SCALABLE RELIABLE MULTICAST
IN WIDE AREA NETWORKS

A Dissertation Presented

by

SNEHA KUMAR KASERA

SCALABLE RELIABLE MULTICAST

IN WIDE AREA NETWORKS

A Dissertation Presented

by

SNEHA KUMAR KASERA

Approved as to style and content by:

_____

Donald Towsley, Co-chair

_____

James F. Kurose, Co-chair

_____

Krithivasan Ramamritham, Member

_____

Weibo Gong, Member

_____

James F. Kurose, Department Chair
Department of Computer Science

To my wife, my daughter and my parents

ACKNOWLEDGEMENTS

ABSTRACT

SCALABLE RELIABLE MULTICAST
IN WIDE AREA NETWORKS

SEPTEMBER 1999

SNEHA KUMAR KASERA

B.S., CALCUTTA UNIVERSITY

M.E., INDIAN INSTITUTE OF SCIENCE

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Donald Towsley and Professor James F. Kurose

Many applications including one–to–many file transfer, information dissemination (e.g., stock quotes and web cache updates), distance learning, shared whiteboards, multi–party games, and distributed computing can benefit from reliable multicast. The goal of this dissertation is to design and evaluate large scale multicast loss recovery architectures and protocols for IP multicast capable networks, that make efficient use of both the network and end–system resources and scale to applications that can have thousands of receivers spanning wide area networks.

One of the important problems in multicast loss recovery is that of a receiver receiving unwanted retransmissions of packets lost at other receivers. We present a new approach to scoping retransmissions in which a single multicast channel is used for the original transmission of packets and separate multicast channels are used for scoping retransmissions to

"interested" receivers only. We find that a small to moderate number of multicast channels can be recycled to achieve almost perfect retransmission scoping for a wide range of system parameters. We also propose two mechanisms for implementing retransmission channels, one using multiple IP multicast groups and the other using a single IP multicast group in conjunction with additional router support. We show that the second approach reduces both host processing costs and network bandwidth usage.

Another problem arises when the sender alone bears the burden of handling loss–feedback and supplying retransmissions to a large group of receivers. Local recovery approaches, in which entities other than the sender aid in loss recovery, distribute the loss recovery burden and also reