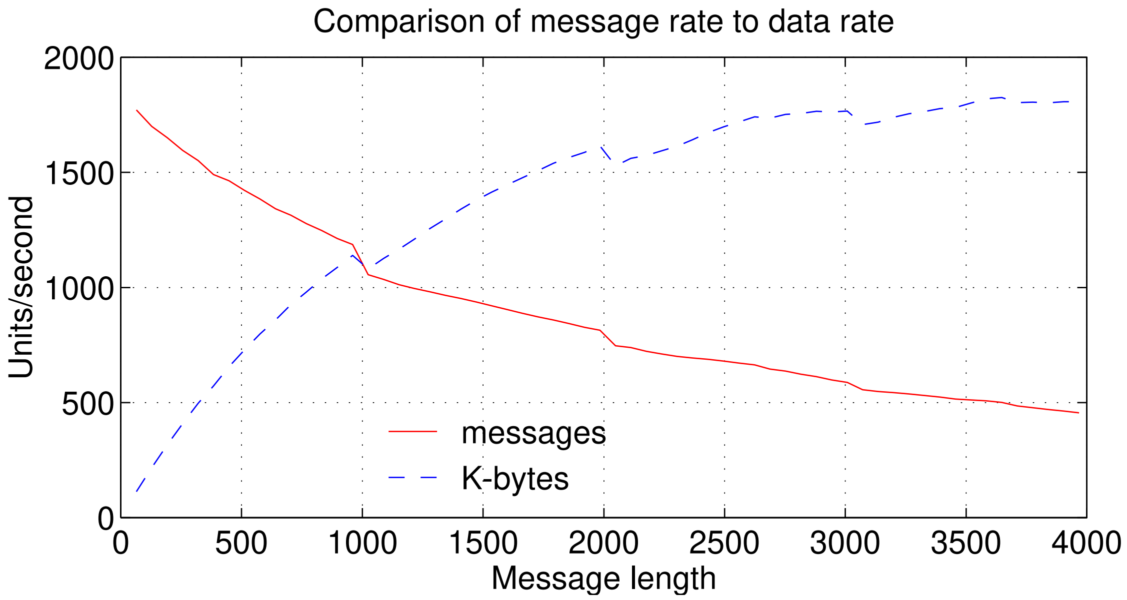

Figure 3.4: Token-ring comparison of the data rate in terms of messages/second and K bytes/second for a test with 5 senders and 1 receiver.

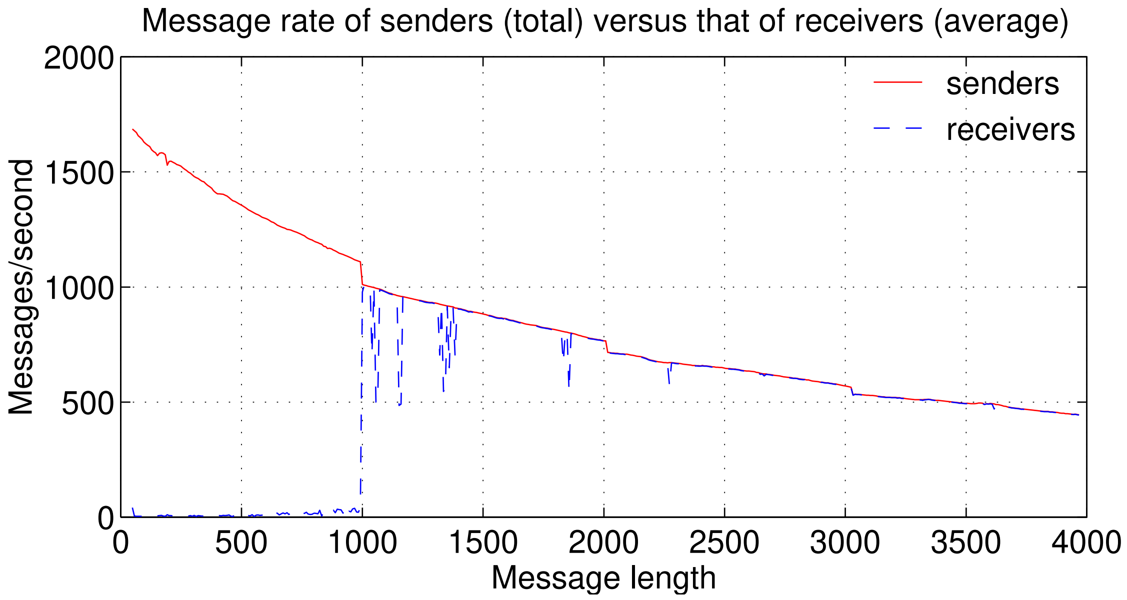

Figure 3.5: Token-ring message rate comparison for 4 senders and 2 receivers. This test was run on a 16 Mbit/second token ring.

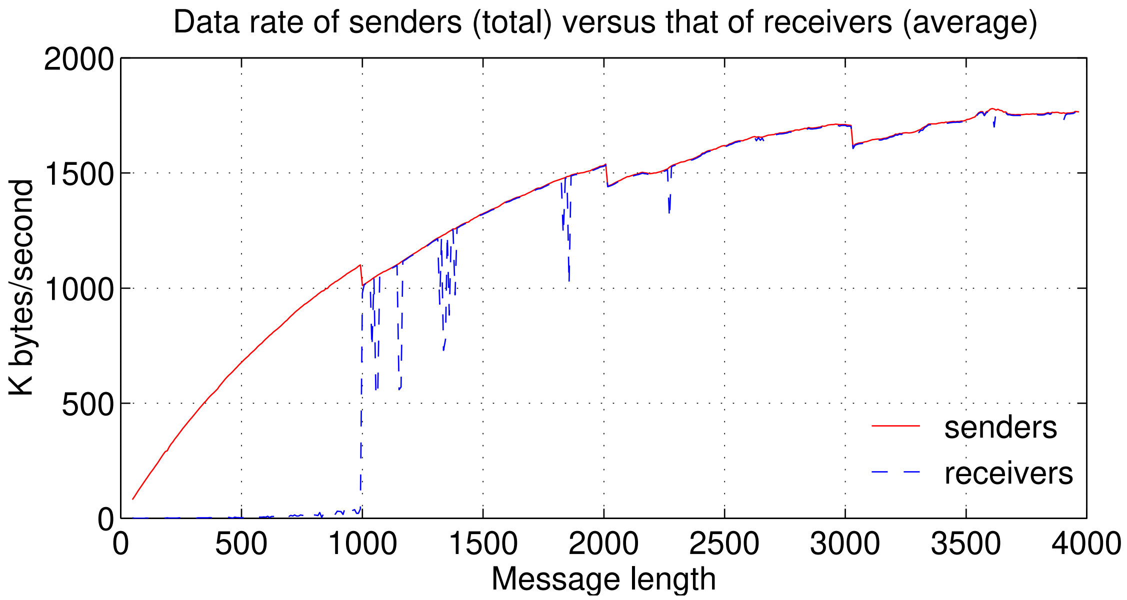

Figure 3.6: Token-ring data rate comparison for 4 senders and 2 receivers. This test was run on a 16 Mbit/second token ring.

token-ring testing. This anomaly is discussed further in section 3.2.2.

For figures 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, and table 3.1 there is one receiver and a varying number of senders. When the senders' transmission rate was above approximately 1000 messages per second, the receiver was unable to receive most of the messages sent. As the total senders' transmission rate decreases below 1000 messages/second, the receiver's reception rate increases to approximately meet the senders' transmission rate. This is shown in figures 3.7, 3.8, and 3.9 by graphing the senders' rate compared to the receiver's rate. In figure 3.7 the senders' and receiver's rates are nearly the same over the entire range, so that the graph looks like a single line. For this test the total of the senders' rate never exceeded 1000 messages per second. In the next two figures it is clear that the reception rate for the receiver rises to approximately equal the total senders' transmission rate when the total senders'

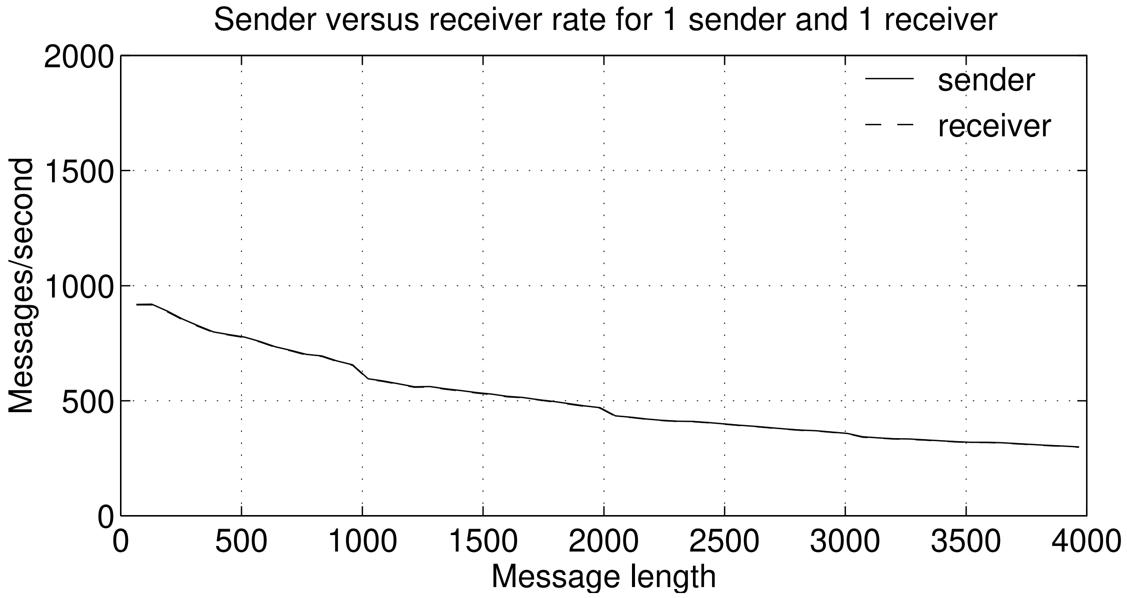

Figure 3.7: Token-ring message rate comparison for 1 sender and 1 receiver.

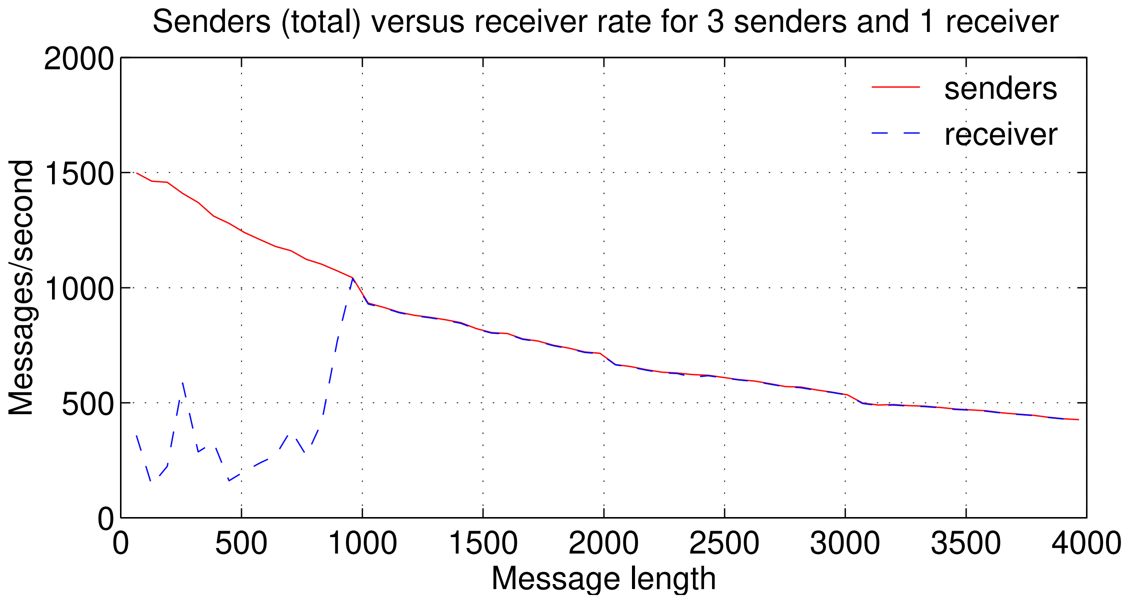

Figure 3.8: Token-ring message rate comparison for 3 senders and 1 receiver.

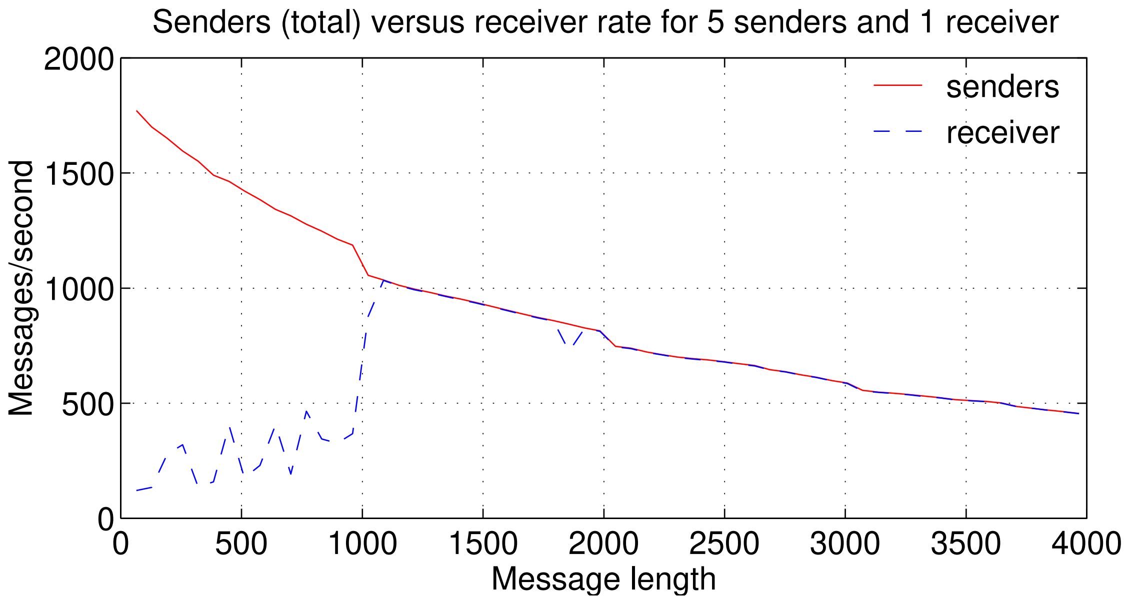

Figure 3.9: Token-ring message rate comparison for 5 senders and 1 receiver.

transmission rate is below approximately 1000 messages per second. For the test shown in figure 3.8 there were 3 senders and 1 receiver, and for the test shown in figure 3.9 there were 5 senders and 1 receiver. Note that the message rate and message length where the receiver's rate rises to meet the senders' rate is similar for both graphs. Figure 3.10 shows the senders' and receiver's rates for five tests (including the three just presented in figures 3.7, 3.8, and 3.9) superimposed on one another. For the tests with 2 to 4 senders, the receiver's rate rises to meet the senders' rate when the senders' rate drops below approximately 1000 messages per second. Figure 3.11 shows only the receiver's rates, and Table 3.1 shows the data from these tests around the point where the receiver's rate rises to approximately meet the senders' rate for these tests. Clearly, this shows that the message rates are approximately the same, from 1004 to 1068 messages/second, where the convergence takes place. These results

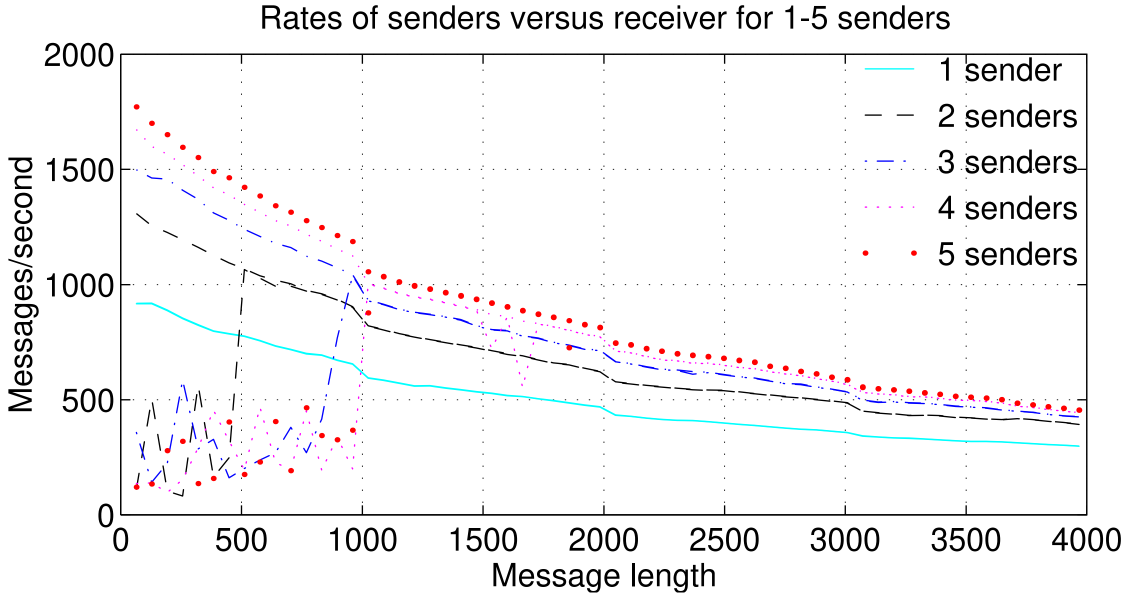

Figure 3.10: Token-ring comparison of senders (total) and receiver rates for 5 tests where the number of senders varied from 1 to 5 and each test had 1 receiver.

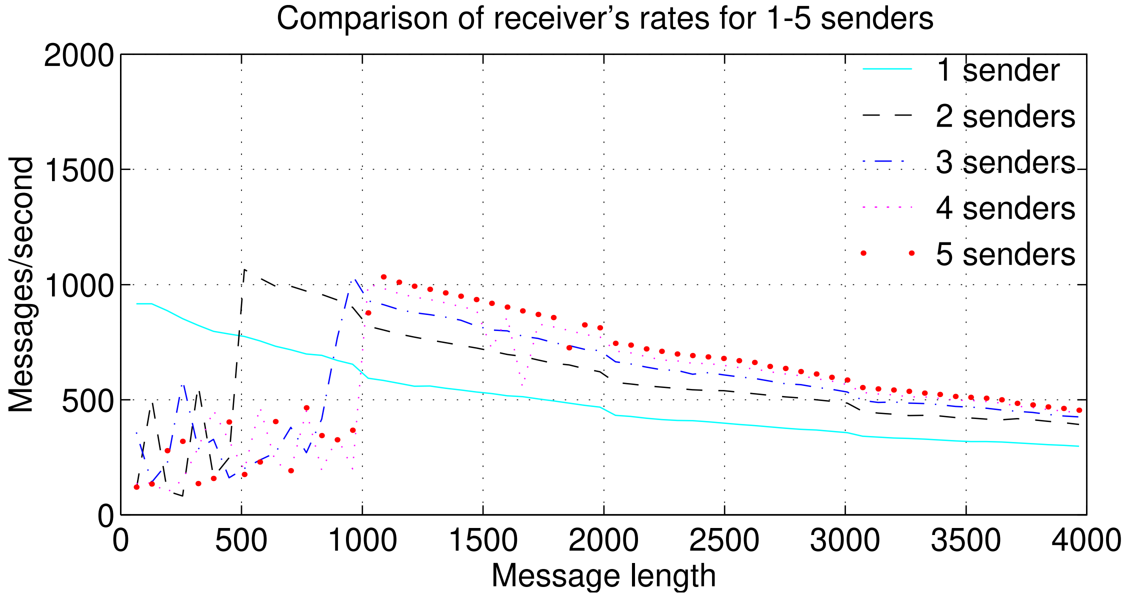

Figure 3.11: Token-ring receiver message rates for tests with 1 to 5 senders and 1 receiver.

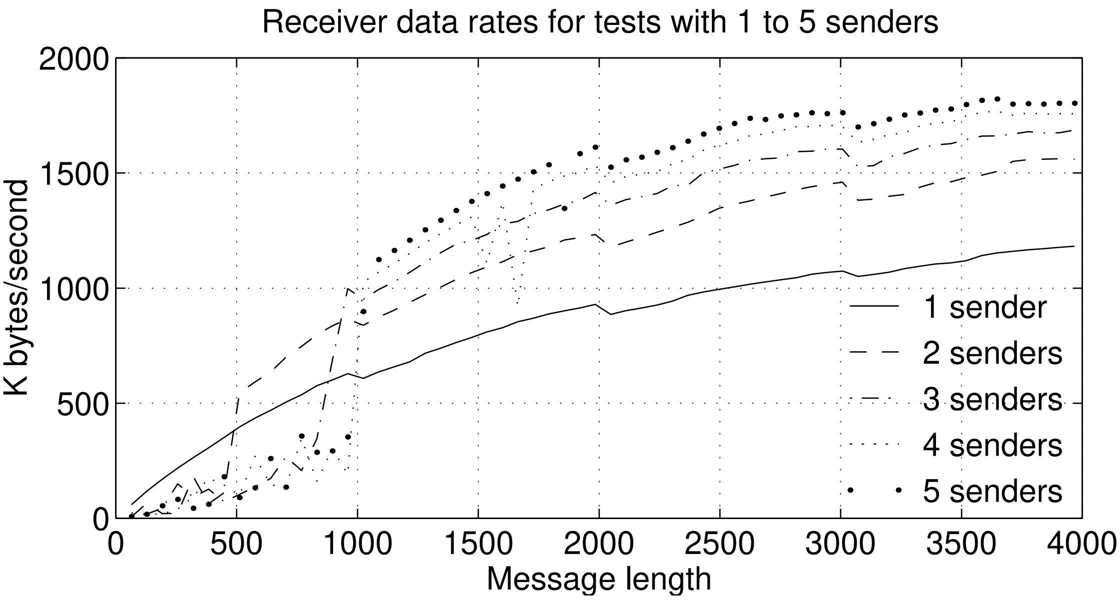

Figure 3.12: Token-ring receiver data rates for tests with 1 to 5 senders and 1 receiver.

are representative of all the testing that was done on token-ring, including some testing on a 4 Mbit/second token-ring. This characteristic of the curves is independent of network speed.

Since the message rate and message length were similar at the point of convergence, the data for the same tests were also analyzed in terms of K bytes/second, but the analysis did not show the same strong correlation. Figure 3.12 shows the receiver's data from the same test with 1 to 5 senders and 1 receiver. The receiver's rate does rise to meet the senders' rate, but the convergence points, in terms of K bytes/second, are more spread out (see table 3.1), from 546 K bytes/second to 1029 K bytes/second. The message lengths are also spread out, from 500 to 1000. In addition, the test with 1 sender and 1 receiver has message lengths and data rates below all of these ranges with no difficulty. Therefore this indicates that receiver overrun was not correlated

Table 3.1: Message rate convergence data for tests with 2 to 5 senders and 1 receiver.

Number of senders	Length	Senders K bytes/sec.	Senders Msg./sec.	Receiver K bytes/sec.	Receiver Msg./sec.	Receiver Loss %
2	384	432.8	1127	63.1	164.3	84.62
	448	490.2	1094	111.1	248	74.25
	512	546.9	1068	545.1	1064	.03
	576	597.9	1038	592.0	1027	.71
	640	652.2	1019	635.6	993.2	2.21
3	832	917	1102	345.7	415.5	62.0
	896	960.8	1072	688.0	767.8	28.19
	960	1001	1043	997.8	1039	.08
	1024	954.9	932.6	951.3	929	.11
	1088	995.5	915.0	993.2	912.9	.003
4	896	1033	1153	301.6	336.6	70.63
	960	1079	1124	191.3	199.2	80.81
	1024	1029	1004	1026	1002	0.00
	1088	1072	985.4	1069	983.2	0.00
	1152	1109	962.8	1106	960.5	.02
5	960	1139	1186	353.0	367.7	66.59
	1024	1080	1055	897.7	876.7	16.74
	1088	1126	1034	1123	1032	0.00
	1152	1166	1012	1163	1010	0.00
	1216	1210	995.3	1207	993.1	0.00

to message length or data rate (in terms of K bytes/second).

These two sets of analysis show that for these tests the ability of receivers to keep up with senders who are multicasting is determined by the message rate of the multicast, independent of the other factors that were measured.

3.2.2 Token-Ring Anomaly

Some of the token-ring tests produced some unusual results. These results were clearer when testing at finer resolutions, 8 bytes between test points instead of 64. The two unexpected discontinuities in the data were a *performance discontinuity* and a *receiver discontinuity*. Figure 3.13 illustrates both discontinuities.

The first problem affected the overall performance of both senders and receivers equally. Normally, continuous improvement (smooth curves) is expected in performance as the message length increases. However, what the tests show is a slight drop in performance every time the message length crosses a 1000 byte boundary. This is illustrated by circle **A** in figure 3.13. Because receivers can only receive data if it is sent by the senders, there is no way of determining (from the application level) if the anomaly affects only senders or both senders and receivers. Because senders are affected, the performance of senders and receivers is affected equally, which is why this is called a performance discontinuity. (It is observable on the 64 byte interval token-ring tests already presented.) The test application is insensitive to these boundaries, so this must be an attribute ("feature") of the underlying system.

The second anomaly that was observed only seems to affect receivers. It is illustrated in circle **B** of figure 3.13. In a region where, based on the results of the

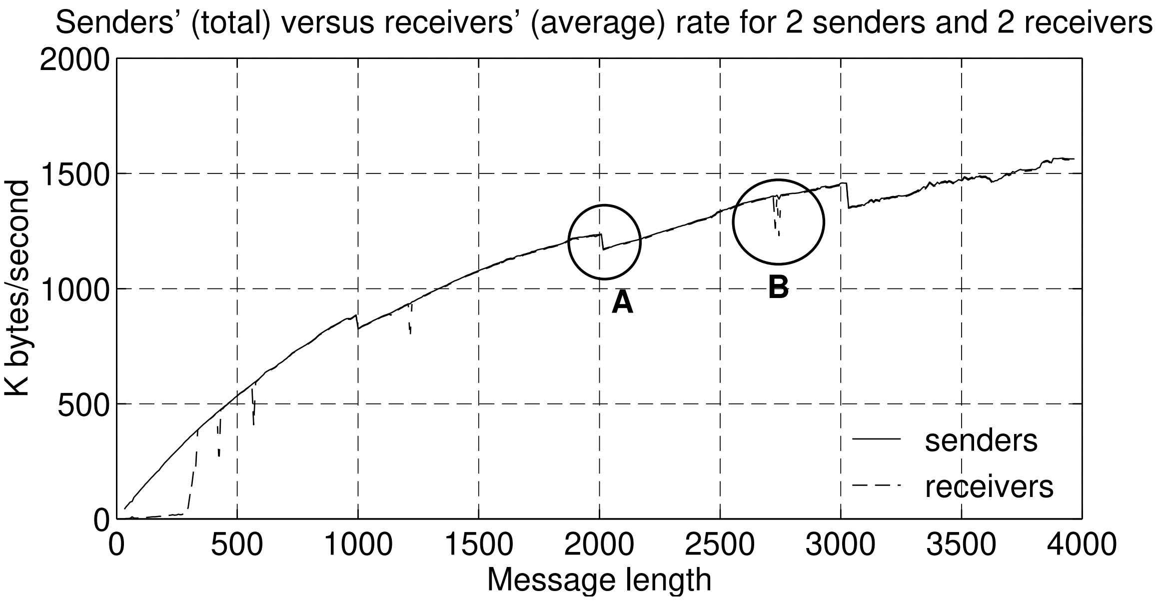

Figure 3.13: Token-ring 8 byte interval test illustrating anomaly. Circle **A** illustrates a performance discontinuity and circle **B** illustrates a receiver discontinuity.

previous section, the receivers were more than capable of keeping up with the senders, the receivers' rate drops unexpectedly. This drop is represented by spikes down from the top curve in this figure. Once the senders' transmission rate drops below the receivers' reception limit, the graph should show a single line representing the senders' and receivers' performance.

Some of our testing was done with 8 byte intervals (between the test points) and some with 64 byte intervals. Many of the tests — for example, those presented in figures 3.5 and 3.6 — exhibit this unexpected receiver discontinuity. It appears as spikes down from the senders' performance after the receivers' and senders' curves initially meet. A more detailed examination was done to explore this anomaly. Figure 3.14 shows a detailed test on a 16 Mbit/second token-ring for a range where the receivers should be able to keep up with the senders. It is more instructive to look

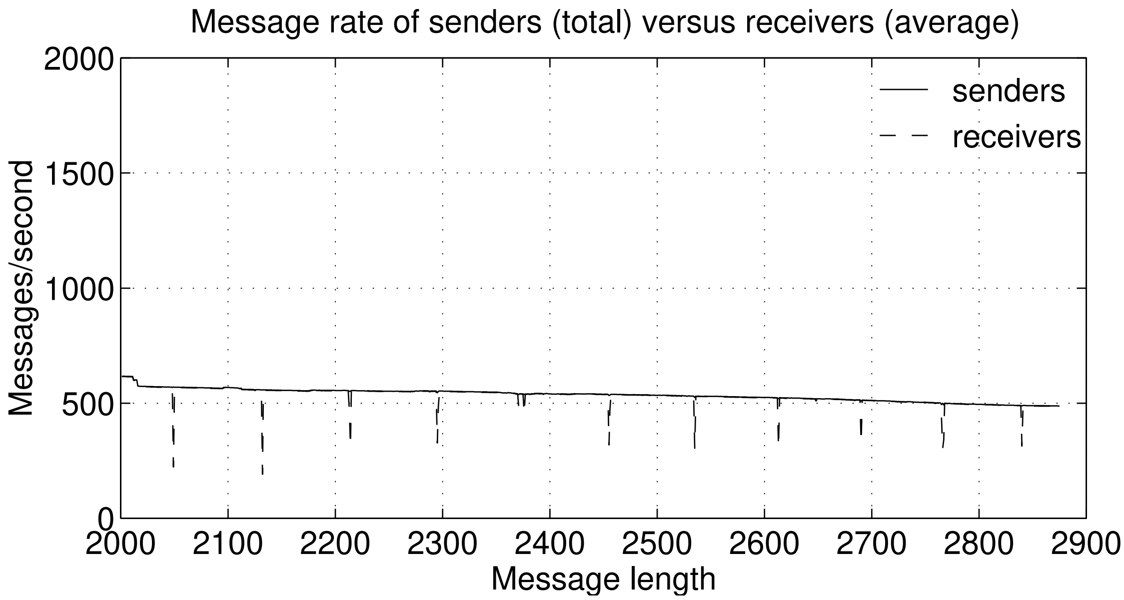

Figure 3.14: Performance loss anomaly in terms of messages per second.

at the results from this test in terms of the average percentage message loss, which is shown in figure 3.15. This test was done at 1 byte intervals with messages of 2001–2875 bytes in length. The average percentage loss for this test of all the points where the spikes occurred is 36.30% (including the double peaks at 2371 and 2376). The minimum and maximum loss percentages were 10.74% and 64.47% respectively. For this test the average length of interval between points where the discontinuity occurs is 79.1 bytes. (Only one of the double peaks was included in the average, because each receiver in this test had an error at one of the two peaks.) When a similar test was run on a 4 Mbit/second token-ring, the points at which loss occurred were $77 + (104 * n)$ for $n = 1, 2, 3, \cdots$ bytes apart. (The token-ring adapters manage their buffers in doubly linked segments of 112 bytes[Int90].) This indicates that something within the adapter hardware and microcode was causing it to lose messages independent of

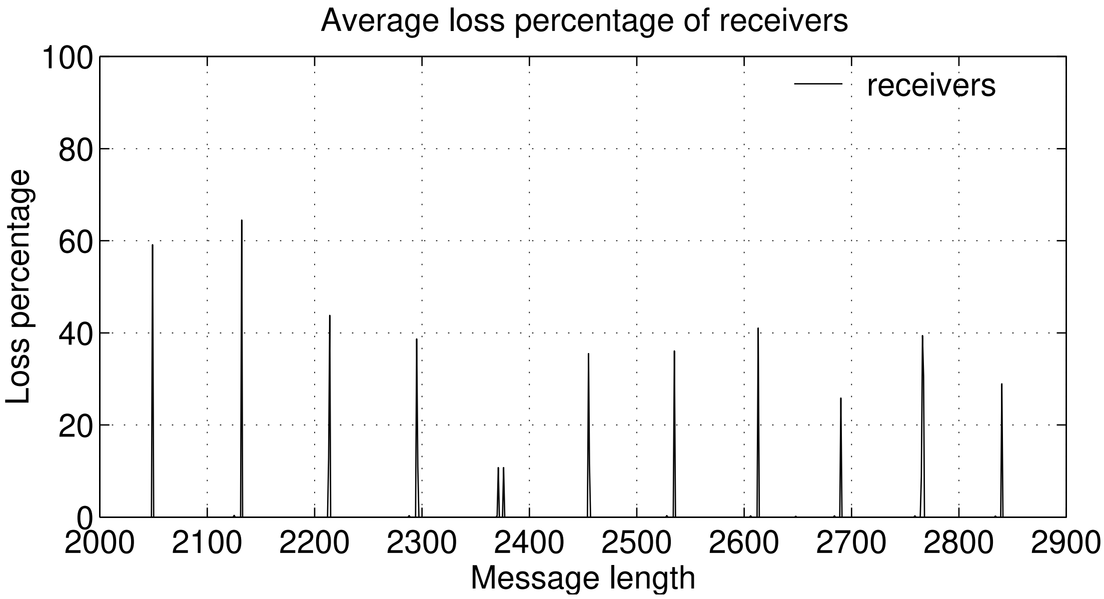

Figure 3.15: This picture shows the message loss percentage from a test on a 16 Mbit/second token-ring. The graph was generated by transmitting 10,000 fixed length messages at one byte intervals from 2,001 to 2,875 bytes. There were 2 senders and 2 receivers in the test. The message rate was not sufficient to cause the indicated losses. In our opinion the losses were caused by an anomaly in the network adapter (hardware and microcode) of the RISC/6000 model 550 (AIX version 3.1.5) used for the test.

the message rate. There is no explanation as to why the points shift as the adapter speeds up, except to presume that the problem is sensitive to the communications rate.

This was not the main focus of our work, but it is instructive because it is common to assume that there are no failures in the underlying hardware system. The usual assumption is that losses occur either in the operating system or the network. This investigation indicates that significant losses can be introduced by the communication adapter. Further, it indicates that if an application is consistently experiencing unexpected losses for messages of a particular size, changing the size of the message may be an effective method of avoiding the failure.

3.2.3 Emulating Multicast with Point-to-Point

Point-to-point communication is commonly used to provide multicast function. The same test system was used to evaluate the performance of point-to-point communications simulating multicast. This was done to compare multicast simulated by point-to-point with actual multicast. Multicast performs better as the group size increases, as would be expected. However, for the token-ring, point-to-point communication between 1 sender and 1 receiver is faster than multicast. Figure 3.16 illustrates our finding. Recall that, as shown in figure 3.7, the rate for multicast with 1 sender and 1 receiver was never above 1000 messages per second. Even though the sender can send faster with point-to-point, for the test shown in figure 3.16 the receiver is still limited to receiving approximately 1000 messages per second. This test was done at 8 byte intervals and still shows some erratic behavior but does not

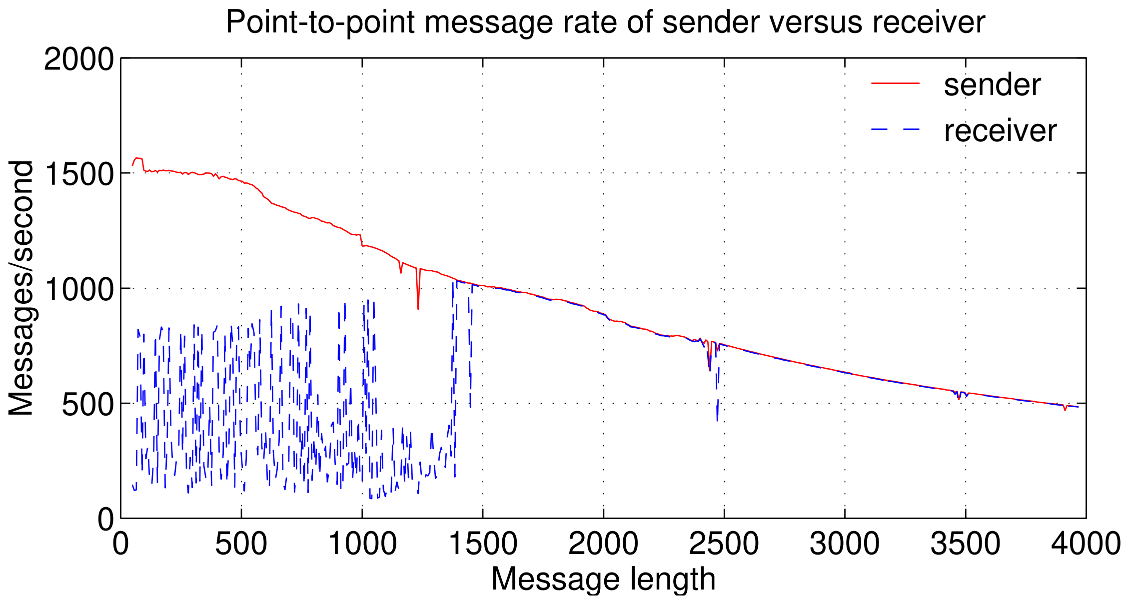

Figure 3.16: Token-ring point-to-point 1 sender and 1 receiver data rate comparison.

appear to have the performance discontinuity that was present with broadcast. (This was not examined in detail.)

Even though performance is much better with 1 sender and 1 receiver, as the number of receivers increases, figure 3.17 shows that performance drops off dramatically. This is as expected, because for broadcast/multicast communication there is no “cost” to the sender for adding an additional receiver, whereas for point-to-point communication the sender has to send an additional message for each additional receiver. Figure 3.17 shows the senders’ and receivers’ rates on the same graph. However, for 2 to 5 receivers the sender’s rate is below 1000 messages per second, so the sender’s and receivers’ rates are essentially identical.

Total sender performance in terms of messages per second can be improved dramatically by increasing the number of senders and using a single receiver. This

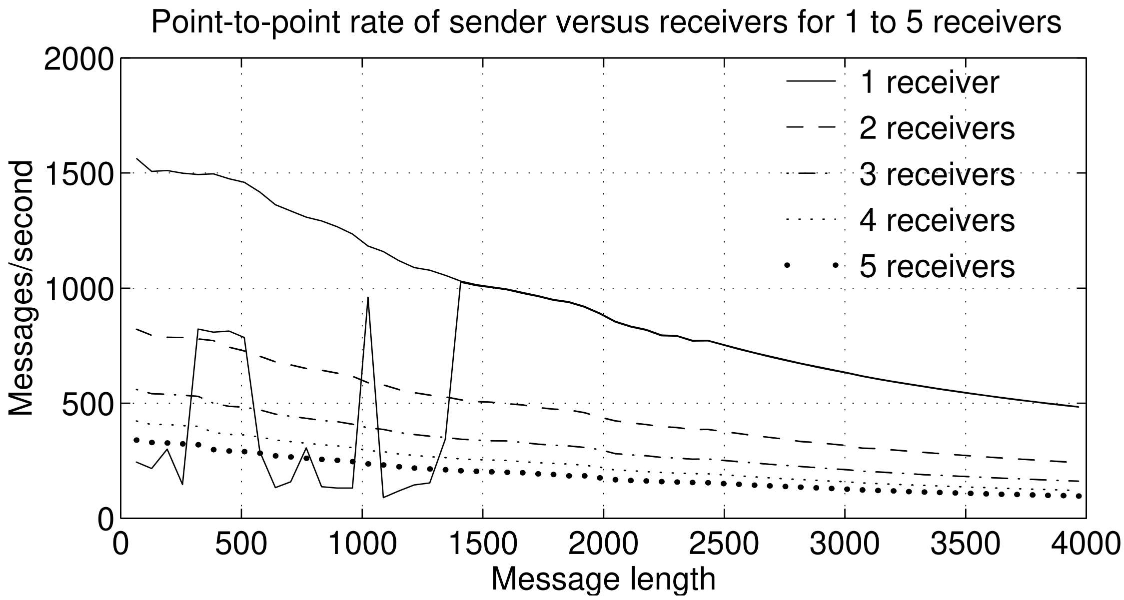

Figure 3.17: Token-ring point-to-point 1 sender and 1 to 5 receivers data rate comparison.

technique also avoids the severe degradation that point-to-point emulation of multicast experiences as the number of receivers increases. Nevertheless, as shown in figure 3.18, the receiver in these tests is still limited to approximately 1000 messages per second. Therefore improving the total senders' performance does not produce a corresponding improvement in receivers' performance.

All of the curves except for the 1 sender and 1 receiver curves merge at a message length of about 2,000 bytes as the message rate drops below 1000/second. The 1 sender and 1 receiver curves' convergence point is at message length 1500 bytes, as shown in figure 3.16, which uses the same data. Figure 3.19 reinforces this point, showing the data for 4 senders and 1 and 2 receivers on the same graph. Because there are multiple receivers, the senders' performance degrades as expected and the two curves meet "sooner".

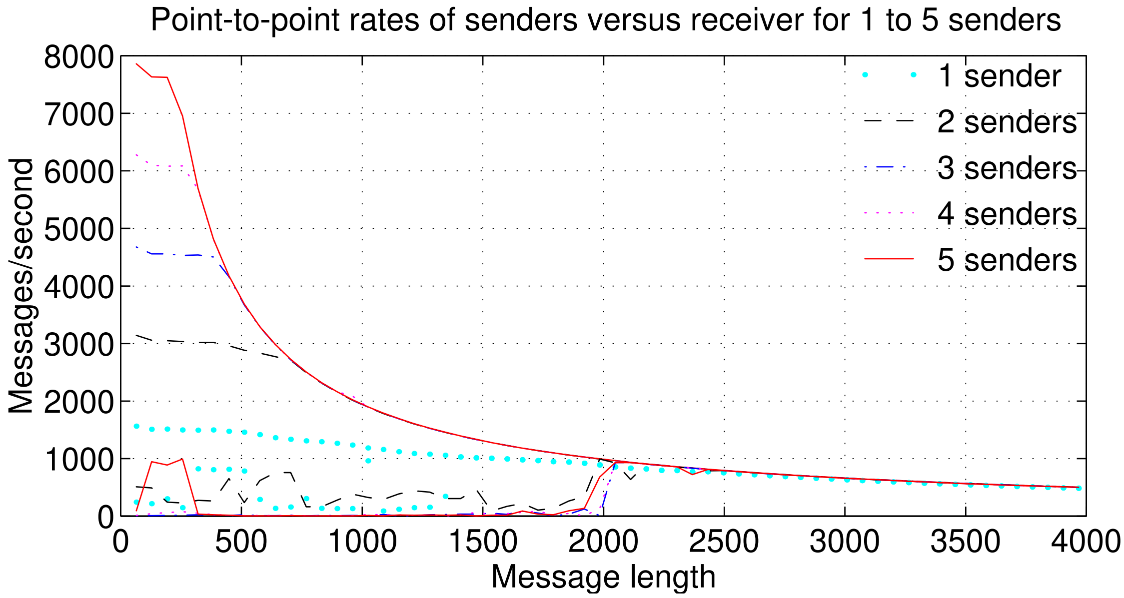

Figure 3.18: Token-ring point-to-point 2 to 5 senders and 1 receiver data rate comparison.

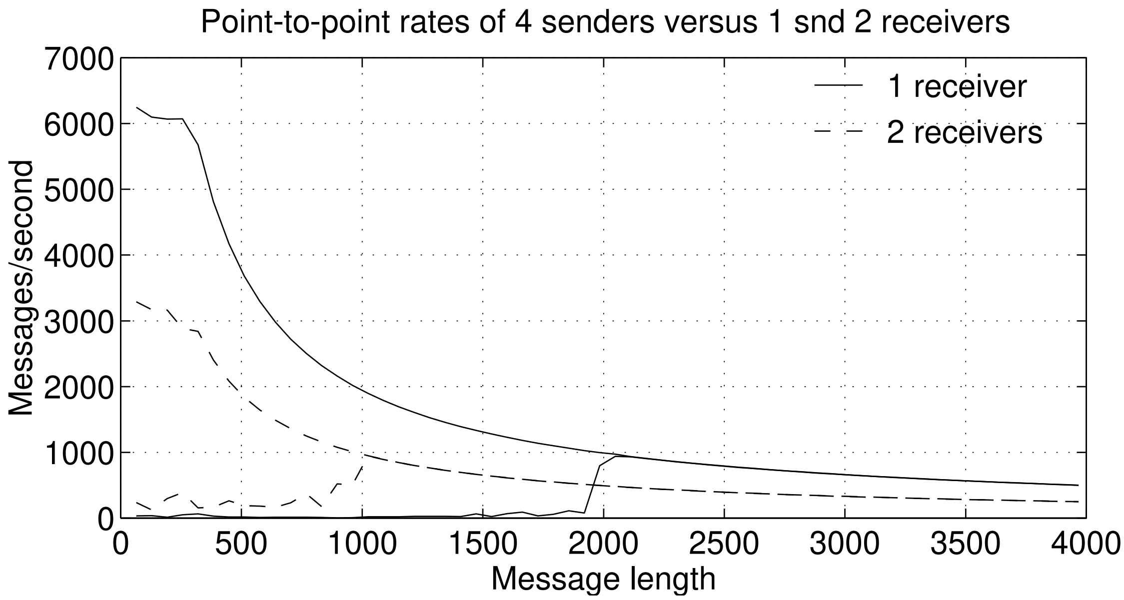

Figure 3.19: Token-ring point-to-point 4 senders and 1 and 2 receivers data rate comparison.

It is interesting to note that even though these tests were using the same message interval, there was no strong evidence that the receiver discontinuity exists when using point-to-point communications to emulate multicast.

3.2.4 Ethernet Testing

The next objective was to determine whether the initial results represented general characteristics of multicast or were in some way related to the system and network that was used for the tests. A change was necessary to make this evaluation. The test system was shifted from IBM RISC/6000s model 550s to HP 720s, and from isolated systems and LANs to shared systems and lightly loaded LANs. Finally, it was shifted from a token-ring network to ethernet. The systems used in the new tests were matched, the same model running the same level of the operating system. One of the benefits of this change was that there were more systems available to use for testing. However, because there were other activities occurring on both the systems and the networks used for these test, this shift limited the test results to identifying trends. Even though the testing was done at a similar level, if there were anomalies in the data, they could just as easily be associated with other users as with the hardware. Nevertheless, the benefits of having more systems to test with, of having a testing environment that was more realistic, and of moving to a different network type and hardware vendor, were more important than the disadvantages.

The software used for testing ethernet was the same as the software used for the token-ring tests. It was only necessary to change the interface to the device driver. HP's LLA interface was used instead of IBM's GDLC interface. Both interfaces

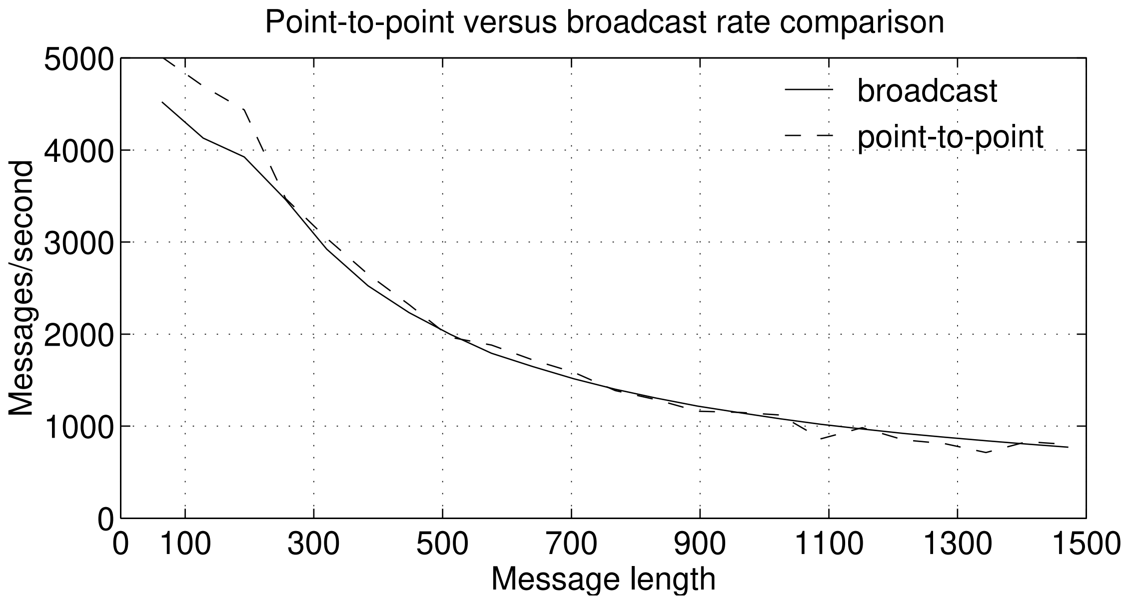

Figure 3.20: Ethernet comparison of point-to-point versus broadcast performance.

implement SAPs, and multicast was simulated the same way, broadcast to different SAPs. Because of the change from token-ring, with a maximum packet size of 4096 bytes, to ethernet, which has a maximum packet size of about 1500 bytes, it was also necessary to change some miscellaneous constants associated with the interface. However, it was not necessary to build MAC headers for the LLA interface. Ethernet also supports multicast, but multicast hardware was not available to application level programs. The test structure was not changed. Most of the tests were at 64 byte intervals (with a few at 8 byte intervals), and most tests transmitted 10,000 messages for each length tested.

For ethernet, point-to-point transmission is slightly faster than broadcast, as shown in figure 3.20. In terms of messages per second, the HPs and ethernet were faster than the IBM systems. (This is probably due to a more mature implementation

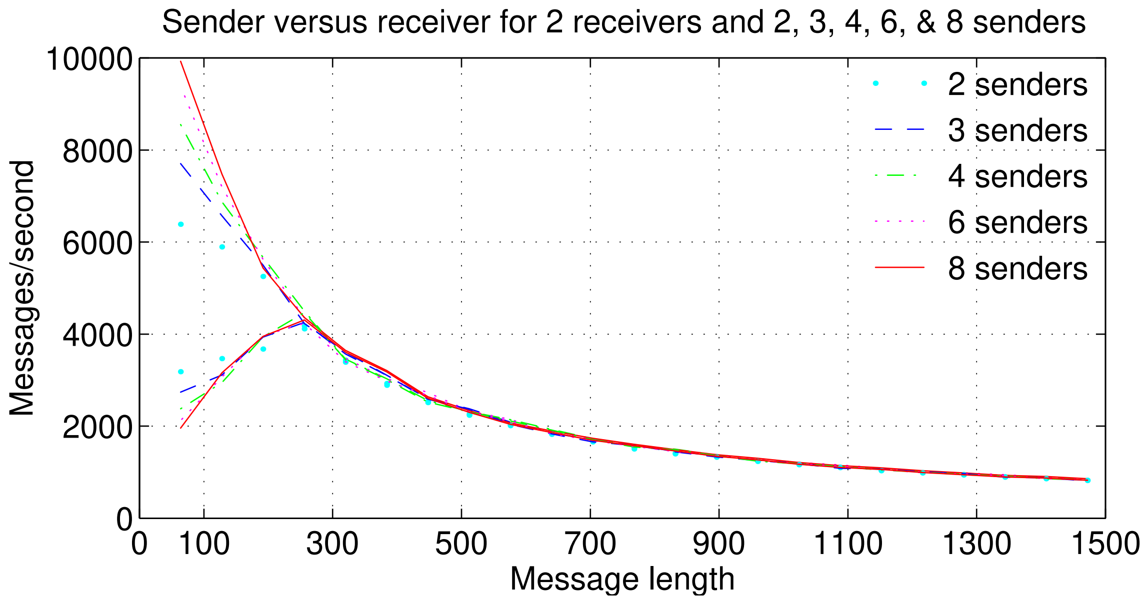

Figure 3.21: Ethernet multicast rate comparison for 2, 3, 4, 6 and 8 senders and 2 receivers.

of UNIX and the underlying communications interfaces.) The trend for broadcast is the same, rate limited, as shown in figure 3.21. For figure 3.21 there were 2 receivers used during the test. Even though many more messages can be transmitted, the receiver is limited to the range of 4100–4500 messages per second. The wider spread of the limit is probably due to the load on the network and the load on the system participating in the test. Point-to-point emulation of multicast is slower than actual multicast because a separate message must be sent to each receiver. If this disadvantage is avoided and a single receiver is used while varying the number of senders, the rate limitation can clearly be seen, as shown in figure 3.22.

With more systems it is easier to see the impact of increasing the number of receivers for point-to-point emulation of multicast. Figure 3.23 shows tests with 3 senders and 1 to 4 receivers. In this figure the senders' and receivers' rates are

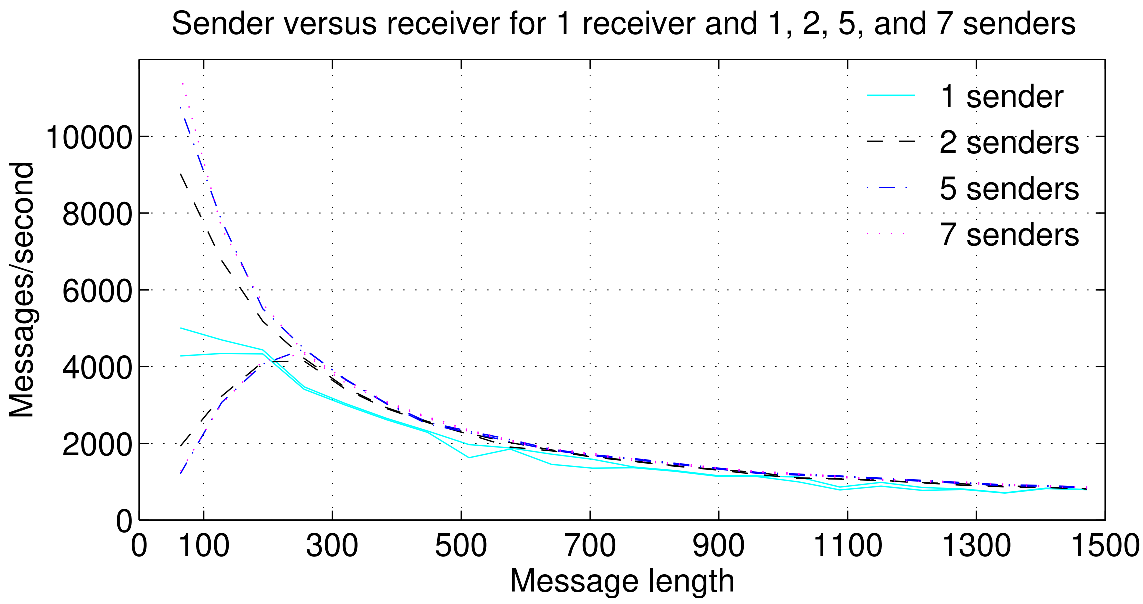

Figure 3.22: Ethernet point-to-point rate comparison for 1, 2, 5, and 7 senders and 1 receiver.

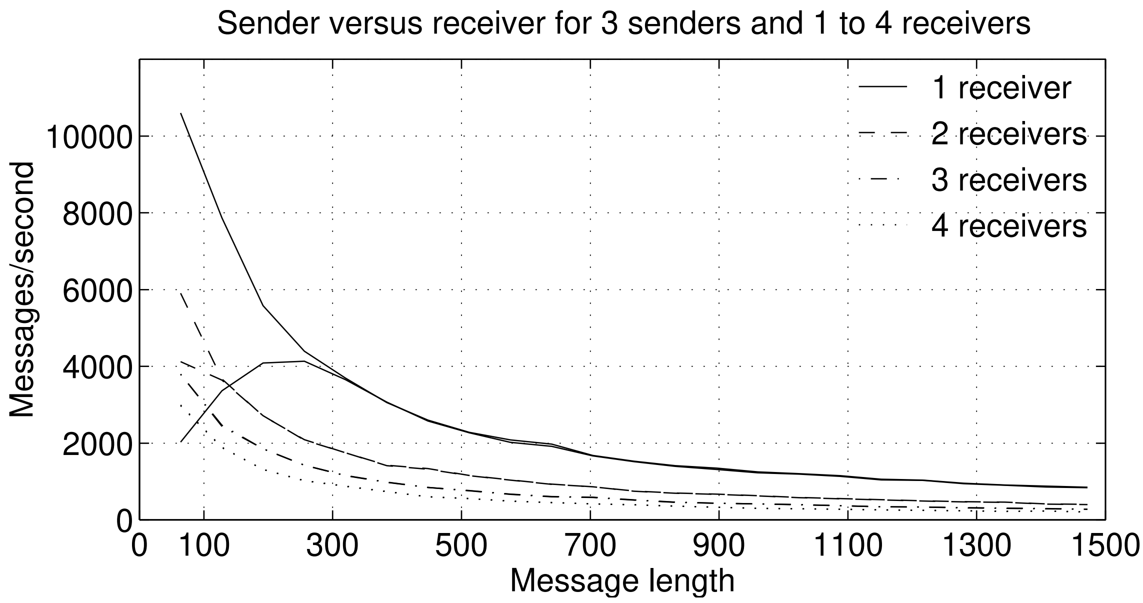

Figure 3.23: Ethernet point-to-point rate comparison for 3 senders and 1 to 4 receivers.

graphed using the same line type as before. However, for 3 and 4 receivers the senders' rate lies below the rate limitation of the receivers, so only a single line appears. Therefore point-to-point emulation of multicast on ethernet has the same characteristic as multicast, rate limited.

3.3 Summary

In this chapter primitive multicast performance was studied. The goal was to characterize multicast performance well enough to develop a design approach for a multicast flow control protocol. Surprisingly, the test data show a performance anomaly with the token-ring network. Nevertheless, a transport-layer-independent characterization of multicast communications was found; this characterization can be used to design a flow control protocol.

Multicast performance was tested for both token-ring and ethernet networks. Point-to-point emulation of multicast as well as multicast was tested in both environments. The principal trend that was found in the tested environments was that receivers are rate limited. The test data suggest that the rate limit varies with the load on the system, as one would expect. No matter how fast the senders send packets, receivers are limited by some rate, and if packets arrive faster than the receivers' rate limit, a high percentage of loss occurs. (No attempt was made to determine where within the systems the messages were being lost.) Further, on both networks point-to-point communication between two systems was faster than multicast between two systems. However, as expected, point-to-point emulation of multicast degrades severely as more receivers are added. Therefore, throughput cannot be dramatically improved by adding more senders with point-to-point emulation of multicast because of the receiver's rate limitation.

All of this suggests that rate control could be an effective component of flow control for multicast. A send-rate based method will adjust the send rate in response to errors because it assumes that if messages are being lost it is primarily due to receiver overrun. A send-rate based approach could also be used with other flow control techniques. If such a protocol were developed, the results presented indicate that it would be useful in current systems that emulate multicast with point-to-point communications as well as future systems which will exploit the emerging multicast technologies.

Chapter 4

Multicast Flow Control

This chapter studies the development of a flow control algorithm which incorporates the observations of chapter 3. It was observed that systems participating in multicast communication are rate limited. Therefore, if multicasts occur at a rate above the limit of a particular system, that system will lose a high percentage of messages. The goal is to develop a general purpose flow control protocol which exploits this observation and makes no assumptions about communication patterns or network topology.

4.1 Flow Control

The purpose of flow control is to allow communicating partners to send as many messages as possible without overrunning each other. Flow control can be either reliable or unreliable. Reliable flow control corrects (retransmits) errors that are detected; unreliable flow control does not correct errors. Even though unreliable flow control leaves error correction to higher levels of the system (which may not correct these errors), unreliable flow control is important for congestion avoidance and control.

Traditionally, errors are assumed to be caused by network congestion, which occurs when more data are fed into the network than the network has the capacity to handle, resulting in packet loss. In managing congestion the underlying assumption is that errors affect all packets equally, and therefore if everyone experiencing errors reduces the number of packets they transmit, congestion will subside. Further, it is assumed that users consuming the most bandwidth will experience the most errors and as a result will reduce their transmission rate fastest. Studies of congestion control have resulted in a number of improved techniques for flow control[Jac88].

Initially, flow control was developed for point-to-point communication networks. A path between two communicating applications might be made up of several point-to-point links. Figure 4.1 illustrates a simple point-to-point network with three concurrent conversations.

In a point-to-point network each link is between two computers. Each computer can act as either a sender or a receiver of messages. Flow control is an end-to-end protocol used to insure that a sender does not overrun the receiver. Reliable flow control guarantees that messages are delivered to the receiver. In this context, the basic principle of congestion management is relatively straightforward: when congestion occurs demand must be reduced. For example, it is easy to see in figure 4.1 that when link $p \leftrightarrow q$ becomes overloaded, if all the participants in the conversations which are using the link reduce their demand, the congestion will be corrected. Even if p

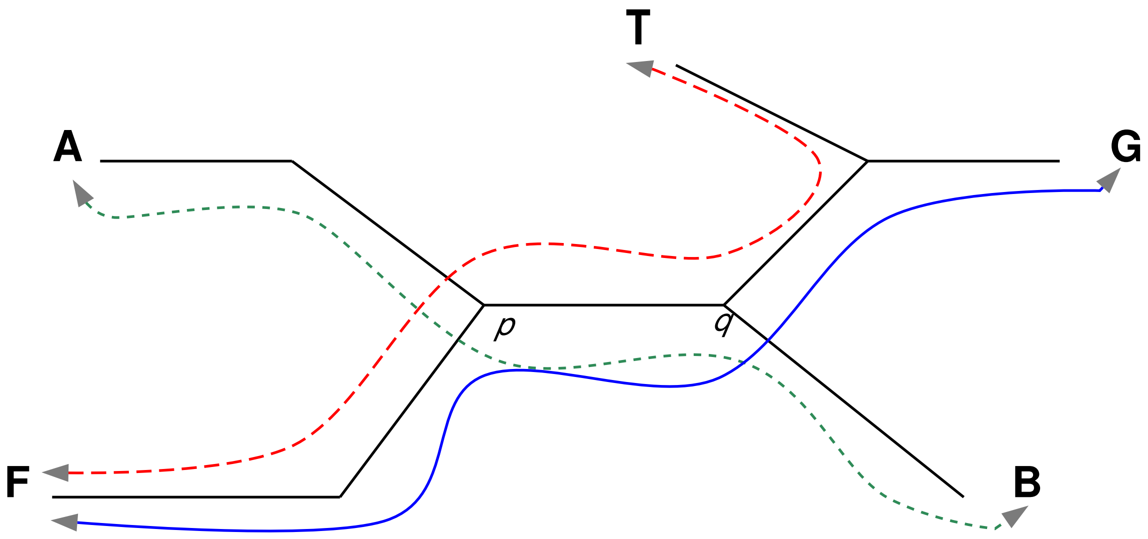

Figure 4.1: End-to-end flow control on a point-to-point network. The straight line segments represent point-to-point links. There are three concurrent conversations, **F**–**T**, **A**–**B**, and **F**–**G**. All three conversations share link $p \leftrightarrow q$.

and q have a reliable link between them, if all of their other communication partners (which are unlabeled in figure 4.1) continue to present more data for transmission across the reliable link than p and q can receive, packets will continue to be dropped. When packets have been transmitted for a period of time without error, then participants can increase the number of messages they send in an attempt to take advantage of excess capacity. Techniques to manage increasing and decreasing the number of messages sent to avoid erratic network performance for point-to-point networks have been published by Van Jacobsen [Jac88] and integrated into the standard internet protocol suite.

4.1.1 Functions of Flow Control

Flow control can be divided into three functions: *supply control*, *error detection*, and *error recovery*. Error recovery is optional, but a protocol must have supply control and error detection. In developing a flow control protocol it is necessary to have a clear understanding of the functions that must be provided.

Supply control determines when messages may be sent. The goal of supply control is to send messages as fast as possible without causing errors. The ability to send messages is constrained or released based on information received from error detection.

Error detection determines when an error has occurred. Error detection's determination is final, whether the error may actually have occurred or error detection may be assuming the error has occurred. The structure of a flow control protocol determines whether an error will first be detected by the sender or by the receiver. Obviously, whether errors are first detected by the sender or the receiver, for reliable flow control the sender (or some process which successfully received the missing packet) must correct the error through retransmission.

When errors have been detected, error recovery retransmits the message or requests retransmission, depending on whether the sender or receiver detected the error. When error recovery has been notified of an error by error detection, error recovery will attempt to correct the error until it is told that communication has been lost or it knows the error has been corrected.

A protocol that corrects errors generally does not send new data until it knows that data sent in error have been corrected. Depending on how supply control and error detection are done, a protocol which includes error recovery may send out new data before it realizes that an error has occurred with previously sent data. In this case, exactly how the protocol proceeds after correcting the error depends on the protocol design.

Normally the flow control protocol is not responsible for determining that communication has been lost. This determination is done by some other part of the system.

4.2 Multicast Flow Control

Multicast communications can be used independent of flow control[DFW90,Pal88]. Whether or not flow control is wanted, multicast communication differs from point-to-point communication because each message is directed to multiple recipients. (For the degenerate case, where there is one recipient, multicast is equivalent to point-to-point communications.) The collection of intended recipients is called the multicast set. Because each multicast is destined for multiple recipients, multicast flow control must explicitly or implicitly incorporate information from/about all of the intended recipients of each multicast message.

For multicast communications, a process multicasts (sends) messages to a multicast set. In general, each process which does a multicast expects that the multicast will be received by every member of the multicast set. A multicast is successful if it is received by every member of the multicast set; it has failed (or experienced an error) if one or more members of the multicast set do not receive the message. The general goal of multicast flow control is to allow processes which use multicast to send as many messages as possible without incurring errors.

It was observed in chapter 3 that recipients of multicast are rate limited. Therefore, if any process multicasts messages faster than the lowest rate for a member of the multicast set, errors will occur. Errors will also occur if the the total rate of multicast to a particular multicast set is faster than the lowest rate of a member of the set.

There are two general approaches to rate-based flow control: send-rate control and rate reservation. *Send-rate control* means controlling the minimum transmission intermessage spacing. Adjusting the intermessage spacing directly controls the maximum rate at which participating systems can send (or multicast) messages. *Rate reservation* means guaranteeing end-to-end bandwidth between communicating parties. Rate reservation also controls the maximum rate available to communicating parties. However, the difference between send-rate control and rate reservation is best illustrated in a multi-hop network like the example in figure 4.2. There are five links in the conversation between **F** and **T** in figure 4.2. For this conversation, send-rate control does not require that all intermediate systems deliver messages to the next link in the conversation at the specified rate. Send-rate control only requires that $p0$ and $p5$ deliver messages to their partners, $p2$ and $p4$ respectively, at the specified rate, whereas rate reservation fails unless all intermediate systems on the path deliver messages to the next system on the path at the specified rate [Jai92]. Therefore send-rate control specifies the maximum rate at which messages can enter the network, whereas rate reservation specifies the maximum rate at which messages traverse the network.[1]

[1]The bandwidth is reserved even if it is not consumed by the application.

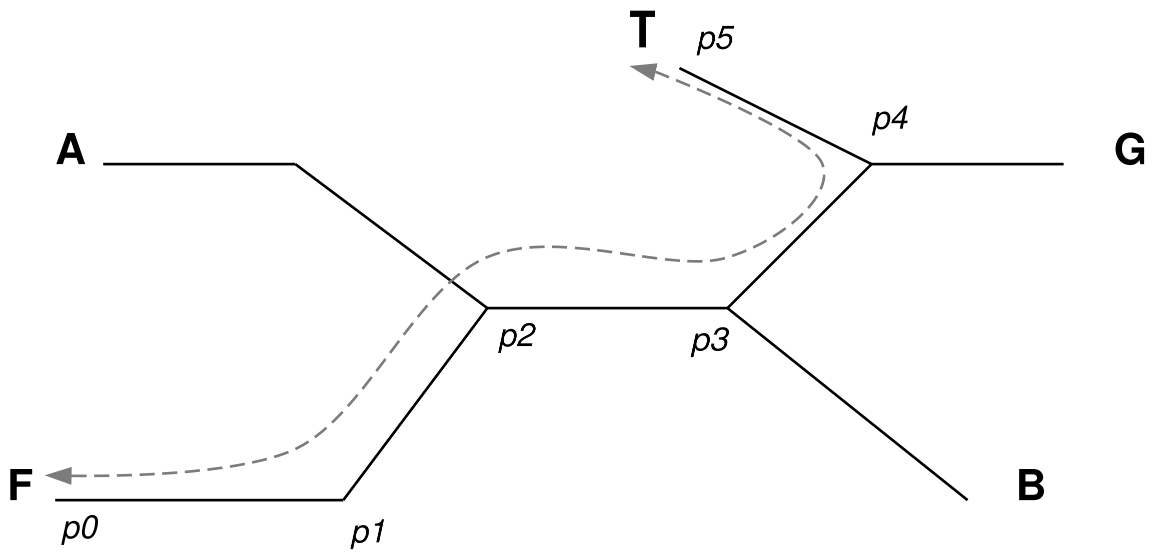

Figure 4.2: A multi-hop network with a single conversation between **F** and **T**. The conversation between **F** and **T** traverses five segments of the network.

The goal of reliable multicast flow control is to assure that all members of the multicast set receive every message. This means that reliable multicast flow control must recover from those errors which do occur. In a multicast setting errors can be corrected by the sender or by a member of the multicast set who successfully received the message.

Many existing point-to-point communication protocols use a "sliding window" for supply control.[2] To address error detection they combine packet numbering with positive or negative acknowledgment. Finally, they use retransmission for error correction.

A second type of flow control protocol uses rate reservation for supply control, combined with the traditional techniques for error detection and error recovery.[3] This

[2]Sliding window protocols are actually a form of indirect rate control.
[3]Error recovery is optional; it is only used when a reliable protocol is desired.

technique is generally used for time sensitive applications where the application must have a guarantee that messages will arrive in a certain amount of time (for example, voice or video communications).

The protocol developed in this chapter will differ from previous approaches by using send-rate control for supply control, combined with standard techniques for error detection and error correction.

4.3 Multicast Network Environment

The network environment in which multicasting is occuring must be understood in order to define the constraints under which a protocol will operate. In chapter 3 multicast performance was evaluated from the application level. At the application level, the network includes the operating system, device driver(s), communications adapter(s), physical wiring, and all intermediate systems which bridge or route packets. This network normally has the following properties:

1. It provides a best effort delivery; messages can be lost, duplicated, or reordered.
2. Processes can join or leave a multicast set without notifying other members of the set.
3. Members of the multicast set are not restricted to multicast communications.
4. Members of a multicast set may belong to more than one multicast set.

This is the typical environment. On a single LAN segment where there are no bridges or routers, messages will normally not be reordered. If reordering does occur, it is usually because of a failure in the adapter or operating system. Reordering is not normally a problem until there are multiple routes to a destination.

4.4 Multicast Network Assumptions

In order to develop a protocol, the following assumptions are made to further define the environment in which the protocols will operate:

1. Messages will arrive in a finite amount of time, which is known or can be calculated.

2. From an error perspective, a multicasted packet looks like n unicast packets.

3. Network performance and errors affect all multicast packets equally. For the n packets in assumption 2, errors are independent and identically distributed.

4. A packet multicasted to n receivers is no more or less likely to be lost than n packets simultaneously unicasted to the same n receivers.

Assumption 1 is reasonable but not absolutely true. It is possible for messages to be delayed indefinitely long in current communication networks. However, for the purposes of error recovery it is common to assume or calculate a maximum allowed delay (or time out) value. Messages which fail to arrive within the specified time or which are in transit longer than this amount of time are assumed to have been lost. Wide area networks use related techniques so that errors in routing tables or other errors will not cause packets to circulate indefinitely or remain queued for transmission indefinitely.

Assumptions 2, 3, and 4 are also reasonable but not necessarily true. Many local area networks — for example, ethernet, token-ring, and FDDI — support multicast directly and therefore do not replicate each multicast packet n times.[4] Thus, there are periods in which a network-induced error in a single packet can affect more than one user. In this case a multicast packet does not look like n unicast packets, as assumed. Similarly, because network performance affects all packets on the network equally and a multicast packet has multiple destinations, a packet multicasted to n receivers does not have the *same* probability of loss as n simultaneously unicasted packets. However, note that in a wide area network a multicast *could* require n separate messages. These assumptions also implicitly assume that the probability of error in a multicast set is independent of the distance between the sender and receiver. In the general case this is also not true. Nevertheless, for highly reliable networks like ethernet, token-ring, or FDDI, these assumptions are reasonably close to reality [SE90,Dee88].[5]

4.5 Communication Patterns

Multicast is by nature a group form of communication where the group is defined by the multicast set. The properties and assumptions previously listed for the multicast network environment do not require that each member of the multicast set knows the membership of the multicast set or that membership of the multicast set is continuous. Membership of a multicast set is defined as *continuous* if for each multicast the

[4] A packet which is multicasted in a WAN will be replicated. Ideally, the replication will not occur until it is necessary to distinguish routes to members of the multicast set [DC90]. This means that, depending on the topology of the multicast set, a single multicast packet *may* result in n packets on the network.

[5] A network is highly reliable if the probability of a network-induced error on a single packet is very low.

membership of the set remains fixed until the multicast succeeds or until a member is known to have failed. Concurrent multicast in the same multicast set could have the same membership. Sequential multicast to the same multicast set from a single process may have different membership. For noncontinuous sets a multicast may be in progress when a process joins or leaves the multicast set.

The model defined by multicast communications has the following properties: Any process p_i in the network can send out a sequence of messages, $M_1, M_2, \cdots, M_n$, each of which is destined for a multicast set, $S_1, S_2, \cdots, S_n$. Multicast communications make no requirement on $S_i \bigcap Sj$ for $i, j \in 1, \cdots, n$. Further, multicast communication does not require that the membership of S_i is known to p_i. Finally, the general model does not require that members of the multicast set want to receive all messages, nor does it state that processes multicasting to the set expect their messages to be received.

For the purposes of the protocols developed in this chapter, the multicast network environment is constrained with a few additional assumptions:

1. Membership of the multicast set is known by all members.

2. Each member of the multicast set wants to receive all messages multicasted to the set.

3. The multicast set can be ordered, and all members know or can compute the same order.

4. The multicast set is continuous.

Any environment which meets these general assumptions will be called a "group communication" environment.

The protocols developed in this chapter are intended for use in a group communication environment. As a result, issues associated with joining a group, detecting the failure of a group member, continuity of the multicast set, and leaving a group are assumed to be handled by the group communication system. The membership of the multicast set is continuous and therefore remains the same until the protocol is notified that it has changed. The protocols can provide interfaces for group change information. The development presented here is primarily concerned with flow control while the multicast set is continuous.

4.6 Flow Control Design Issues

There are a few general issues associated with developing a rate-based flow control protocol for multicast communications which should be taken into consideration when designing a flow control protocol. They are multicast set communication patterns, overlapping multicast sets, implosion, open versus closed loop control, rate synchronization, and timers.

4.6.1 Multicast Set Communication Patterns

Multicast communication is group-based communication. Communication between computers is rate limited, whether the communication is broadcast, multicast, or point-to-point. From the perspective of a single multicast set there are five types of communication that can occur, four of which are illustrated in figure 4.3. The four

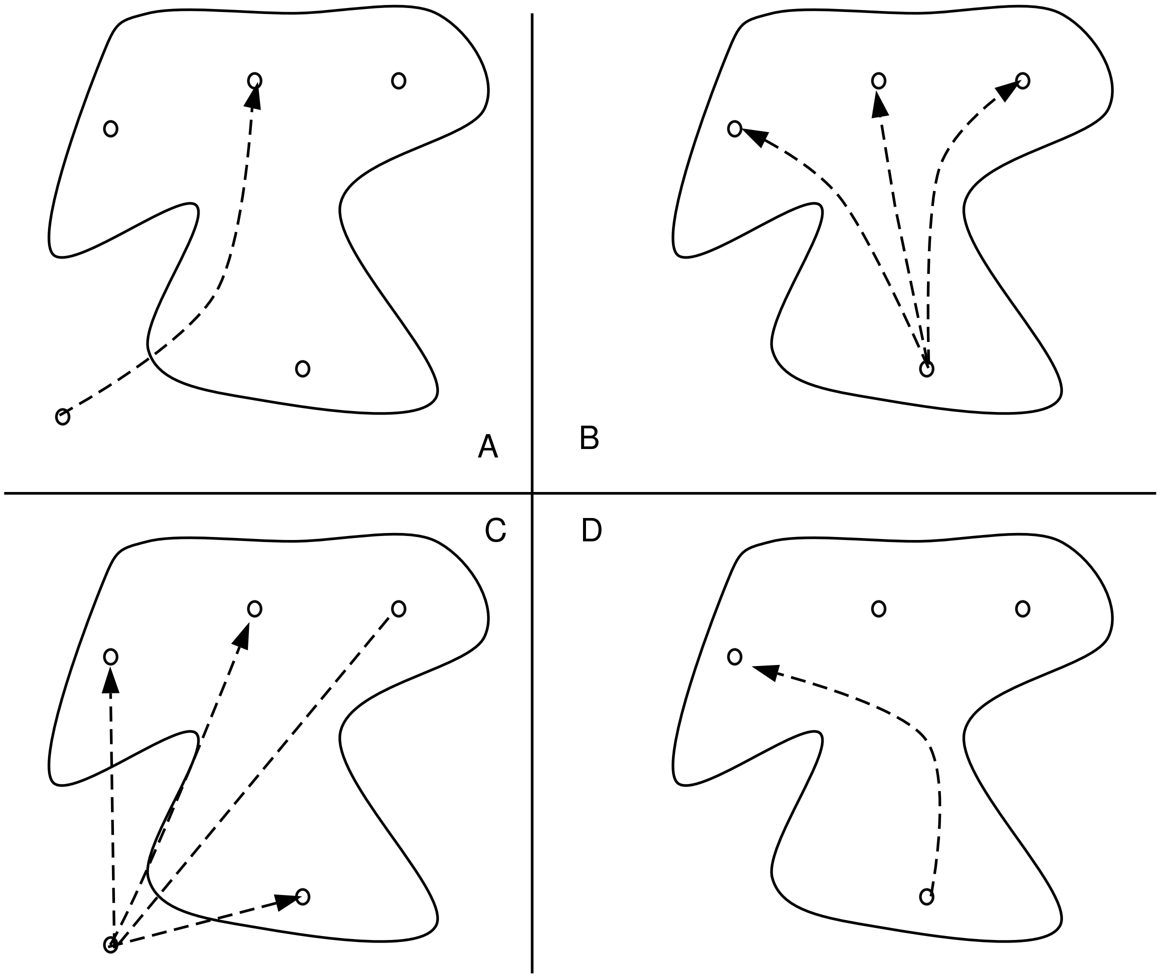

Figure 4.3: Types of communication from the perspective of the multicast set. Quadrant A is point-to-point, quadrant B is multicast inside the multicast set, quadrant C is multicast to the multicast set from outside the multicast set, and quadrant D is point-to-point inside the multicast set.

types illustrated in figure 4.3 can be divided by message type — point-to-point or multicast — and by origin — inside of or outside of the multicast set. Broadcast is a fifth type of message which affects members of the multicast set equally, independent of where it originates. Each type of communication consumes capacity (all forms of communication share the same network) and therefore limits the number of other types of messages that can be received.

A flow control protocol can only adjust those communicating partners which are participating in the protocol. Therefore point-to-point, multicast, or broadcast messages that originate from processes which are not participating in the protocol will not be affected or "controlled" by the protocol. Point-to-point messages to a member of a multicast set can participate in the protocol if a separate multicast group which consists of the two processes involved in the point-to-point communication is formed.

Since a flow control protocol can only affect those systems that are participating in the protocol and communication is rate limited, if there are a large number of messages from systems not participating in the protocol the flow control protocol will be defeated. A flow control protocol is *defeated* when the capacity of the system to receive messages is consumed by messages from systems not participating in the protocol. The protocol developed in this dissertation does not disallow messages from outside the multicast set, but this protocol may be rendered ineffective by them.

4.6.2 Overlapping Multicast Sets

There are two issues associated with overlapped multicast sets; end-to-end flow control across multiple overlapped sets and error leakage between overlapped sets. The

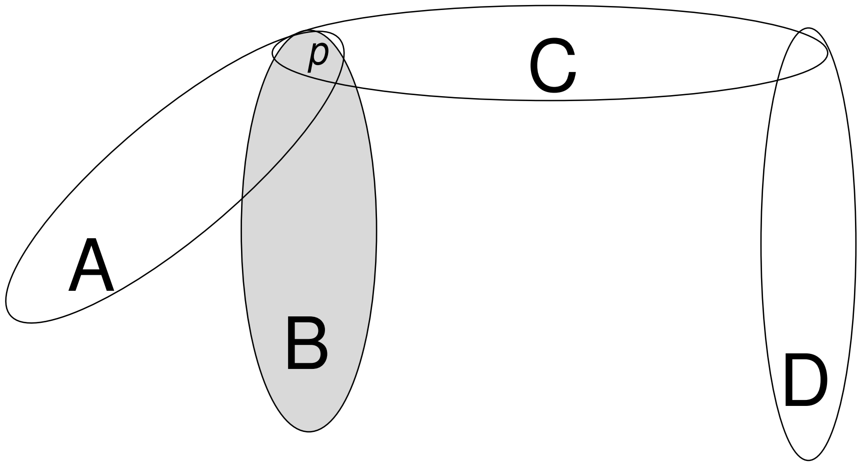

Figure 4.4: This figure illustrates overlapping multicast sets. Process p is a member of Groups A, B, and C. Group B is experiencing errors. The group overlap problem is to develop a protocol so that errors in group B do not unnecessarily affect groups A and C.

protocols developed in this dissertation do not explicitly address end-to-end multi-group flow control issues. This work focuses on flow control within a single multicast set. Flow control across multiple multicast sets is only implicitly addressed by the assumptions of our communications model. These assumptions are not assumed to optimally address this issue.

Error leakage occurs when errors in one multicast set affect an overlapping multicast set. Members of a multicast set are allowed to belong to multiple multicast sets. When multicast sets overlap (have at least one common member), it is desirable for multicast communications within each set to proceed at the fastest possible rate. In figure 4.4, A, B, C and D are multicast sets, $p \in (A \cap B \cap C)$, and multicast set B is experiencing excessive errors. The errors in multicast set B should not unnecessarily affect multicast sets A and C. However, if p is the process which is causing B to slow down, then it is expected that errors at p will also affect multicast sets A and B.

To address this issue our protocol will keep supply control, error detection, and error correction information on a per-group basis. Whether or not errors in one group cause an overlapping group to slow down is probably more dependent on the application design than the flow control protocol. Nevertheless, the flow control protocol should not unnecessarily contribute to this problem.

4.6.3 Implosion

Implosion, broadly defined, is the problem that responses to a multicast can overwhelm the sender[JSW91]. A multicast message which causes an immediate response to the sender could overrun the sender independent of any rate control mechanisms. This is especially true as the size of the group grows. If group sizes are small and the sender is not otherwise overloaded or nearly overloaded, then implosion is not a significant problem. Various solutions have been proposed, from randomly delaying responses [SG92b,PTK94] to a designated acknowledger [Nar90].

4.6.4 Open Loop Versus Closed Loop Control

In general, closed loop control is self limiting and open loop control is not self limiting. From the perspective of flow control, what is being limited is the number of messages that can be sent before an error is detected. Rate-based flow control is an open loop approach; therefore there is no explicit internal limit to the number of messages that can be sent before an error is detected. There are closed loop approaches to flow control; for example, a windowed approach is closed looped because the number of messages that can be sent before an error is detected is limited by the window size, independent of other factors.

For rate-based methods the number of messages that can be sent (or multicasted) before an error is detected is a function of the transmission rate and the error detection time. The error detection time is bounded below by the round-trip delay and above by the error time-out. In the general case the error time out is calculated (a multiple greater than one) from the maximum round trip delay. For multicast communication the maximum round trip time generalizes to the round trip delay to the slowest member of the group.

For unreliable protocols open loop versus closed loop is not much of an issue because detected errors are not corrected. However, for reliable protocols there is an issue, because the sender has to be able to correct detected errors. Therefore a reliable protocol has to retain transmitted messages until it knows that they have been received; otherwise errors cannot be corrected.[6] Additionally, depending on whether the protocol has ordering guarantees, receivers may also have to buffer messages[7] when an error occurs. For example, if the protocol is reliable and ordered, given a sequence of messages, $m_1, m_2, \cdots, m_n, \cdots$, if an error occurs on $m_i (i < n)$ which is detected after m_n has been transmitted, after retransmitting m_i the protocol can continue with either m_{i+1} or m_{n+1}. In order to continue with m_{n+1} the receive protocol must buffer $m_{i+1} \cdots m_n$ until the error at m_i is corrected. This can be problematic, given the unlimited nature of open loop control methods. Even if the underlying protocol does not guarantee order, the buffering requirement is imposed on any layer above

[6] Actually the message has to be retained by the transmitter or someone who successfully received it.

[7] Buffering messages is actually a performance optimization. The protocol could start retransmitting after a known error. All receivers would have to discard duplicate messages.

the protocol which guarantees order. However, rate-based methods can easily be combined with other techniques to convert them to closed loop control and thereby limit the exposure to buffering issues.

4.6.5 Rate Synchronization

Send-rate control applied to multicast communication means adjusting the rate of transmission[8] of group members based on the presence or absence of errors. Send-rate control can be applied without fairness. *Fairness* means dividing the transmission capacity of the group evenly. Fairness is a reasonable property for a flow control protocol; however, if desired, fairness does not need to be continuously enforced. Continuous enforcement of fairness would require rate synchronization. Instead of requiring fairness, rates can be periodically resynchronized to redistribute transmission capacity.

Rates can be easily resynchronized around group changes, and there are a number of resynchronization protocols which can be used. One such protocol is the following:[9] When the flow control protocol receives a group change notification, the leader[10] of the group multicasts a message to all members of the group requesting that they each send their current rate and the number of messages they have multicast since the last rate resynchronization period. All group members respond with the requested information.[11] After collecting all of the responses, the leader averages the send-

[8] Minimum intermessage gap.

[9] This is a high level description and therefore does not specify what to do for all the error cases. This protocol is subject to the same environment and assumptions.

[10] The group member who is ordered first by all members of the group; see assumption 3 on page 56.

[11] Failures of group members during the protocol can be easily handled and are not addressed.

ing rates of all group members who have multicasted messages since the last rate resynchronization. This average is multicasted to the group as the new initial rate.

If groups are extremely stable and the rate adjustment messages are occasionally lost, in theory a single member could get an unfair share of the rate. This problem can be avoided by establishing a rate resynchronization timer in combination with resynchronizing rates around group change. The leader of the group sets the rate resynchronization timer whenever group membership changes. If this timer expires before a membership change occurs, the leader resets the timer and then runs the same resynchronization protocol defined for group change.[12]

Rate resynchronization is ignored in this development of the protocol because the development was not integrated into a a group communications environment.

4.6.6 Timers

For rate-based flow control, timers are a central part of the approach. Time will be used to perform the following functions:

- Measuring and enforcing intermessage spacing.
- Detecting increasing network capacity.
- Determining sensitivity of the protocol to consecutive errors.
- Detecting message acknowledgment timeout.

These functions are implemented by simple timers driven by the wall clock. The granularity of the timers and the relative size of the timers determines the respon-

[12] There is no requirement to delay messages while the resynchronization protocol is running.

siveness and sensitivity of the algorithm. Optimizing timers is a challenging task. The optimal values for timer driven events or protocols often vary depending on the circumstances. No attempt has been made to find optimal values. This presentation uses timers that demonstrate the viability of the approach.

4.7 Initial Protocol

The goal of the initial algorithm is to demonstrate whether or not send-rate control can be effectively used for supply control for multicast communications. The initial protocol can be easily described in terms of the three functions of flow control: supply control, error detection and error correction. This protocol uses send-rate control with positive acknowledgments for supply control. Error detection is done by numbering the packets combined with positive acknowledgment and an acknowledgment timeout. For error correction retransmission is used; the sender uses point-to-point retransmission if only one member of the multicast set misses the messages and uses multicast retransmission otherwise. This protocol is closed loop flow control because of the positive acknowledgment.

The transmission rate is implemented by enforcing a minimum separation between messages. This separation is increased or decreased to constrain or release the supply of messages. Acknowledgments are not subject to the minimum separation.

The algorithm has four "timers": *rate-increase timer*, *rate-decrease timer*, *inter-message timer*, and *acknowledgment timer*. The timers are all approximate, because the implementation computes the time (wall clock) when each event is next scheduled to occur and checks regularly whether or not a given timer has expired. Therefore,

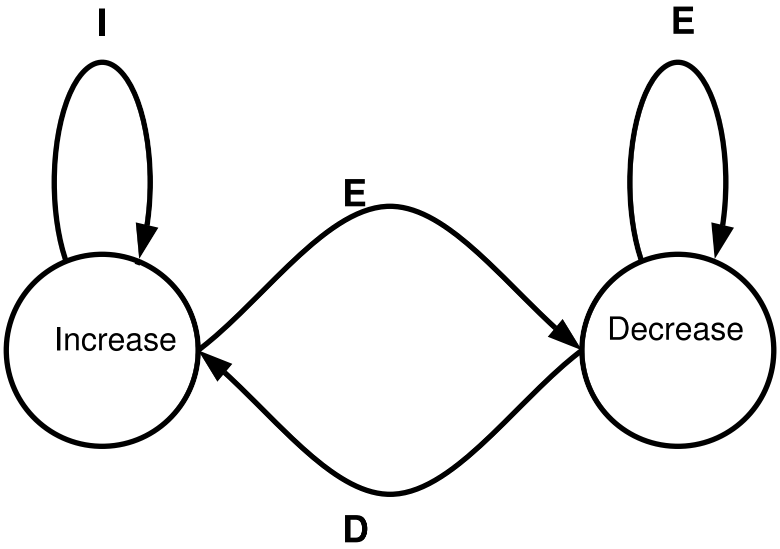

Figure 4.5: State transition diagram representing flow control protocol.

the timers specify the minimum amount of time until the event occurs, and an event may not be "detected" until some time after it should have "occured".

The protocol can be completely described by the state transition diagram of figure 4.5. State transitions in this diagram are caused by a timer expiring or error detection. The three inputs **I**, **D**, and **E** in figure 4.5 represent the increase timer expiring, the decrease timer expiring, or an error being detected for the current packet. In increase mode the decrease timer is never set, and in decrease mode the increase timer is never set. The protocol's operation is defined in table 4.1.

To address implosion the multicast sets will be kept fairly small. For larger multicast sets the methods of section 4.6.3 can be combined with the proposed protocol. Even though the multicast sets are being kept small, we can measure the effectiveness of rate-based flow control in this environment.

Table 4.1: Operation of protocol.

State	Input	Operation
Increase	I	The increase timer has expired. Therefore decrease the transmission interval (which must stay positive), remain in increase state, and set another increase timeout.
Increase	E	An error has been detected. Clear the increase timer and set the decrease timer. Mark the protocol as in decrease state. Increase the intermessage spacing by two times the last increase.
Decrease	E	An error has been detected. Increase the interval by two times the last increase and start another decrease timer.
Decrease	D	The decrease timer expired without an error occuring. Set the last decrease multiplier to one. Start an increase timer and shift to increase mode.

Because positive acknowledgment is used, the protocol is closed loop, and therefore the buffering problems of open loop control are avoided.

Even though positive acknowledgment is in and of itself a flow control method, the protocol which has been proposed will allow us to study the effectiveness of direct rate control for multicast flow control. The protocol developed in this chapter differs from previous approaches by using send-rate control for supply control. Other methods of supply control do not directly affect the rate, which can cause problems in a multicast environment where there are multiple senders.

4.8 Summary

This chapter has presented a brief review of flow control which partitioned the problem into three distinct functions: supply control, error detection, and error correction. A definition of multicast flow control was presented which included definitions of the multicast network environment and some further assumptions which the multicast network environment is required to meet in order for the flow control protocol proposed in this chapter to function. Earlier in this dissertation it was suggested that a general protocol could be developed which is independent of communication patterns. In this chapter it was necessary to constrain the communications environment with a few simple assumptions in order to define an environment where the protocols would operate. Further, it was pointed out that because communication is rate limited, any protocol can be defeated by messages from nonparticipating systems. Finally, this chapter presented a review of issues associated with multicast flow control design followed by a presentation of the initial protocol which will be studied for the remainder of this dissertation.

Chapter 5

Experimental Results

In this chapter a study of the performance of protocols based on the design discussion of chapter 4 and the observations of chapter 3 will be presented. A protocol for unreliable multicast flow control and a protocol for reliable multicast flow control will be studied. In addition, the reliable flow control protocol will be applied to the problem of disseminating messages to large groups.

The primary focus of these studies is determining the viability of send-rate control as a component of supply control for multicast flow control protocols. Therefore the protocols that are presented in this chapter are not presumed to be optimally implemented or optimized for the problems they are addressing.

In the course of performing this work, different networks, computer systems, and interfaces have been used for examining the performance of the protocols. Studying and implementing computer protocols puts a heavy load[1] on the systems under test

[1] The types of studies that were performed for parts of this work tended to crash the networks and systems that were used.

and takes time; as a result the systems that were initially available were not available later in the progress of this work. Implementing a protocol involves imposing some overhead on the performance measured in chapter 3. Since the additional studies were not done on the systems and interfaces on which the initial performance measurements were made, calibrations will be presented so that the overhead imposed by the implementation can be accurately projected.

This chapter starts with a study of an unreliable protocol, followed by a study of a reliable protocol, and concludes with a study of how to extend and apply the reliable protocol to the problem of disseminating information to large groups of processes.

5.1 Unreliable Flow Control

The goal of unreliable flow control is to allow as many packets as possible to pass between participants with zero or minimal errors while not correcting for errors. Therefore, unreliable flow control protocols contain supply control and error detection but no error correction.[2] Information from error detection is used by supply control to constrain or release the supply of packets. For rate-based protocols like the one presented in section 5.1.2, this is equivalent to increasing or decreasing the minimum intermessage spacing in response to errors.

Unreliable multicast flow control is generally used in broadcast type applications where there is one sender per group and many receivers — for example, Internet radio and Internet TV. Some protocols which have been optimized for this case — i.e., single sender and multiple receivers — have been published[CP88,Nar90]. Our

[2] Error correction is the responsibility of the users of an unreliable protocol.

goal here is to examine whether or not the rate-based approaches we are studying have promise for this problem.

This study will also examine whether or not an algorithm which does not take advantage of the broadcast nature of the problem has any promise. In broadcast protocols the sender does not usually receive its own multicast and the number of senders in a group is limited to one. In the protocol presented below the sender is a full member of the group and therefore receives its own multicast. For this environment, performance of a single sender multicast set is examined and performance of a multisender multicast set is also examined.

5.1.1 Methodology

A limited amount of group semantics were added to the same test system that was used for the performance analysis reported in chapter 3. Specifically, the test system was enhanced so that:

1. Multicast sets are statically constructed and remain for the duration of each test.

2. Senders and receivers are members of the same multicast set.

3. Within a sender, the sending and receiving functions are synchronous.

With these changes the test system meets the communication pattern assumptions made in section 4.5 on page 56, which are necessary to enable the protocol to operate.

The study could have been done by integrating the protocol into a group communications system; however, by not integrating the protocol into a group communications system it is easier to study the central issue: whether or not send-rate control can be an effective part of flow control. Further, reusing the test system means that the results are directly comparable to the results of chapter 3.

The test system still has senders and receivers running at the application level on separate machines, and therefore, because multicast sets are statically constructed, failure detection, joining groups, and leaving groups can be ignored.[3] In this environment the test system is designed for "steady state" testing. The sender sends messages and then terminates. As a result we are studying the protocol under load.

The test system and the protocol presented below do not contain rate resynchronization. The initial rate is established for the first message of the first test point. This rate is adjusted by the protocol as the test proceeds. As each test point is completed, the final rate that was used is the initial rate for the next test point. This methodology is particularly dangerous for unreliable flow control because there are no pseudosynchronization points in the problem. Therefore the protocol is probably much more sensitive to timer values. Consequently, as the number of senders increases there is a greater possibility that their rates (minimum intermessage spacing) will diverge.

The sending and receiving functions within a sender are synchronous; each sender sends a message and then receives messages only while it is waiting for acknowledgments for the message that was just sent. Performance (probably) and complexity would be increased if the send and receive functions of the senders were asynchronous. However, an asynchronous implementation probably would not have added additional

[3]They can be ignored as long as the protocol does not complicate these tasks and the overhead associated with these tasks does not directly affect the performance of the protocol.

insight into the viability of the approach.

Because the protocol is positive acknowledgment, the sending test has to be modified to force senders to continue to receive messages after they have finished sending all of their messages for each test. Otherwise the other senders participating in the test will not be able to correctly complete their tests.

To test the concept of rate-based flow control, the approach was simplified by making the acknowledgment, increment, and decrement timeouts fixed values which were not adjusted based on the performance of the system. In general, these values should be dynamic functions of the size of the multicast set, the current intermessage delay, and the maximum round trip time to a member of the group, and perhaps other factors. As was pointed out in chapter 4, optimizing timers is a complex problem, so selecting fixed values simplified testing the approach even though it probably has affected scalability.

This protocol was implemented in an environment where back pressure was available from the interface used to multicast. The back pressure was used by the protocol to assure that there were no send omissions

5.1.2 Protocol

Communications protocols which operate over serial interfaces where no out-of-ban data is present are usually implemented using layering and encapsulation. Each layer encapsulates the data passed to it from above before sending it to the layer below. The data are encapsulated with a header and optionally a trailer. When two or more systems are communicating, each layer of the protocol communicates with its corresponding layer.

The protocols as presented here are abstractions[4] of what was used to test the concept of rate-based multicast flow control. If this protocol were implemented in a general purpose system, a number of additional issues would have to be addressed. For example, in the general case applications can send only, receive only, or send and receive. Therefore, for the general environment the send and receive side of the protocol must share a queue on which the send side of the protocol queues data which is received while waiting for acknowledgments.[5] The receive side of the protocol must use this queue and the network as sources of data for the application. The semantics of the acknowledgment may also become an issue, Does the acknowledgment represent the protocol layer or a higher layer? For the purposes of this research the acknowledgment is assumed to represent the protocol layer. Because the protocol is implemented as subroutines of the test system, this also corresponds to the application system.

Program 5.1.1 represents the send side of the test protocol, and program 5.1.2 represents the receive side. The programs presented here are pseudocode for the code used to implement the test protocol. Table 5.1 defined the function of the subroutines used in the pseudocode.

Protocol 5.1.1 (Unreliable send protocol)

```
proc multicast_send(mcast_grp, mem_num, msg);
        if (¬send_ok(mcast_grp)) then delay(mcast_grp);  fi
        multicast(msg, mcast_grp, mem_num);
        enough_acks = 0;
```

[4]The abstractions leave out tracing code, details of manipulating internal data structures, code used for internal error detection and correction, and special code for the test environment.

[5]Unless it knows that the application is never going to receive.

```
    while (¬(enough_acks == group_size(mcast_grp))) do
        receive(resp, mcast_grp, mem_num);
        switch (resp);
            case: (mcast_msg);
                ack(resp, mcast_grp, mem_num);
                if (¬(expected_msg(resp, mcast_grp))
                  do
                     nack(EXP, resp, mcast_grp, mem_num);
                     adjust_rate_on_error(mcast_grp);
                  od
                fi
                update(LAST_MSG, resp, mcast_grp);
            break;
            case: (mcast_ack);
                if (new(resp, mcast_grp))
                  then enough_acks+ = 1;
                fi
            break;
            case: (mcast_nack);
                if (new(resp, mcast_grp))
                  do
                     update(MISSED, resp, mcast_grp);
                     adjust_rate_on_error(mcast_grp);
                  od
                fi
            break;
        endswitch;
        check_inc_dec_timer(mcast_grp);
        if (¬(enough_acks == group_size(mcast_grp)) ∧
           (ack_timer_expired(mcast_grp)))
           do
               adjust_rate_on_error(mcast_grp);
               nack(LAST, dummy_msg, mcast_grp, mem_num);
               enough_acks = group_size(mcast_grp);
           od
        fi
    od
```

Protocol 5.1.2 (Unreliable receive protocol)

```
proc multicast_receive(mcast_grp, mem_num, ret_msg);
```

```
while (¬new_msg(receive(ret_msg, mcast_grp, mem_num))) do
        check_inc_dec_timer(mcast_grp);
        switch (ret_msg);
                case: (mcast_msg);
                        ack(ret_msg, mcast_grp, mem_num);
                        if (¬(expected_msg(ret_msg, mcast_grp))
                          do
                                nack(EXP, ret_msg, mcast_grp, mem_num);
                                adjust_rate_on_error(mcast_grp);
                          od
                        fi
                        update(LAST_MSG, ret_msg, mcast_grp);
                        return ;
                break;
                case: (mcast_ack); comment : Not needed for receive
                        if (new(ret_msg, mcast_grp))
                          then enough_acks+ = 1;
                        fi
                break;
                case: (mcast_nack);
                        if (new(ret_msg, mcast_grp))
                          do
                                update(MISSED, ret_msg, mcast_grp);
                                adjust_rate_on_error(mcast_grp);
                          od
                        fi
                break;
        endswitch;
od
```

On the sending side, protocol 5.1.1 has been optimized to discard incoming packets after they are received. Since the sender never processes the data, this avoids having to queue the incoming packets for the receive side of the protocol. The performance of the senders is slightly better because this optimization was done.

The protocol presented here is implemented as a layer in this type of communication system. It uses a header for encapsulating data and representing control

Table 5.1: Definitions of functions used in pseudocode for the unreliable flow control algorithms.

ack	Sends the group an acknowledgment for the indicated message.
ack_timer_expired	True if the acknowledgment timer has expired.
adjust_rate_on_error	Shifts the protocol into ready to decrease state, increasing the intermessage spacing and setting the decrement timer.
check_inc_dec_timer	Checks the increment/decrement timer and shifts the protocol state, resets the timer, or decreases the intermessage spacing if necessary.
delay	Waits until the minimum intermessage spacing has elapsed.
expected_msg	True if the message is the next expected message from the sender or the message has been seen before.
group_size	Returns the size of the multicast set.
multicast	Sends a multicast to the group.
nack	*EXP* Multicast a negative acknowledgment for the next expected message from the sender of *resp*. *LAST* Multicast a negative acknowledgment for the last message which was sent to this group.
new	This returns true if the indicated packet has not been seen before.
receive	Receives a message from the group.
send_ok	True if the minimum intermessage spacing has occured since the last send.
update	*LAST_MSG* Updates the last received message from the sender of the current message. *MISSED* Updates the last missed message based on the information in the message.

messages. This protocol sends three types of messages to the corresponding layer: multicast message, multicast acknowledgment, and multicast negative acknowledgment. The multicast message contains data multicasted to the group. The multicast acknowledgment is sent point-to-point back to the sender of a multicast to acknowledge receipt of a packet. The multicast negative acknowledgment is multicasted to the group whenever a receiver notices that a packet has been lost or whenever a sender knows that a packet was not received by everyone in the group. If a receiver notices that a sequence of messages has been missed, a negative acknowledgment is generated only for the first message in the sequence.

5.1.3 Calibration

The systems used for the unreliable multicast evaluation were not the same as the systems used for the performance analysis of multicast on ethernet in section 3.2.4. To calibrate the two different environments a subset of the ethernet multicast performance analysis tests that were used to produce figure 3.21 in section 3.2.4 were run in the new environment. Figure 5.1 shows the results of the calibration test run. These data look the same as the data presented in section 3.2.4 and show clearly that a rate limitation exists in the new environment. In order to determine the relative performance of the two environments, the total senders' performance in each environment for the tests with 2 senders and the tests with 4 senders are compared in figures 5.2 and 5.3. For both environments the tests were run with lightly loaded systems and networks. The figures show that the total senders' performance in the two environments are indistinguishable, and therefore suggest that the environments

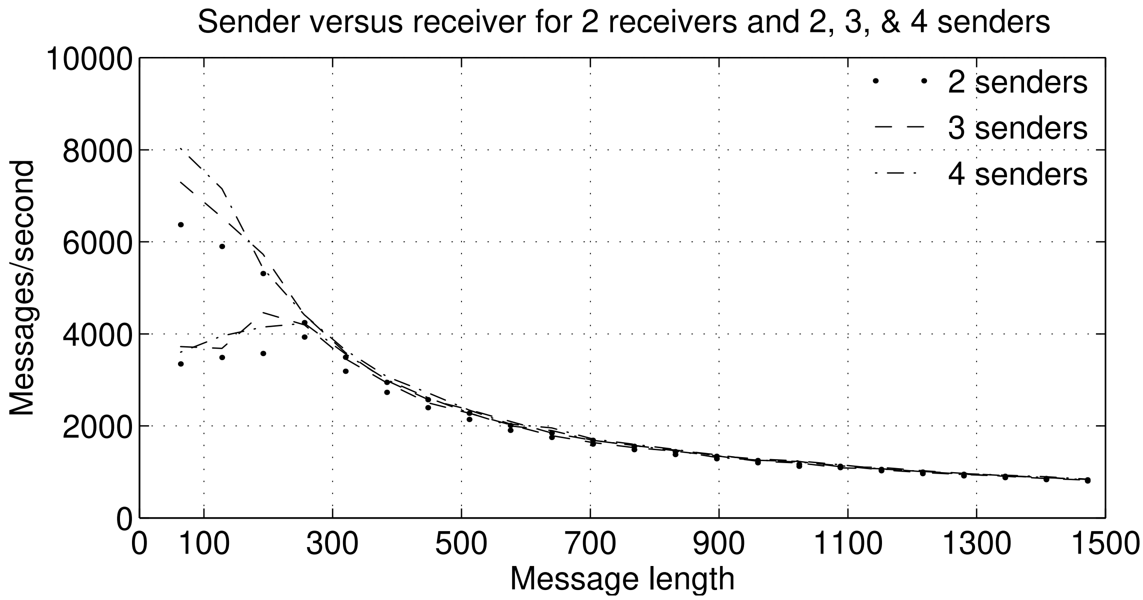

Figure 5.1: Ethernet multicast rate comparison for 2, 3, and 4 senders and 2 receivers.

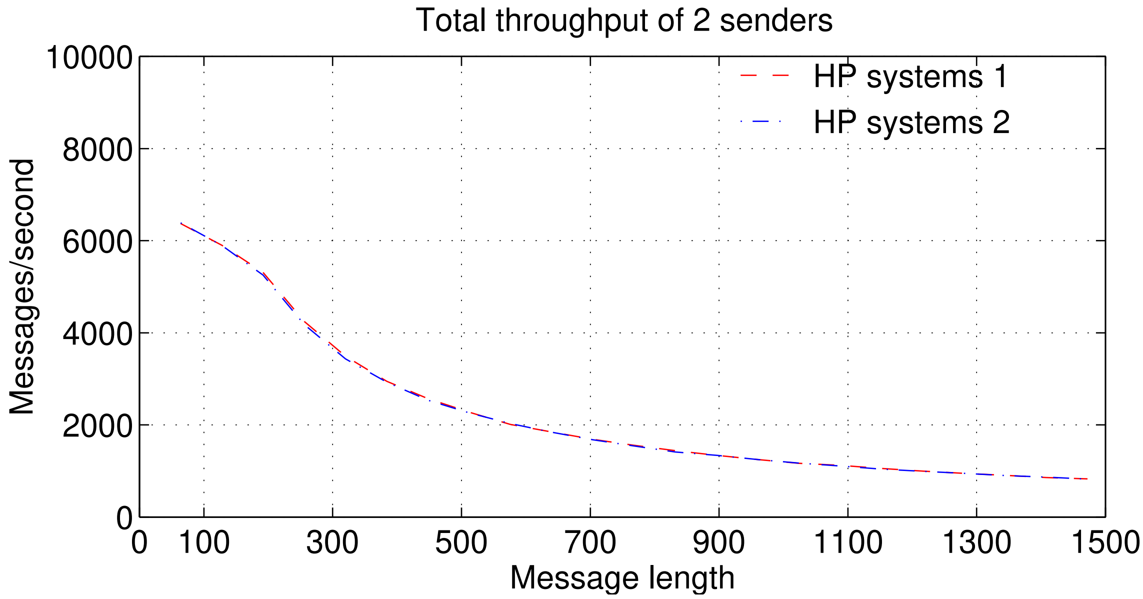

Figure 5.2: Comparison of the total senders' throughput between the two sets of HP systems for test in which there were 2 senders and 2 receivers.

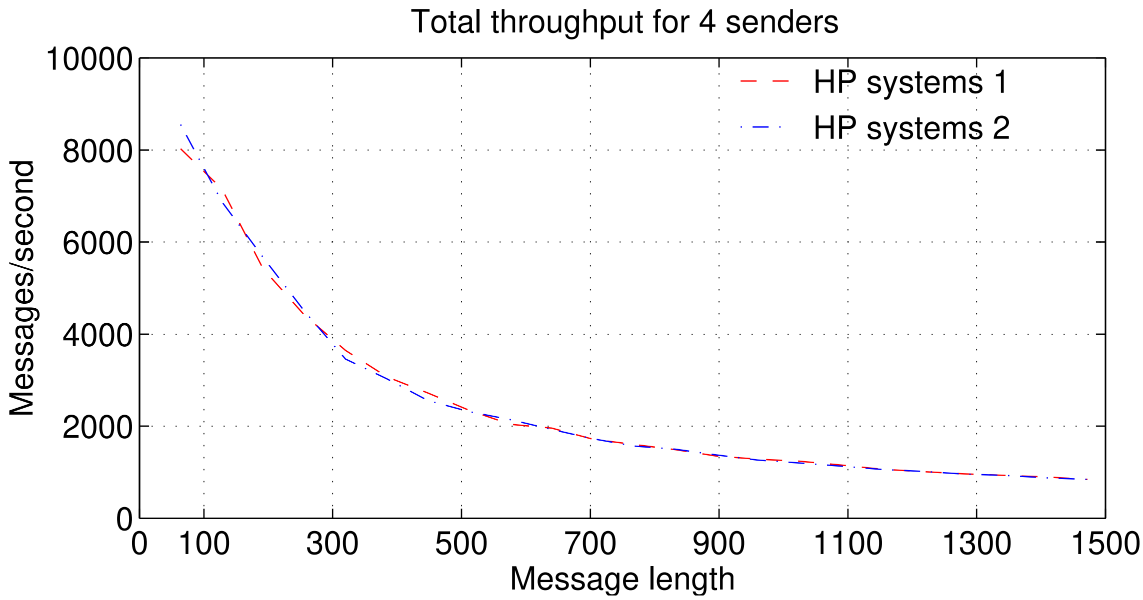

Figure 5.3: Comparison of the total senders' throughput between the two sets of HP computers for test in which there were 4 senders and 2 receivers.

are essentially identical.

5.1.4 Experimental Results

The goal of this portion of the experimental work is to determine if send-rate control can be an effective component of unreliable multicast flow control. These tests are not designed to test the capacity of the network. Therefore the number of messages that were transmitted at each point was reduced to 1000 from 10,000.

In the test system a receiver has considerably less overhead than a sender; therefore receivers lose no data. This is partially a side effect of how the protocol is "tuned" and how the test system is implemented. In all of the tests that were run, no losses were recorded by the receivers; therefore all of the losses were occuring at the sending side. The test system did not keep track of these losses. If the test system had tracked

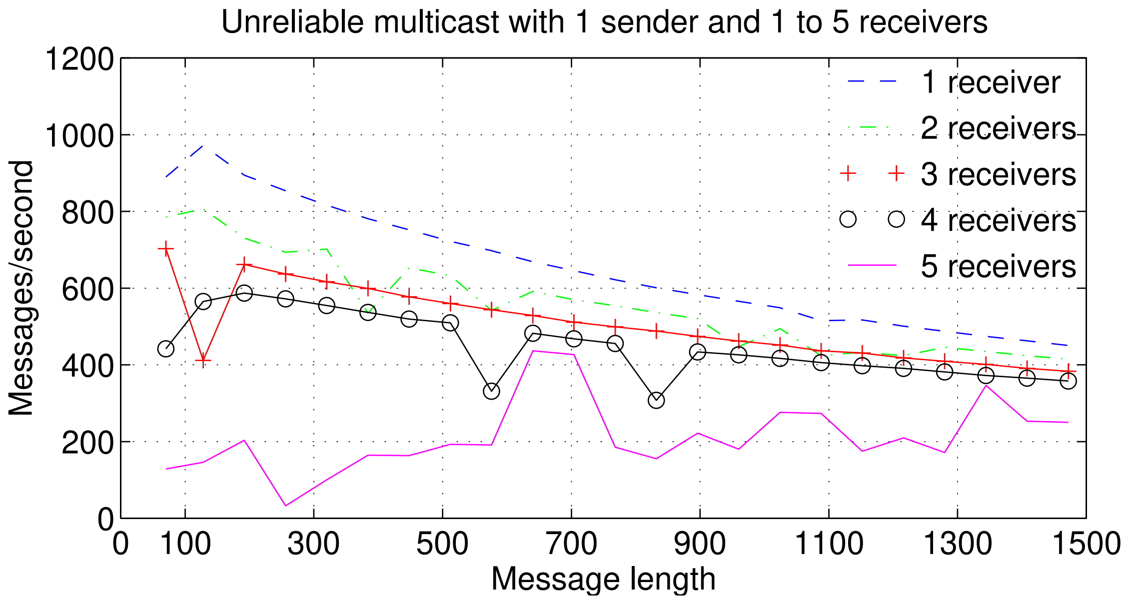

Figure 5.4: Comparison of a single sender's performance for the unreliable multicast protocol when the number of receivers is varied from 1 to 5.

losses at the sending side, then some approximate measure of "efficiency" could have been made.

Figure 5.4 shows the results of running the algorithm in groups with 2 to 6 members and only 1 sender. The performance degrades as the size of the group grows, because the sender spends increasing amounts of time processing acknowledgments. Because these tests were run on lightly loaded machines on shared networks, there is a possibility that some other task contributed to the instabilities. However, it is more likely that acknowledgment time is contributing to increased instability. Since the protocol is using fixed time values instead of variable ones, as more time is spent processing acknowledgments there is an increased probability that messages will be lost or assumed lost.

Figure 5.5 compares the total sender's performance between the calibration test

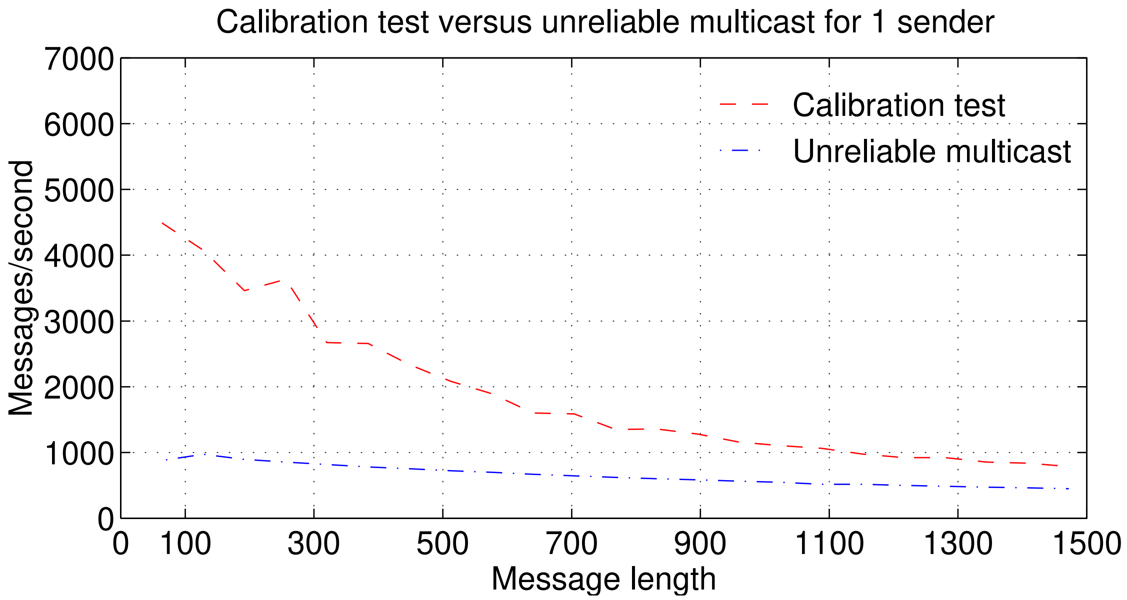

Figure 5.5: Comparison of the sender's performance in the calibration test to the sender's performance with unreliable multicast flow control for tests in which there was 1 sender.

and a test of the unreliable multicast flow control algorithm. The calibration test was run with 1 sender and 2 receivers; the unreliable flow control test was run with 1 sender and 1 receiver. This comparison gives an indication of the overhead imposed on a single sender while using the algorithm. As would be expected, it reduces as the message length increases because more time is spent sending each message.

Figure 5.6 compares the total senders' performance between the calibration test and a test of the unreliable flow control algorithm. The calibration test was run with 2 senders and 2 receivers; the unreliable flow control test was run with 2 senders and 1 receiver. The same general trend is observed as in figure 5.5.

Figure 5.7 compares the results of the senders' performance in the two unreliable flow control tests previously presented. As expected, the test with two senders had

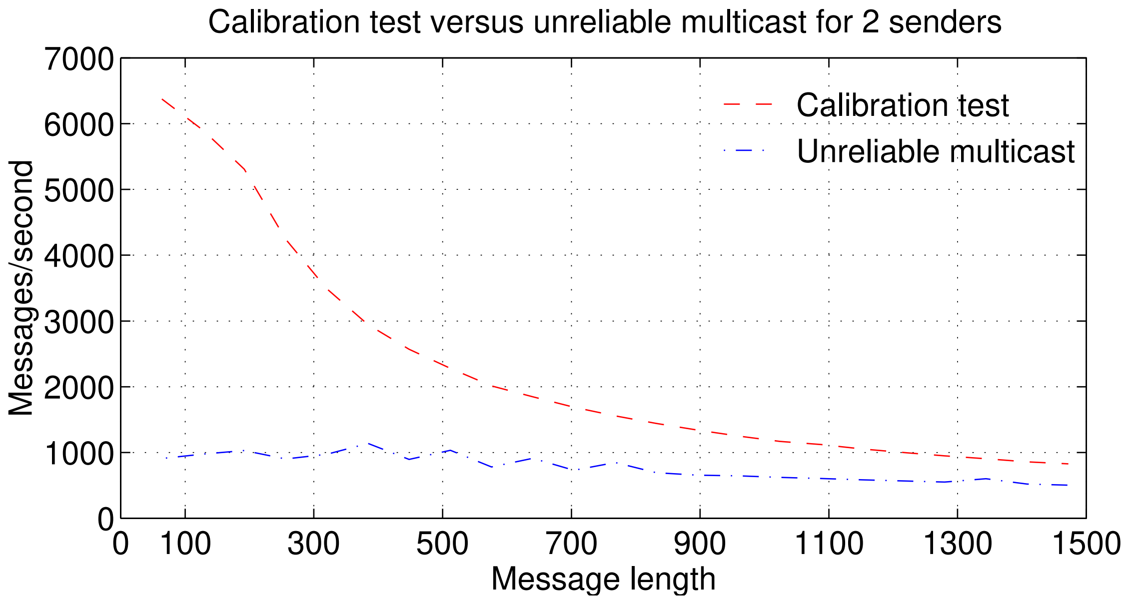

Figure 5.6: Comparison of the total senders' performance in the calibration test to the total senders' performance with unreliable multicast flow control for tests in which there were 2 senders.

slightly better performance than the test with one sender. However the test system failed when an attempt was made to run a test with 3 senders and 1 receiver. The failure logs point to one sender's minimum intermessage spacing becoming extremely unsynchronized with respect to the other two. This caused a failure in the test system because it uses timeouts to check the operation of the protocol. When the protocol takes too long in a test cycle the test system assumes it has failed and aborts.

5.1.5 Conclusions

The results presented show that send-rate control was an effective technique for unreliable multicast flow control in the environments where it was tested. The protocol allows the senders to proceed at a reasonable rate which does not exceed the rate limit for pure receivers as measured by the calibration tests. This indicates that the

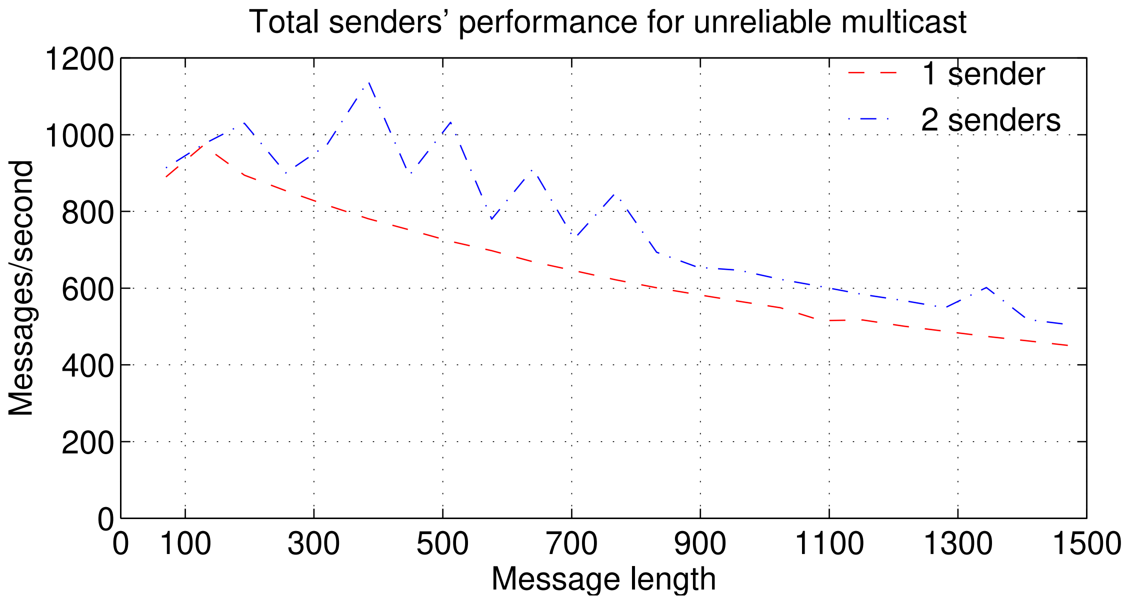

Figure 5.7: Comparison of the total senders' performance for unreliable multicast flow control tests in which there were 1 and 2 senders.

protocol as tested is "send" limited. This implementation of a general protocol does not scale well, because static values are used for the timers and responses are required from every receiver for every message. To improve the performance of the protocol the timer values must be made dependent on the size of the multicast set, inter-message spacing, and maximum round-trip time to a member of the multicast set. Additionally, pseudowindows should be implemented so that positive acknowledgments are not required from every member for every message. Finally, for unreliable multicast, resynchronization of the rates is probably more critical than for reliable multicast. Even though the protocol as tested is not fully generalized, these results have demonstrated that send-rate control can be applied to unreliable multicast flow control.

5.2 Reliable Flow Control

Reliable flow control differs from unreliable flow control in that error correction is implemented. Specifically, the multicast send does not proceed until acknowledgments have been received from all members of the multicast set. When errors are occuring this causes the reliable protocol to run slower than the unreliable protocol. However, the sender knows that each message has been "received".[6]

5.2.1 Methodology

The same system was used to test reliable multicast flow control as was used to test the unreliable flow control protocol. Only the flow control protocols had to be changed for the tests.

Even though the same test system was used for the reliable and unreliable multicast flow control protocol testing, there is a key difference between the two. Because the stop-and-wait reliable send protocol presented below does not proceed until all members of the multicast set receive each packet, the rate at which senders' minimum intermessage spacing (send rate) can "drift" out of synchronization is slowed. This is a "pseudosynchronization" point. However, because there is no explicit rate resynchronization, nothing prevents the senders from becoming widely skewed. The protocol presented below is prevented from taking full advantage of the "pseudosynchronization" point by the fixed timeouts that were implemented as a simplification.

[6]Subject to the semantics of an acknowledgment.

5.2.2 Protocol

Protocol 5.2.1 is the reliable send protocol, which is similar to the unreliable protocol. The principal difference between the reliable and unreliable protocols is that for reliable multicast the sender does not multicast another packet until it knows that the previous packet has been received by all members of the multicast set. This difference causes changes in the handling of multicast packets (lines 10–24), and error correction code has to be added (lines 3, 39–55, 58–59). For error correction, this protocol remulticasts the message if more than one member of the multicast set misses the message; otherwise the correction is sent point-to-point (lines 39–55). Because packets are being retransmitted to correct errors, the protocol cannot unconditionally update the last message received information as was done in the unreliable protocol. Instead it must distinguish between old packets, the expected packet, and unexpected packets (lines 10–24) and act accordingly.

Protocol 5.2.1 (Reliable send protocol)

```
1  proc multicast_send(mcast_grp, mem_num, msg);
2      not_sent = TRUE;
3      while (not_sent) do
4          if (¬send_ok(mcast_grp)) then delay(mcast_grp); fi
5          multicast(msg, mcast_grp, mem_num);
6          enough_acks = 0;
7          while (¬(enough_acks == group_size(mcast_grp))) do
8              receive(resp, mcast_grp, mem_num);
9              switch (resp);
10                 case: (mcast_msg);
11                     if (old_msg(resp, mcast_grp))
12                       ack(resp, mcast_grp, mem_num);
13                     elsif (expected_msg(resp, mcast_grp))
14                         do
15                             ack(resp, mcast_grp, mem_num);
```

```
                        update(LAST_MSG, resp, mcast_grp);
                    od
                else
                    do
                        nack(EXP, resp, mcast_grp, mem_num);
                        adjust_rate_on_error(mcast_grp);
                    od
                fi
            break;
            case: (mcast_ack);
                if (new(resp, mcast_grp))
                  then enough_acks+ = 1;
                fi
            break;
            case: (mcast_nack);
                if (new(resp, mcast_grp))
                  do
                      update(MISSED, resp, mcast_grp);
                      adjust_rate_on_error(mcast_grp);
                  od
                fi
            break;
        endswitch;
        check_inc_dec_timer(mcast_grp);
        if (enough_acks == group_size(mcast_grp))
          not_sent = FALSE;
        elsif (ack_timer_expired(mcast_grp))
            do
                adjust_rate_on_error(mcast_grp);
                nack(LAST, dummy_msg, mcast_grp, mem_num);
                if (1 == (group_size(mcast_grp) − enough_acks))
                  do
                      send_to_one(msg, mcast_grp, mem_num);
                      reset_ack_timer(mcast_grp);
                  od
                elsf
                    enough_acks = group_size(mcast_grp);
                fi
            od
        fi
```

```
                comment : End of waiting for enough acks.
            od
            comment : End of waiting for message to be sent
    od
```

Program 5.2.2 is the receive side of the reliable multicast protocol. This program has changes (lines 5–21) which correspond to the changes made to the send protocol. It differentiates between old packets, the expected packet, and unexpected packets. The code for unexpected packets is outlined here; however, for a stop and wait protocol no unexpected packets should ever occur. As before, the last message received is only recorded for the expected packets.

Protocol 5.2.2 (Reliable receive protocol)

```
1  proc multicast_receive(mcast_grp, mem_num, ret_msg);
2      while (¬new_msg(receive(ret_msg, mcast_grp, mem_num))) do
3          check_inc_dec_timer(mcast_grp);
4          switch (ret_msg);
5              case: (mcast_msg);
6                  if (wanted_msg(ret_msg, mcast_grp))
7                      do
8                          ack(ret_msg, mcast_grp, mem_num);
9                          update(LAST_MSG, ret_msg, mcast_grp);
10                         return ;
11                     od
12                 elsf (old_msg(ret_msg, mcast_grp))
13                         ack(ret_msg, mcast_grp, mem_num);
14                 elsf
15                         do comment : This cane should not occur
16                             nack(EXP, ret_msg, mcast_grp, mem_num);
17                             adjust_rate_on_error(mcast_grp);
18                             update(LAST_MSG, ret_msg, mcast_grp);
19                             return
20                         od
21                 fi
22             break;
23             case: (mcast_ack); comment : Not needed for receive
```

Table 5.2: Definitions of additional functions used in reliable flow control protocols.

old_msg	True if the packet has been seen before.
reset_ack_timer	Resets the ack timer to expire in the future.
send_to_one	Point-to-point retransmission of the indicated packet.

```
                        if (new(ret_msg, mcast_grp))
                          then enough_acks+ = 1;
                        fi
                  break;
                  case: (mcast_nack);
                        if (new(ret_msg, mcast_grp))
                          do
                              update(MISSED, ret_msg, mcast_grp);
                              adjust_rate_on_error(mcast_grp);
                          od
                        fi
                  break;
          endswitch;
    od
```

The subroutines defined in table 5.1 are also used in the reliable flow control protocols. A few additional subroutines are needed. These are defined in table 5.2.

5.2.3 Calibration

These tests were performed in the same systems and network as the tests of the unreliable flow control protocol. Therefore the calibration results of section 3.2.4 also apply to these tests..

Figure 5.8 compares the sender's performance of the reliable flow control algorithm in a test where there was 1 sender and 1 receiver to the corresponding calibration test. There was no significant difference between reliable and unreliable flow control when compared to the calibration results for this test.

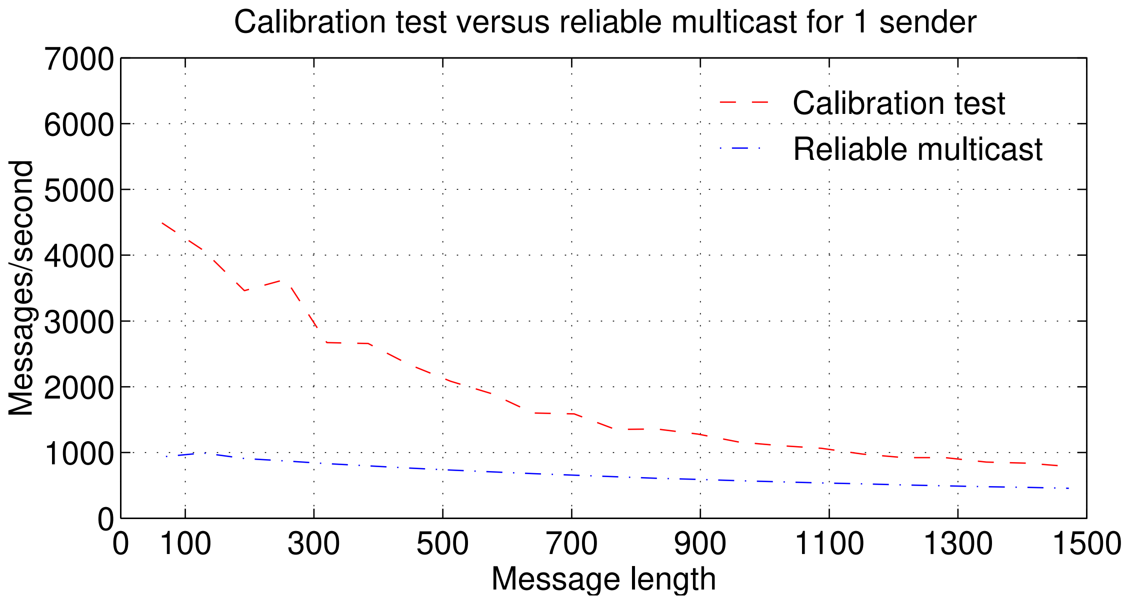

Figure 5.8: Comparison between the sender's performance for reliable multicast and the calibration for tests in which there was 1 sender.

5.2.4 Experimental results

The main advantage reliable flow control has over unreliable is that no messages are lost. The standard environment for unreliable flow control is groups with a single sender and varying numbers of receivers. Figure 5.9 shows the performance of a single sender in a test with 1 to 5 receivers for reliable multicast flow control. This graph is similar to the performance of unreliable flow control as shown in figure 5.4. Unreliable flow control is expected to send more packets than reliable flow control. However sending more packets does not mean that more packets are received by *all* members of the group. Figures 5.4 and 5.10 indicate that both protocols found nearly the same rates. The rates may be similar because these protocols are dominated by their "stop-and-wait" processing of acknowledgments.

The performance of the reliable and unreliable flow control protocols is very simi-

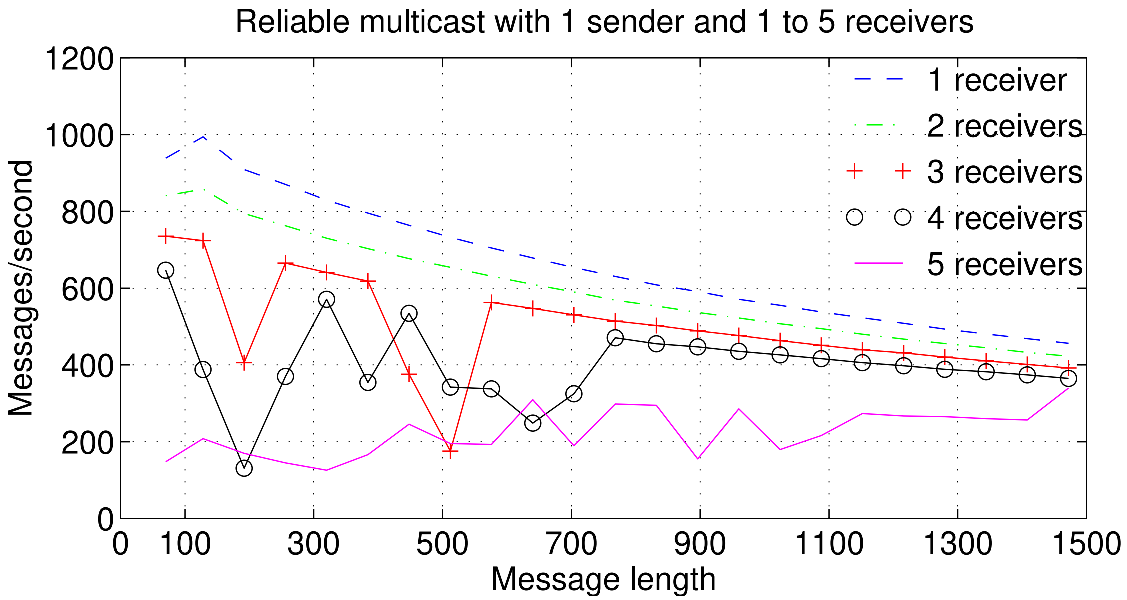

Figure 5.9: Comparison of a single sender's performance for reliable multicast when the number of receivers is varied from 1 to 5.

lar, as shown in figure 5.10, where the performance for a single sender with 1 receiver and a single sender with 4 receivers is compared. For the simplest case, the reliable protocol slightly outperforms the unreliable protocol. However, because these tests were on shared machines and networks and the tests were performed at different times, the difference is probably not significant. For the tests with 1 sender and 4 receivers, it appears that if the reliable protocol had been more stable the performance would have been approximately equivalent. Error correction in the reliable protocol causes the sender's performance to be more erratic.

Figure 5.11 shows the total senders' performance for a series of tests with 1 receiver and from 1 to 3 senders. It is interesting to note that for the test with 2 senders the total performance improved as it did for unreliable multicast. However, total performance in the test with 3 senders varies quite a bit and seems to have generally

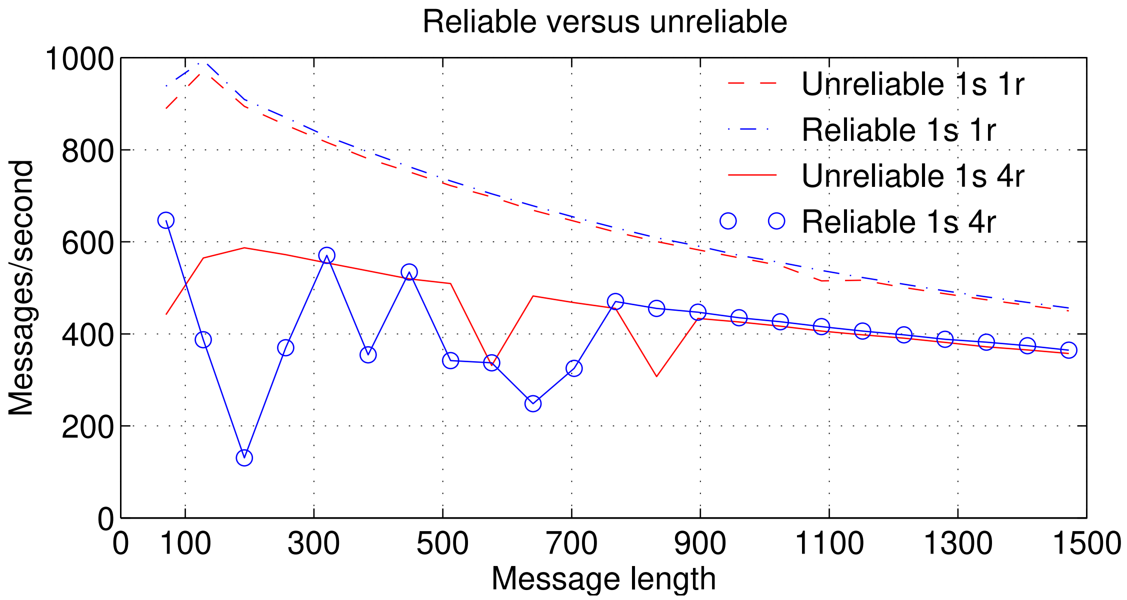

Figure 5.10: Comparison between the sender's performance for unreliable and reliable multicast flow control for tests in which there were 1 sender and 1 and 4 receivers.

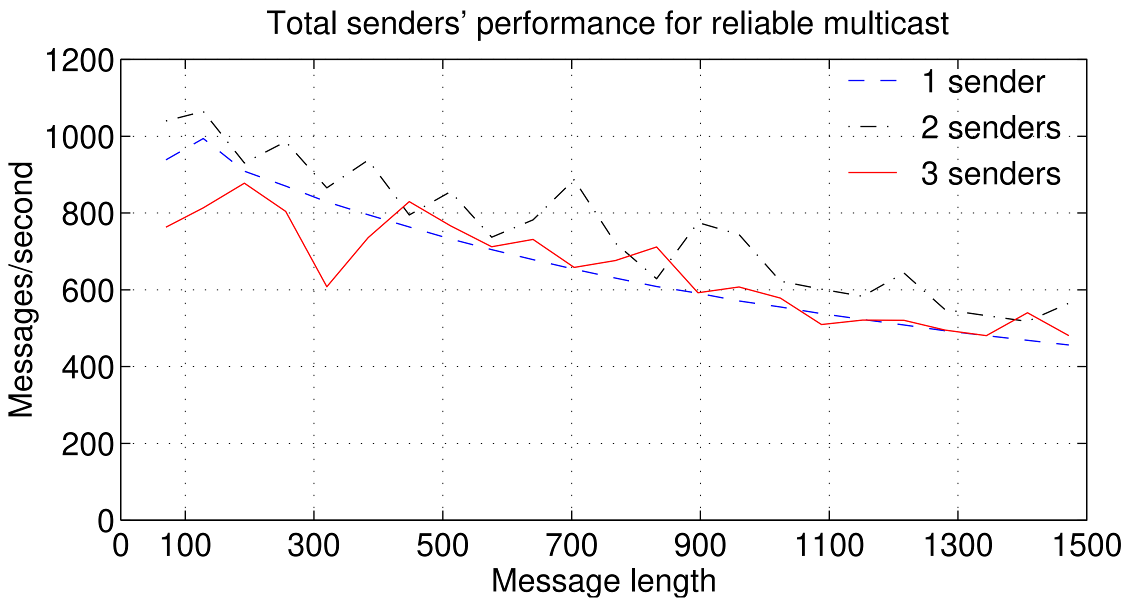

Figure 5.11: Comparison of multiple senders' total performance for reliable multicast for tests with a single receiver and 1 to 3 senders.

degraded. The unreliable protocol was unable to complete this test. The test system failed during a test with 4 senders and 1 receiver for the reliable protocol in the same way as the test system failed for the unreliable protocol during the 3 sender 1 receiver test. This indicates that the reliable protocol benefits from the pseudosynchronization points for multiple sender groups and that rate resynchronization may be more important than was previously thought. As before, the failure of the 4 sender test is attributed to the fixed timeout values.

5.2.5 Conclusions

These results show that send-rate control is also an effective technique for reliable multicast flow control in the environments where it was tested. As expected, as the size of the multicast set grows, the performance of the positive acknowledgment protocol degrades. The fact that the unreliable protocol did not significantly outperform the reliable protocol indicates that these protocols are dominated by the stop-and-wait attribute. The reliable protocols seemed to be more stable than the unreliable protocols; however, it is still the case that scaling was limited by the use of static timer values. As in the case of the unreliable protocol, the reliable protocol would benefit from the implementation of a pseudowindow and the associated buffers.[7] Even though the reliable protocol was not fully generalized, these results have shown that send-rate control can be applied to reliable flow control.

[7] A pseudowindow would wait for an acknowledgment on every $n^{\underline{th}}$ message instead of every message. Adding a window requires buffering packets until all acknowledgments have been received.

5.3 Disseminating Information to Large Groups

Internet radio, Internet TV, net news, and stock quoting systems are examples of applications that disseminate information to large groups of users. These applications typically have a single source and a large number of destinations. The question that naturally arises is, How fast can the protocols that have been presented disseminate information to a large group of users? As was shown in sections 5.1 and 5.2, the performance of positive acknowledgment protocols degrades rapidly as the group size increases even in the single sender case. Therefore additional techniques will be investigated to see if the performance of cascading is better than the performance of a large single group.

Instead of building up a single large group this investigation looks at applying cascaded overlapped groups to this problem. Figure 5.12 illustrates the technique that will be applied. Each group will have one sender and multiple receivers. The group structure will form a tree. The receivers in a group can either be pure receivers or forwarders. Pure receivers are the leaves of the group tree and forwarders are the internal nodes of the group tree. Forwarders at level i receive messages in one group and multicast those (forward) messages to a group at level $i + 1$. Forwarders and pure receivers are assumed to be targets for the disseminated messages. Given this approach, the basic question is, How well does this technique perform when used with the reliable multicast protocol to scale to larger groups? This implicitly asks, Is the performance better than the performance of a single group with the same number of receivers?

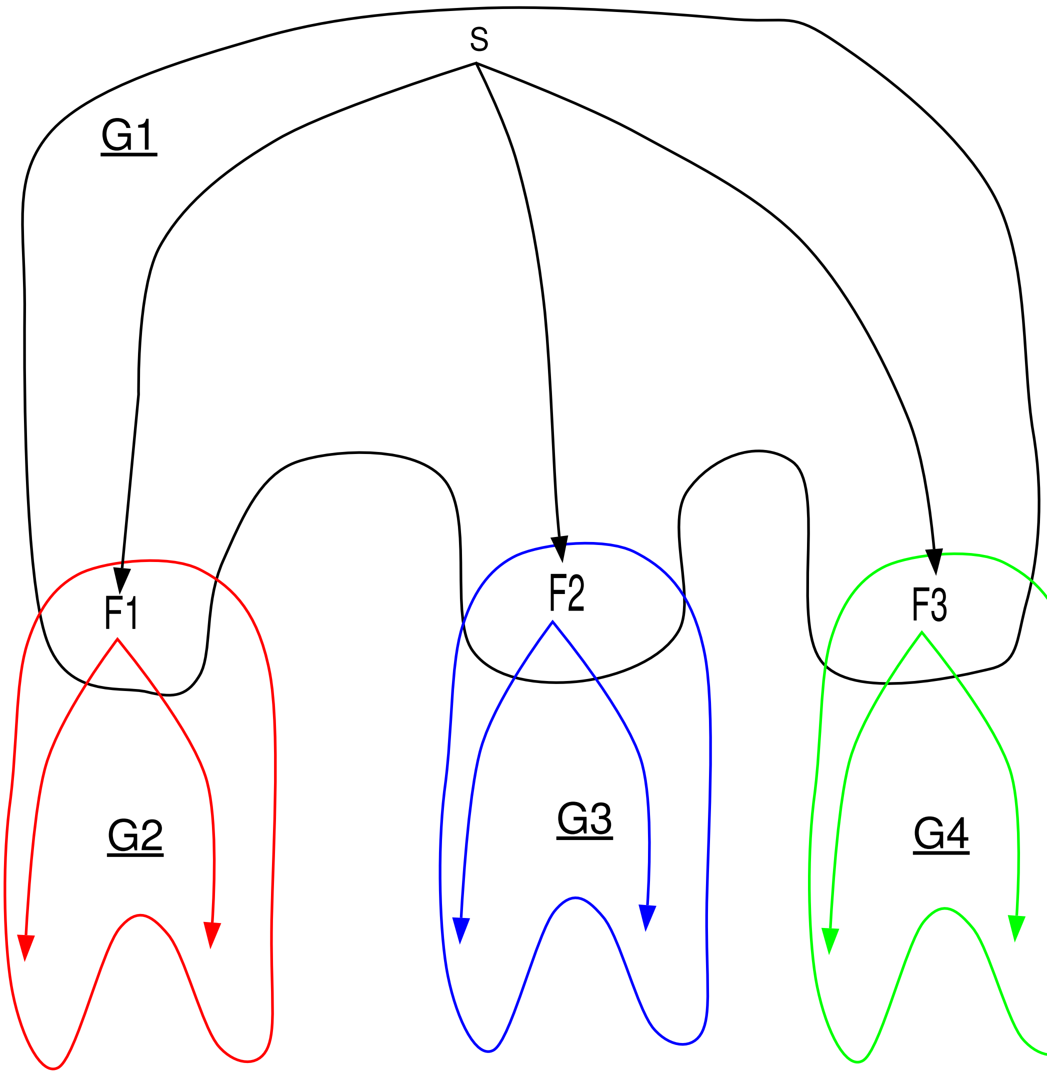

Figure 5.12: Overlapped cascaded group structure for message dissemination. There are four groups G1, G2, G3, and G4. G1 is at level 0 and the others are at level 1. F1, F2, and F3 are receivers which forward from G1 to their respective groups at the next level.

5.3.1 Methodology

Because the reliable and unreliable multicast protocols had similar performance, this study proceeded using the reliable protocol for message dissemination.[8] If the unreliable protocol were optimized, it should perform better than the reliable protocol, so this study demonstrates a minimum expected level of performance.

The test system was changed by adding a forwarder, allowing for multiple concurrent multicast sets, and adding an interface to IP multicast. While the core of the forwarder was inherently simple, this addition caused a restructuring of the test system, because the original system was built with the implicit assumption that the investigation would focus on issues related to a single group with multiple senders and receivers. In addition, the tree structure of overlapped groups was created and initialized top down and started bottom up so that overhead imposed by controlling the test would not interfere with the results. The test protocol specified in appendix A was modified to account for the forwarder and the tree structure.

The forwarder added for these tests was synchronous. It did not send and receive packets concurrently. Instead, it started by receiving a message, and that message had to be successfully forwarded before the next message would be received.

Interfacing the revised test system to IP multicast increased the flexibility of testing, because more machines supported IP multicast. Using IP multicast also allowed quicker development of the tests for overlapped groups. However, IP multicast is fundamentally different from the device driver interface that was used to initially develop and evaluate the protocols.

[8] Therefore this study looks at reliable message dissemination.

To access IP multicast these tests used UDP, which is a higher level unreliable interface that does not supply back pressure to senders, whereas the device driver interface used in earlier tests provides back pressure by informing a user whenever it is not able to send a packet.[9] Therefore send omissions did occur with the UDP interface to IP multicast, whereas they did not occur with the device driver interface. The consequences of this change in the character of the interface are described in more detail in section 5.3.3. Nevertheless, since IP multicast is becoming a standard interface to multicast functionality, using this interface is consistent with testing at the application level, and therefore the results will more accurately represent the performance that could be achieved by applications using these protocols.

5.3.2 Protocol

The reliable protocol that was presented in section 5.2 was used for the cascaded testing. The restructuring that allowed multiple concurrent groups only affected the data structures used by the protocol.

5.3.3 Calibration

Several different things have to be calibrated. The test system has moved from IBM and HP systems to Sun systems. The underlying network is still a 10 Mbit/second ethernet. However the multicast interface has been moved from a device driver interface to UDP and IP multicast. The principal difference is that the device driver interface was reliable and supplied back pressure information to users, whereas UDP

[9] There can be two reasons why the device driver fails to send packets: either there are no kernel buffers available or the device is busy.

is unreliable and does not supply back pressure information to users. Therefore what is actually being calibrated is the UDP interface to IP multicast on Sun systems.

There were more machines which supported IP multicast and therefore could be used for testing. However, the machines which support IP multicast have heterogeneous configurations, whereas the machines that were used for the previous tests had homogeneous configurations. Since it was necessary to run the tests on heterogeneous configurations, an attempt was made to minimize the number of distinct configurations used in each test. In this environment the calibration tests were run on Sparc 10s and 20s.

In order to compare the results for this section to results from chapter 3 and section 5.2 it is necessary to calibrate the machines and interface with the same test that was used in the previous calibrations. This was done by porting the test that was run in chapter 3 to the UDP interface to IP multicast.[10] Figure 5.13 shows the results of running the calibration test with one sender and one receiver. The same data reduction analysis was done on these data as on the calibration data for previous tests.

Recall that the UDP IP interface to IP multicast is fundamentally different from the device driver interface that was used for the earlier calibrations. The calibration test was written assuming that the back pressure information was accurate. The reduction and analysis assume that the sender is accurately reporting the number of messages sent. However, for this interface these assumptions are invalid. In particular, the sender is not sending as much data as reported and indicated in figure 5.13. The

[10] This is the same test that was used in section 5.2.

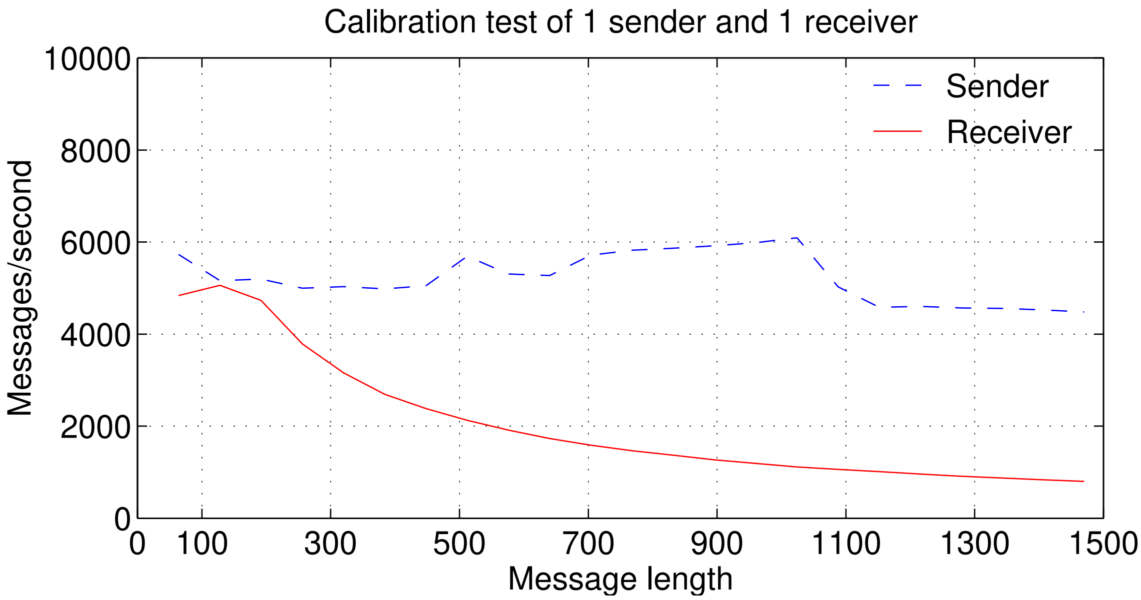

Figure 5.13: Initial calibration test for IP multicast interface with 1 sender and 1 receiver, both Sparc 20s.

inaccuracy of the sender is illustrated in figure 5.14, which shows the same data as figure 5.13 but from the perspective of bytes per second as compared to the limit for a 10Mbit/second ethernet. It is not possible for the sender to send substantially more data than the capacity of the underlying network.[11] Therefore send omissions are occuring, and as a result the interface is more bandwidth limited than message rate limited.

It is worthwhile to review the structure of the multicast interface that is being used. Figure 5.15 shows the structure of this interface. The actual multicasting is being done by Deering's IP multicast extensions to SunOS[Dee89]. Each packet is passed from the application to UDP, then to IP, then to the device driver. UDP always reports that every packet has been sent. However, this is not true. The device

[11] All measurements were also taken from the application level using the time of day clock.

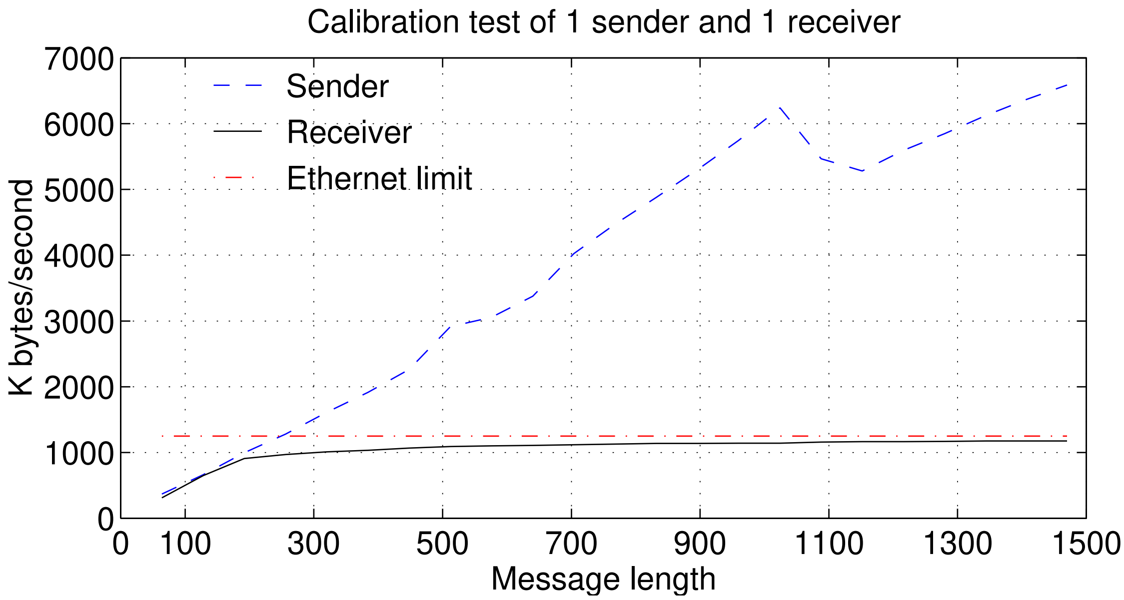

Figure 5.14: Sparc 20 calibration with 1 sender and 1 receiver.

driver supplies back pressure information to IP; however, UDP ignores any back pressure information that it receives. Neither UDP nor IP guarantee that packets are sent. If either receives more packets than can be sent, some of the packets are discarded. When packets are discarded a send omission (or failure to send) occurs. It is reasonable to assume that the corresponding problem could exist on the receive side. Nevertheless, the objective of this work is to investigate rate-based flow control protocols for multicast, so an extensive analysis of the design or structure weaknesses of the underlying system was not done.

Figures 5.16 and 5.17 show what happens to the receiver's performance as the number of senders is increased. In figure 5.16 results are shown in terms of K bytes/second and in figure 5.17 the same results are shown in terms of messages/second. The senders in these tests are all sparc 20s and the receiver is a sparc 10. As the number

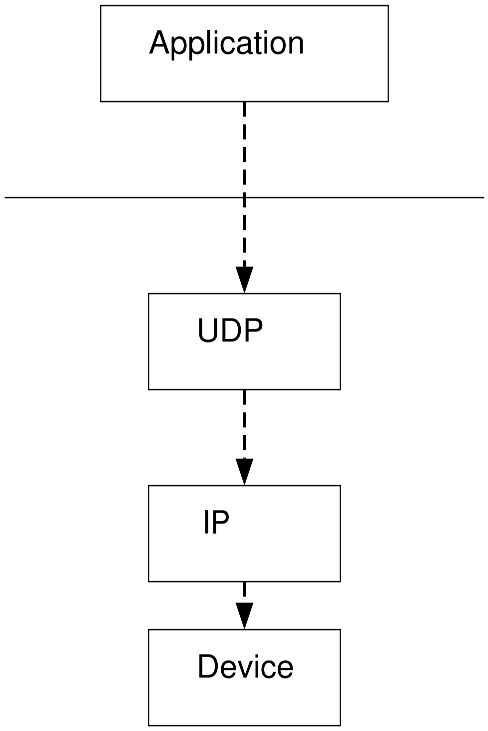

Figure 5.15: Structure of the interface to IP multicast.

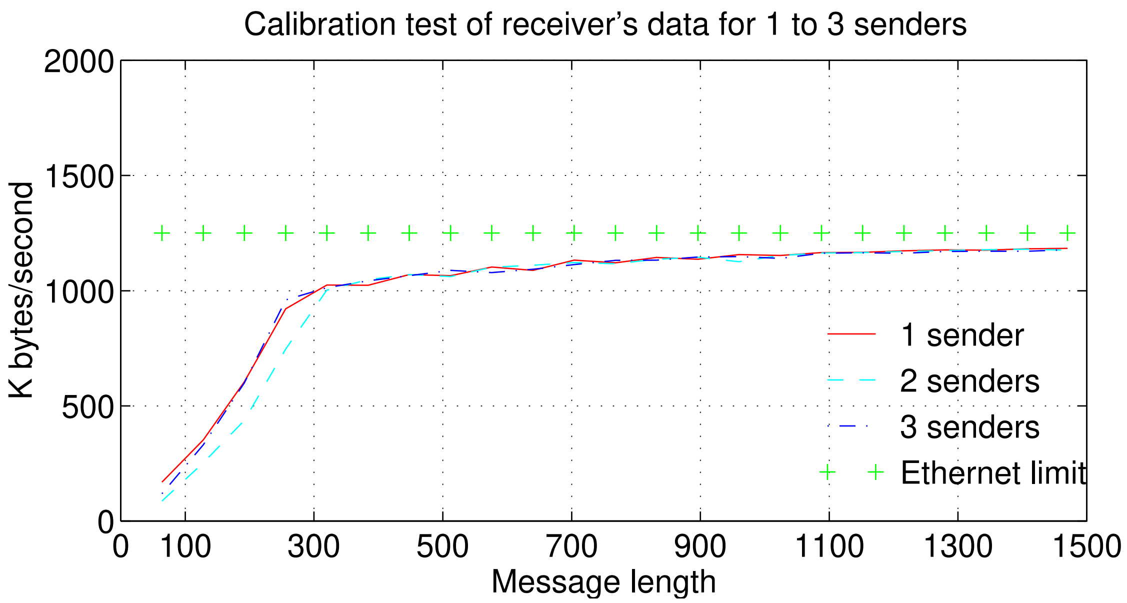

Figure 5.16: Data rate calibration run with 1 to 3 senders and 1 receiver. The senders are all sparc 20s and the receiver is a sparc 10.

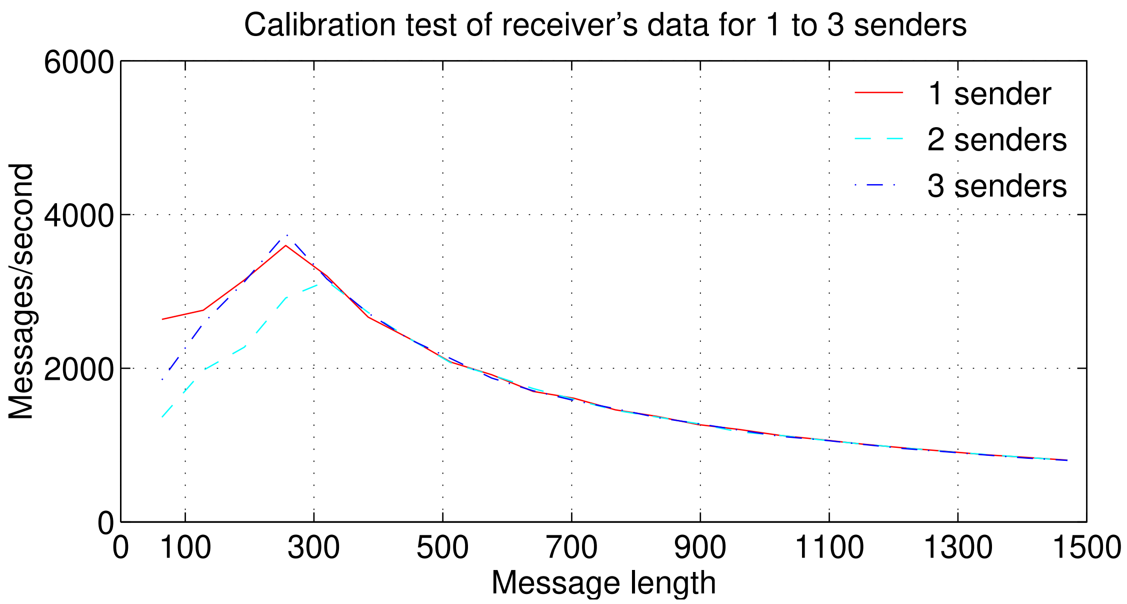

Figure 5.17: Message rate calibration run with 1 to 3 senders and 1 receiver. The senders are all sparc 20s and the receiver is a sparc 10.

of senders increases, the receiver's performance drops off and then improves to the limit. Unlike in previous tests (see figure 5.1), adding additional senders did not improve the receiver's performance for shorter messages.[12] The lack of improvement can partially be attributed to send omissions. If send omissions were not occuring, then adding enough senders would saturate the network even for small message lengths and performance would be driven by the receiver's rate limit. Understanding fully what is occuring in figure 5.16 is complicated by the fact that send and receive omissions may be occuring.

In this environment the performance of the protocol will be more limited by the sender's bandwidth limitation than by the receiver's receive limit. Further, since a corresponding problem probably exists in the receiver, if the senders were able to saturate the network with small messages, the receivers could be limited by additional receive omissions above the device driver level. It is interesting to note that the lack of back pressure from the sending interface makes it difficult for the sender to obtain maximum capacity and therefore affects the overall performance of the application.

5.3.4 Experimental Results

Even though the calibration test found a substantially different interface than was used for the other test, it is worthwhile to note that the protocol does not need revision. No revision is necessary because the response to send omissions should be the same as the response to receive omissions: reduce the rate. Since send omissions occur

[12]Message lengths in the performance graphs do not include length of the protocol header, UDP header, or IP header. Therefore, performance in absolute terms was actually much closer to the limit for 10 Mbit/second ethernet.

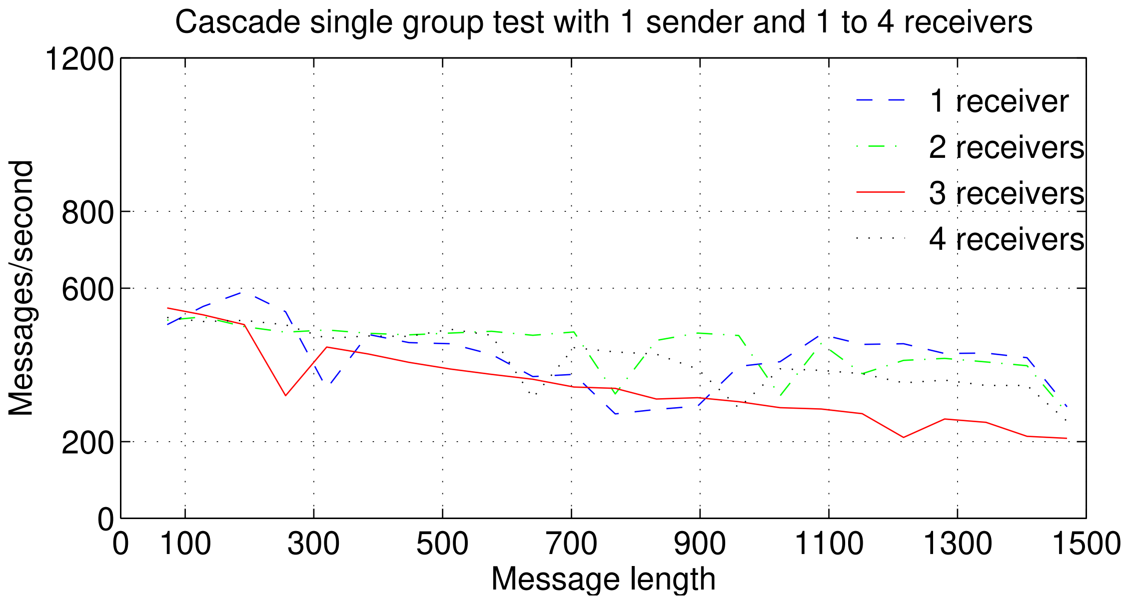

Figure 5.18: Receivers' performance for message dissemination to a single group with 1 sender and 1 to 4 receivers. The sender for the 1 and 2 receiver test is a sparc 10 and for the 3 and 4 receiver tests is a sparc 20, and the receivers are sparc 10s and 20s.

because the sender requests that packets be transmitted faster than the underlying interface's capacity, reducing the rate at which the sender makes these requests will also correct this problem.

This section examines message dissemination from the perspective of the receiver's performance. However, since the protocol used is synchronous and reliable, the sender's and receiver's rates are essentially identical. These tests were run on shared systems on lightly loaded networks.

Figure 5.18 shows message dissemination performance for a single group as the number of receivers varies from 1 to 4. The performance range of this test is narrower than the performance range for the corresponding test shown in figure 5.9.

Figures 5.19 and 5.20 examine the causes of the narrower performance range a bit

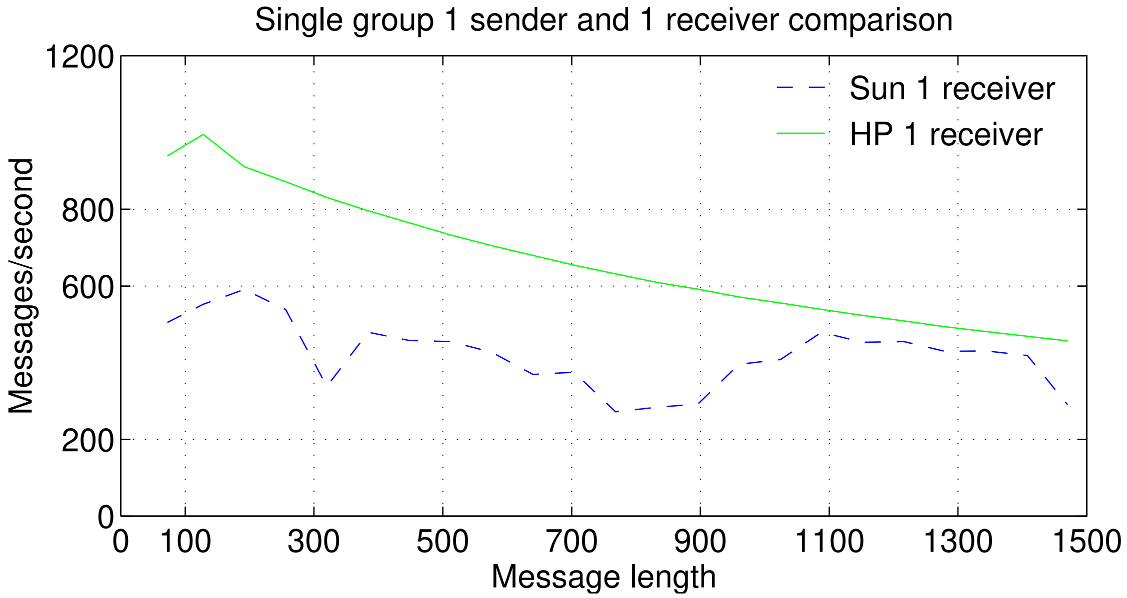

Figure 5.19: Comparison between HP and Sun systems of the receiver's performance in a single group with 1 sender and 1 receiver.

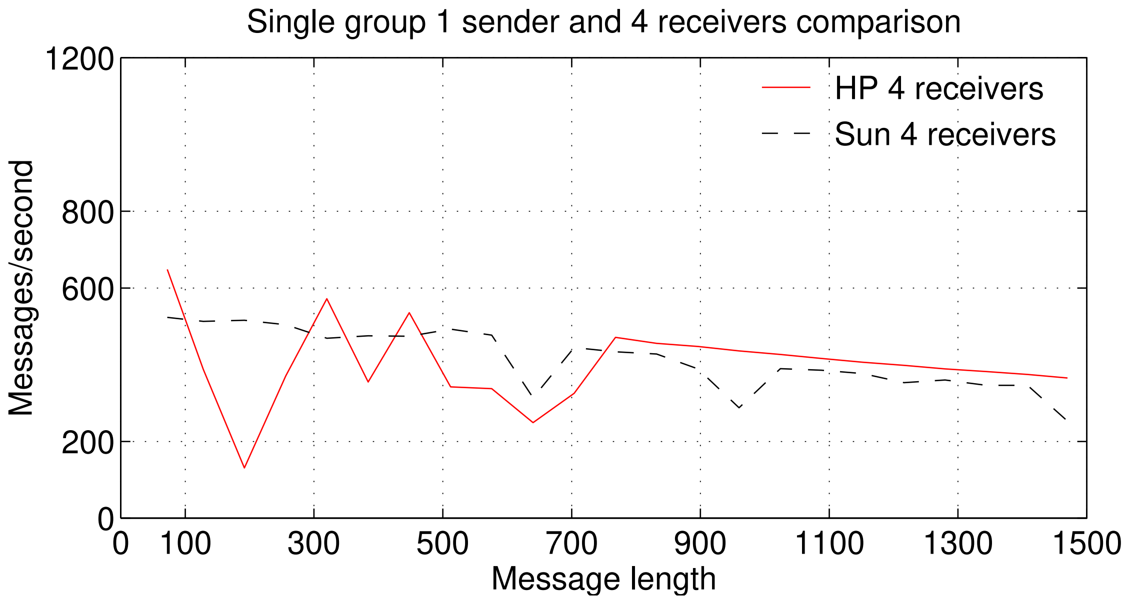

Figure 5.20: Comparison between HP and Sun systems of the receivers' performance in a single group with 1 sender and 4 receivers.

further. In figure 5.19 there is a substantial difference between the receiver's performance in the two 1 sender 1 receiver groups. Because the HP and Sun systems that were used are approximately equivalent, the performance difference can be attributed to the effects of using a higher level interface and the associated send omissions. As the size of the group grows, figure 5.20 indicates that the difference between the two systems becomes inconsequential. The difference becomes less significant because the stop-and-wait (positive acknowledgment) attribute of the protocol is having a greater impact on overall performance.

Positive acknowledgment protocols are also sensitive to latency, although the impact of latency decreases with message size, especially on a LAN. Figure 5.21 shows the impact of increased latency on the protocol for this test environment. The latency for the different segment test is approximately twice the latency for the same segment test. Extra latency is introduced because different segments are connected into a single subnet address by a PowerHub.

In order to examine the basic questions, a cascaded test was done. For this test two member groups were cascaded in a chain. The group at the top of the chain had a sender and a receiver. Whenever two groups overlapped, the point of overlap was a forwarder. The final group in the chain had a forwarder and a receiver. Figure 5.22 shows the performance of message dissemination for a series of tests constructed this way. The performance range for these tests is approximately the same as the performance range for multiple receivers in a single group.

Figure 5.23 shows the steady state message dependencies caused by the combination of a positive acknowledgment protocol and a synchronous forwarder. Messages

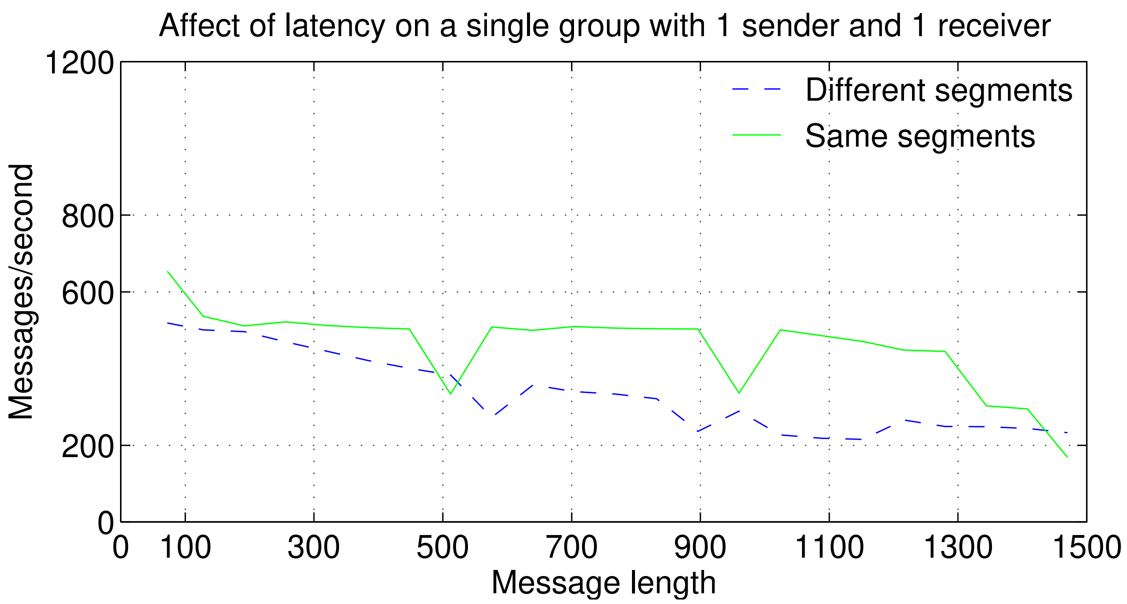

Figure 5.21: Comparison of receiver's performance for single group tests with different latency. The first test is with two sparc 10s on the same physical network segment and the second test is with two sparc 10s on different network segments. In both tests the systems were on the same subnet.

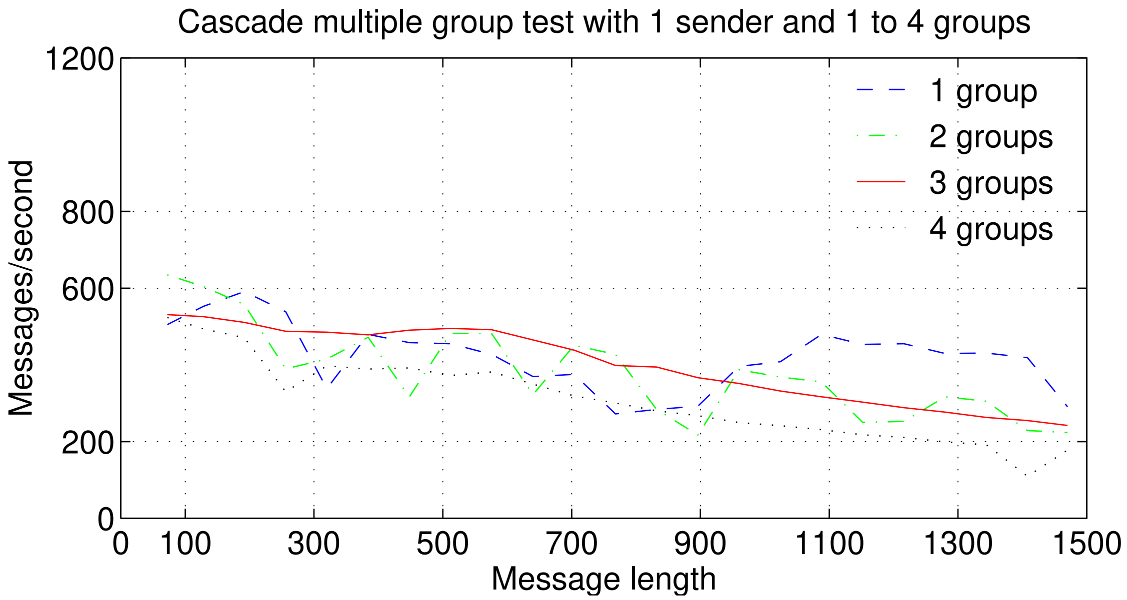

Figure 5.22: Comparison of receiver's performance for cascaded groups where there is 1 sender and 1 forwarder per group. The number of overlapped groups varies from 1 to 4. This test represents performance for a single group chain.

are shown horizontally, and groups are the vertical alignment. Pairs of arcs represent the send and associated acknowledgment for a message. The vertical dashed lines are message-to-message dependencies introduced by the synchronous positive acknowledgment protocol. The group-to-group dashed lines are dependencies introduced by the synchronous forwarder. For example, M2 cannot be received by the forwarder in group 1 until it has received an acknowledgment for M1 which it sent to group 2. Therefore, it is clear from this figure that the send of the fourth message is dependent on the receipt of the third message in group three.

In order to improve performance, at least one of the dependencies illustrated with dashed lines in figure 5.23 must be removed. The vertical dependencies can be removed by making the protocols asynchronous or by introducing pseudowindows,

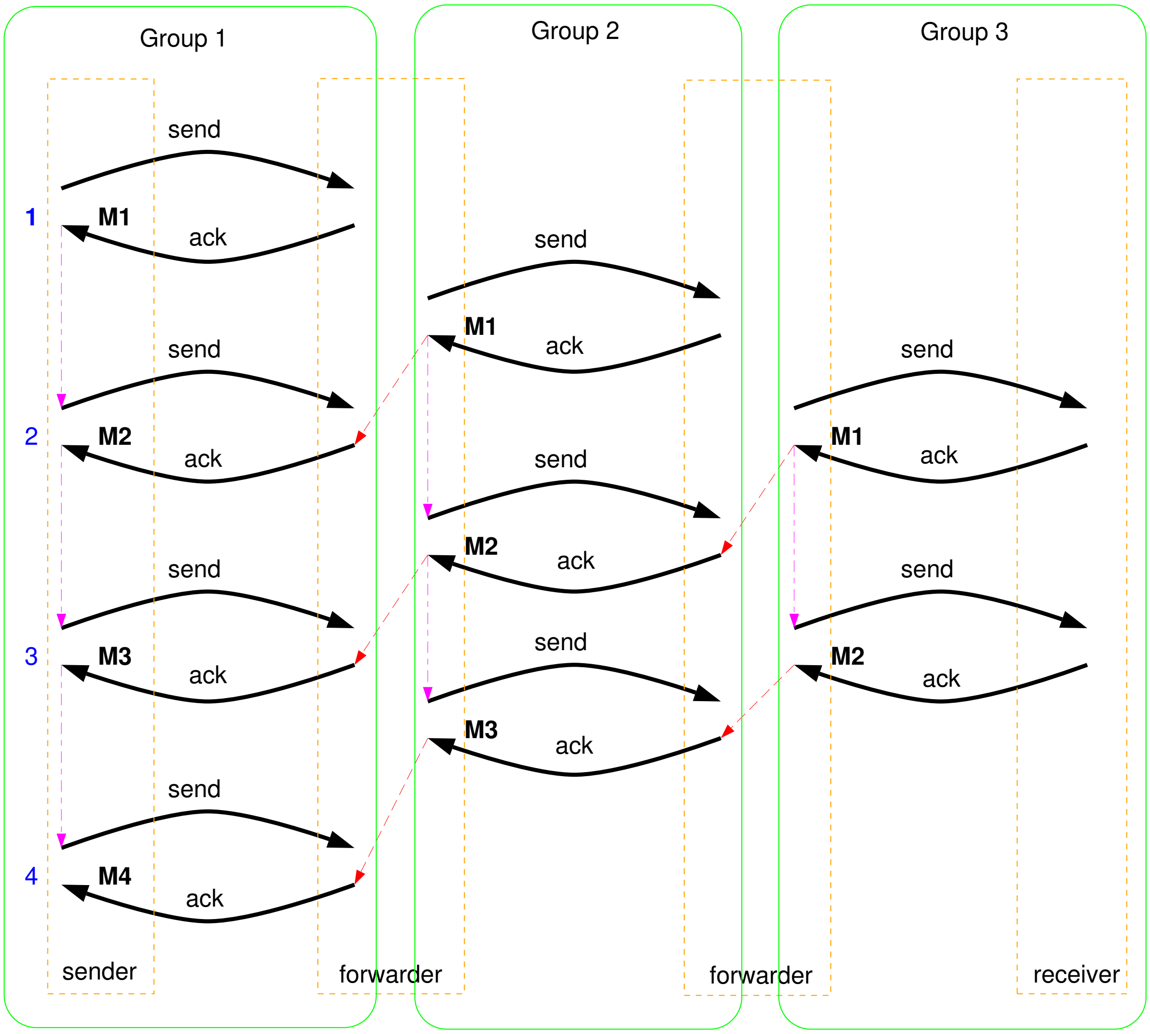

Figure 5.23: Illustration of the dependencies generated by the synchronous protocol combined with the synchronous forwarder. Dashed square boxes represent senders, forwarders, or receivers. Rounded boxes represent the groups. Each group has two members. Vertical dashed arrows indicate the message-to-message dependency caused by the positive acknowledgment protocol. The group-to-group dashed arrows indicate the dependencies caused by the synchronous forwarder.

which would allow a fixed number of messages to be sent before waiting for all acknowledgments. The group-to-group dependencies can be eliminated by dividing the forwarder into two asynchronous parts which run in a producer-consumer relationship with a buffer between them. The first part of an asynchronous forwarder receives messages in one group and then queues these messages to be sent by the second part. The second part would read the queued messages and send them to the lower level group.

5.3.5 Conclusions

In this environment, where packet processing time is significantly less than packet generation and send time, there is no benefit from cascading messages with a synchronous forwarder. The results indicate that to gain a performance benefit from cascading messages the underlying protocol must be enhanced with pseudowindows, the underlying protocol must be asynchronous, or the two parts of the forwarder must be asynchronous.

5.4 Summary

This chapter presented studies of a reliable and an unreliable rate-based multicast flow control protocol based on the generic design presented in chapter 4 and a study of the application of the reliable protocol to the problem of message dissemination. The pseudocode for implementing unreliable and reliable protocols was also presented. The implementations that were studied were basic, without fully generalized timers or any asynchronous behavior. Nevertheless, the send-rate control approach to multicast flow control was shown to work in the limited environments where it was tested. The reliable protocol was applied to the problem of disseminating messages. The message disseminating implementation that was tested used a synchronous forwarder. The goal was to investigate whether performance would be better with cascaded groups than with a single large group. Even though cascading has potential, performance was not improved over that of a single group because of the intermessage dependencies introduced by the combination of a synchronous forwarder with the synchronous positive acknowledgment protocol. Based on the observed dependencies, suggestions for improving the performance were made.

Chapter 6

Conclusion

This dissertation presents a study of the performance of multicast on a single LAN. Multicast performance was studied on ethernet and token-ring LANs. In addition, the performance of point-to-point emulation of multicast was also studied. These studies indicated that in an environment where senders were not otherwise limited, multicast performance was limited by the receiver's ability to receive messages, independent of the message length. This was true even when multicast was emulated with point-to-point messages. These two observations imply that in the absence of other factors communication between systems is rate limited independent of the type of communication: point-to-point, broadcast, or multicast.

A definition of flow control was presented which functionally divides flow control into three components: supply control, error detection, and error correction. Given this understanding, rate control was differentiated from rate reservation as a technique for supply control. A definition of multicast flow control was presented which included a definition of the multicast network environment and some assumptions about the attributes of that environment. A review of the design issues associated with multicast flow control was presented, followed by a presentation of a send-rate control approach to multicast flow control for multiple sender groups which depends on the previously defined environment.

Simplified unreliable and reliable protocols were implemented based on the proposed approach. A performance study of these protocols shows that send-rate control has merit. However, the protocols as studied were limited by unsophisticated timers and their positive acknowledgment structure. In addition, because of positive acknowledgments, the performance of the protocols degrades rapidly as the group size increases. It was proposed that performance of the simplified protocols could be improved by adding dynamic timers and pseudowindows with the associated buffers where required.

The simplified reliable protocol was applied to the problem of reliably disseminating messages to a large number of processors. The protocol was not changed to take advantage of the single sender/group nature of message dissemination. The dissemination system utilized a synchronous forwarder. It was shown that because of the synchronous nature of the forwarder, combined with the synchronous reliable protocol, no performance benefit was gained by cascading groups over a single large group. These results suggested that in order to improve performance using cascading, either the forwarder must have two asynchronous parts (a receiver and a sender) or the underlying protocol must allow for more asynchrony.

This work also suggest that back pressure is a useful component for optimally utilizing the capacity of a communications interface.

Multicast flow control is a complicated problem. This dissertation has presented a study of existing systems, a discussion of the issues, a protocol approach, and a study demonstrating that the protocol approach works in the environments where it was tested. The proposed approach is a general purpose flow control protocol which has a limited number of assumptions about the surrounding environment. In particular, the protocol makes no assumptions about the number of senders per multicast set. The viability of the approach was the focus of this dissertation. Therefore, scaling was a secondary issue. Since scaling is an important issue for flow control protocols, suggestions for more robust implementations have been made.

Appendix A

Details of Test Protocol

Figure A.1 shows the details of the test protocol used by the initial test system. The initiator gets control information from a script file which tells it how many senders and receivers are participating in the test as well as how many messages of each length to transmit. A single test can have multiple subsections.

Receivers in each test start waiting for messages as soon as they receive the *TEST_SET*. Their total test time is calculated from the receipt of the first test message to the receipt of the last test message.

The test consists of the sender(s) sending messages to the receivers. When the sender(s) have completed sending all the required messages, they send *TEST_END* to the initiator. Each sender calculates its test time from sending its first message to sending its last message.

The initiator then checks the state of all participating processes. This also terminates the test for the receivers. (This message is not a test message.) If all processes

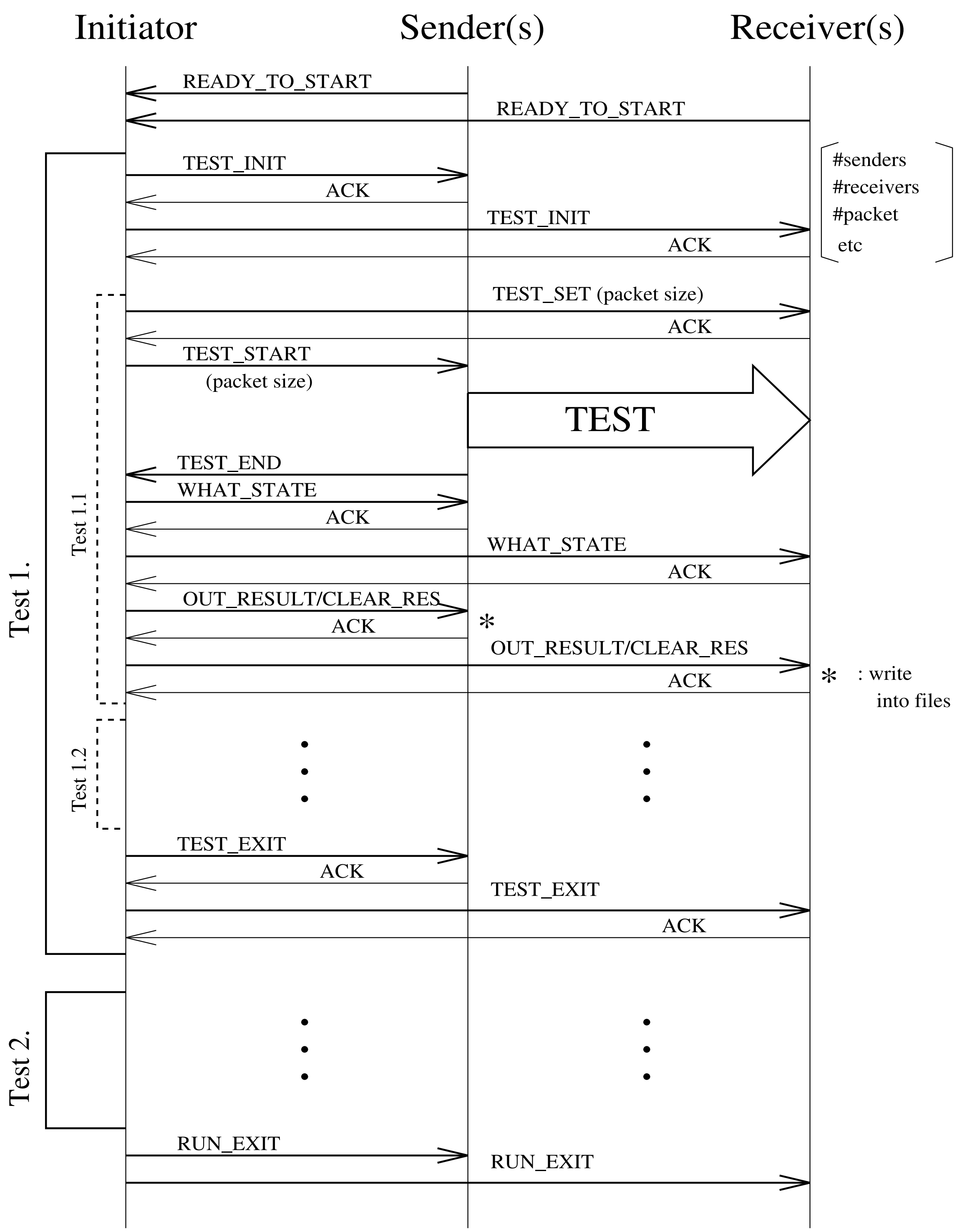

Figure A.1: Protocol used by the initial test system.

are in the correct state, then they are told to record their test information and the next test cycle is started. If at least one process fails the test or is in the wrong state, then the test cycle is rerun. If the initiator loses contact with a process (no reply is received to the *WHAT_STATE* query), the initiator aborts the remainder of the test.

Appendix B

Comparison to Horus

In order to measure whether or not the proposed approach to flow control has merit, a comparison was made with the Horus system[vRHB94]. Horus was used because of the availability of data and because it had recently been found to be the best performing system which includes multicast and group communication[Orn94]. However, the comparison is not fair because the data and test methodology were not the same for both systems.

For Horus we have one way latency data and a test interval of 512 bytes. This latency data was used to compute the message rate performance. For the protocols proposed in this dissertation we have the measured message rate as seen by a receiver and a test interval of 64 bytes. Both sets of numbers are for multicast flow control over IP multicast. However, for Horus the multicasts are virtually synchronous and for the protocols proposed in this dissertation the multicasts are actually synchronous. In addition, Horus is a fully functioning group communication system, whereas the

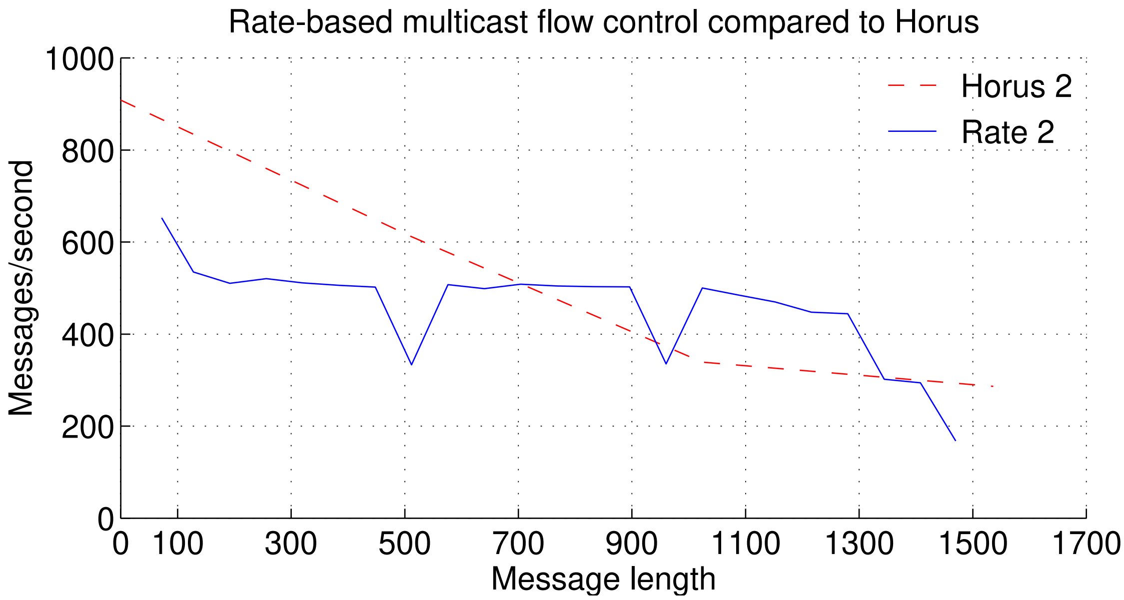

Figure B.1: Comparison between send-rate control and Horus when the group size is 2.

system used to measure the performance of the proposed protocols is lightweight and designed to test the viability of the protocol approach. Also the proposed protocols were not optimized for performance in any sense. Nevertheless the comparison is useful and shows that the proposed protocols have merit.

Figures B.1, B.2, and B.3 compare the message rate of Horus to the message rate of the protocols proposed in this dissertation for groups of size 2, 3, and 4. For figures B.1 and B.2 the tests were run on Sparc 10s. For figure B.3 the Horus test was run on Sparc 10s, but for the send-rate test the sender was a Sparc 20 and the receivers were Sparc 10s. (Since the synchronous protocol is receiver limited, the data are reasonably comparable.) In this rough comparison the protocols proposed in this dissertation seem to scale better and as a result perform better than Horus. However, since this is not a direct fair comparison, the only conclusion that can reasonably be

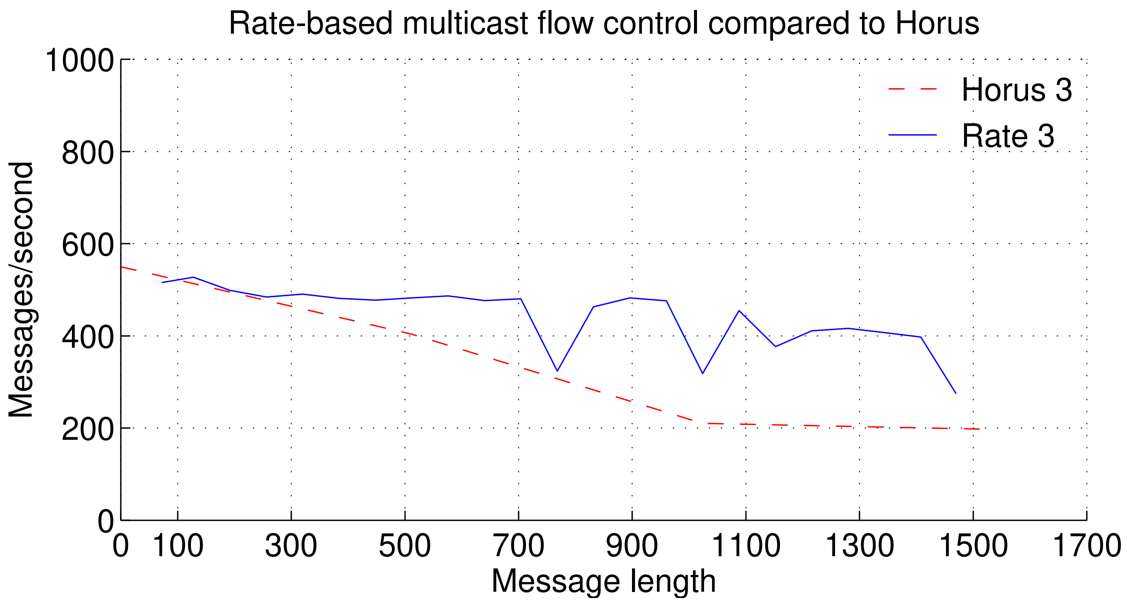

Figure B.2: Comparison between send-rate control and Horus when the group size is 3.

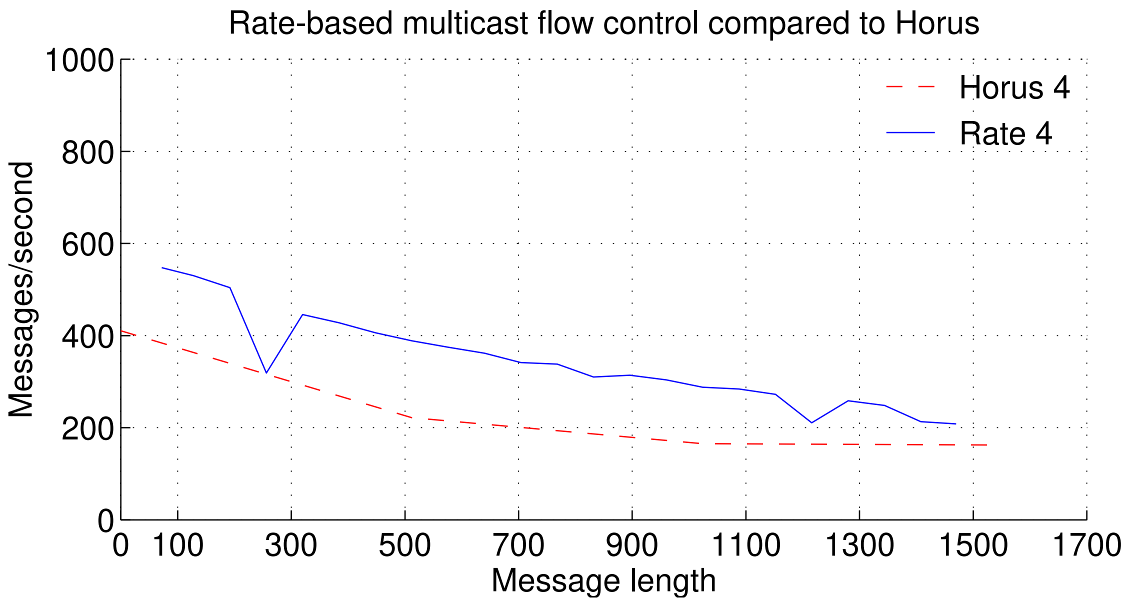

Figure B.3: Comparison between send-rate control and Horus when the group size is 4.

presumed is that the protocols are comparable in speed.

The credit-based flow control mechanism used in Horus is a higher level protocol than the send-rate control protocols proposed here. To do a fair comparison either the evaluation test used in this dissertation would have to be implemented over Horus or the protocols proposed in this dissertation would have to be incorporated into Horus. If either of these were done, a better comparison of the relative performance of the two protocol approaches would result.

Appendix C

Timers

As has been pointed out in this dissertation, optimizing timers is a complex problem that was not addressed. There are three timers in the proposed protocol: the increment timer, decrement timer, and acknowledgment timer. In order to avoid the problem of optimizing these timers, they were set to fixed values. The timers and timer-related values used for the test reported are shown in table C.1. These values were arrived at after a brief period of experimentation. They are not presumed to be optimal.

As a rule of thumb, the acknowledgment timeout should be less than the increment and decrement timeouts. However, the size of the increment and decrement timeouts will determine the protocol's "responsiveness" to changes in network load. As is pointed out in section 5.1.5, timer values should probably depend on the size of the multicast set, intermessage spacing, and maximum round-trip time to a member of the multicast set.

Table C.1: Time related values.

Field Name	Value
Initial Intermessage spacing	2000 microseconds
Microseconds per unit	40
Increment timeout	500 units
Decrement timeout	500 units
Acknowledgment timeout	400 units

Bibliography

[ADKM92] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. Transis: A communication sub-system for high availability. In *FTCS-22 The Twenty-Second International Symposium on Fault-Tolerant Computing*, pages 76–84. IEEE Computer Society Technical Committee on Fault-Tolerant Computing, Institute of Electrical and Electronic Engineers, July 1992.

[AFM92] Susan M. Armstrong, Alan O. Freier, and Keith A. Marzullo. Multicast transport protocol. RFC 1302, Network Information Center, February 1992. Network Working Group RFC 1301, new proposal.

[Bac86] Maurice J. Bach. *The Design of the UNIX Operating System*. Prentice Hall, 1986.

[BC94] Kenneth P. Birman and Timothy Clark. Performance of the Isis Distributed Computing Toolkit. Technical Report TR94-1432, Cornell University, Dept. of Computer Science, June 1994.

[Bir93] Kenneth P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12), December 1993.

[BvR94] Kenneth P. Birman and Robbert van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, 1994.

[CNL89] S. T. Chanson, G. W. Neufeld, and L. Liang. A bibliography on multicast and group communications. *Operating Systems Review*, 23(4):20–25, October 1989.

[Com91] Douglas E. Comer. *Internetworking with TCP/IP Volume 1, Principles, Protocols, and Architecture*. Prentice Hall, 1991.

[CP88] J. Crowcroft and K. Paliwoda. A multicast transport protocol. *Computer Communication Review*, 18(4):247–256, August 1988. SIGCOMM '88 Symposium: Communications Architectures and Protocols.

[CS91] Douglas E. Comer and David L. Stevens. *Internetworking with TCP/IP Volume 2, Design, Implementation, and Internals*. Prentice Hall, 1991.

[CS93] Douglas E. Comer and David L. Stevens. *Internetworking with TCP/IP Volume 3, Client-Server Programming and Applications*. Prentice Hall, 1993.

[DC90] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[Dee88] Stephen E. Deering. Multicast routing in internetworks and extended lans. *Computer Communication Review*, 18(4):55–64, August 1988. SIGCOMM '88 Symposium Communication Architecture and Protocols.

[Dee89] Steve Deering. Host extensions for IP multicasting. RFC 1112, Network Information Center, August 1989. Network Working Group RFC 1112, obsoletes: RFCs 988, 1054.

[DEK+82] P. Decitre, J. Estublier, A. Khider, X. Rousset De Pina, and I. Vatton. An efficient error detection mechanism for a multicast transport service on the DANUBE network. In Piercarlo Ravasio, Greg Hopkins, and Najah Naffah, editors, *Local Computer Networks*, pages 335–347. North-Holland, April 1982. Proceedings of the IFIP TC 6 International In-Depth Symposium on Local Computer Networks.

[DFW90] Bert J. Dempsey, John C. Fenton, and Alfred C. Weaver. The multidriver: A reliable multicast service using the Xpress Transfer Protocol. In *Proceedings: 15th Conference on Local Computer Networks*, pages 351–358. IEEE, Computer Society, 1990.

[FM90] Alan O. Freier and Keith Marzullo. MTP: An atomic multicast transport protocol, July 1990.

[FWB85] Ariel J. Frank, Larry D. Wittie, and Arthur J. Bernstein. Multicast communication on network computers. *IEEE Software*, May 1985.

[GJ84] Inder S. Gopal and Jeffrey M. Jaffe. Point-to-multipoint communication over broadcast links. *IEEE Transactions on Communications*, 32(9):1034–1044, September 1984.

[Int90] International Business Machines Corp. *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference Options and Devices*, 1990. SA23-2646-00.

[Int91a] International Business Machines Corp. *IBM AIX Version 3 for RISC System/6000 Communications Programming Concepts*, 1990, 1991. SC23-2206-01.

[Int91b] International Business Machines Corp. *IBM AIX Version 3 for RISC System/6000. Vol. 1, Communication Concepts and Procedures*, 1990, 1991. GC23-2203-01.

[Int91c] International Business Machines Corp. *IBM AIX Version 3 for RISC System/6000. Vol. 2, Calls and Subroutines Reference: Base Operating System*, 1991. SN32-9022-00.

[Jac88] Van Jacobson. Congestion avoidance and control. *Computer Communication Review*, 18(4):314–329, August 1988. SIGCOMM '88 Symposium Communication Architecture and Protocols.

[Jai92] Raj Jain. Myths about congestion management in high-speed networks. In *Information Network and Data Communication, IV*, pages 55–70. North-Holland, March 1992.

[JQ91] K. Jakobs and U. Quernheim. Multicast communications in networks with arbitary topology. In *Third IEEE Conference on Telecommunications*, pages 245–250. IEEE, March 1991.

[JSW91] Mark G. W. Jones, Søren-Aksel Sørensen, and Steve R. Wilbur. Protocol design for large group multicasting: the message distribution protocol. *Computer Communications*, 14(5):287–297, June 1991.

[KT91a] M. Frans Kaashoek and Andrew S. Tanenbaum. Fault tolerance using group communication. *Operating Systems Review*, 25(2):71–74, April 1991.

[KT91b] M. Frans Kaashoek and Andrew S. Tanenbaum. Group communications in the amoeba distributed operating system. In *11th International Conference on Distributed Computing Systems*, pages 222–230. IEEE, IEEE Computer Society, May 1991.

[Lee91] T.-H. Lee. Improving performance of multidestination ARQ schemes under high error rate conditions. *Electronics Letters*, 27(3):293–294, January 1991.

[Moc83] Paul V. Mockapetris. Analysis of reliable multicast algorithms for local networks. Technical Report ISI/RZ-83-10, University of Southern California Information Sciences Institute, November 1983. NTIS number AD-A135 807/6/HDM.

[Mul93] Sape Mullender, editor. *Distributed Systems.* ACM Press; Addison-Wesley, 2nd edition, 1993.

[Nar90] Ajit P. Naryan. Reliable transfer of data in a local area network with multicast distribution. In *Proceedings. 15th Conference on Local Computer Networks*, pages 310–319. IEEE Computer Society, September 1990.

[Ngo91] L. H. Ngoh. Multicast support for group communications. *Computer Networks and ISDN Systems*, 22(3):165–178, Oct. 1991.

[Orn94] Rimon Orni. A comparative work on the performance of PCODE. Unpublished report, High Availability Lab, Computer Science Department, The Hebrew University of Jerusalem, Jerusalem, Israel, October 1994.

[Pal88] K. Paliwoda. Transactions involving multicast. *Computer Communications*, 11(6):313–318, December 1988.

[Pro92] Protocol Engines Incorporated, editors. *XTP Protocol Definition*, 11 January 1992. Revision 3.6, PEI 92-10.

[PTK94] Sridhar Pingali, Don Towsley, and James F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *Performance Evaluation Review*, pages 221–230, May 1994.

[SE90] R. P. Singh and A. Erramilli. Protocol design and modeling issues in broadband networks. In *IEEE Conference on Communications ICC '90 Including SUPERCOMM Technical Sessions. SUPERCOMM ICC '90*, pages 1458–1463. IEEE, April 1990.

[SG92a] Robert W. Scheifler and James Gettys. *X Window System.* Digital Press, 3rd edition, 1992.

[SG92b] A. P. Seneviratne and A. Ginige. Flow control techniques for multicast communication in distributed multimedia applications. In *Conference on Communication Technology, Services and Systems — Communications '92*, pages 207–211. IE Australia, October 1992.

[Ste90] W. Richard Stevens. *UNIX Network Programming.* Prentice Hall, 1990.

[Tan88] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, 2nd edition, 1988.

[Tuf83] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 1983.

[vRHB94] Robbert van Renesse, Takako M. Hickey, and Kenneth P. Birman. Design and performance of Horus: A lightweight group communications system. Technical Report TR94-1442, Cornell University, Department of Computer Science, August 1994.

[WS88] Jonathan L. Wang and John A. Silvester. Optimal adaptive ARQ protocols for point-to-multipoint communications. In *IEEE INFOCOM '88 - The Conference on Computer Communications. Proceedings of the Seventh Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 704–713. IEEE, March 1988.

[WS89] Jonathan L. Wang and John A. Silvester. Delay minimization of the adaptive go-back-N ARQ protocols for point-to-multipoint communication. In *IEEE INFOCOM '89 - The Conference on Computer Communications. Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 584–593. IEEE, April 1989.

[WS91] Jonathan L. Wang and John A. Silvester. Performance optimization of the go-back-N ARQ protocols over broadcast channels. *Computer Communications*, 14(7):393–404, September 1991.

[Yam90] Y. Yamuchi. Reliable multicast over the mobil packet radio channel. In *40th IEEE Vehicular Technology Conference*, pages 366–71. IEEE, 1990. (Cat. No. 90CH2846-4, Orlando, FL, USA, 6-9 May 1990).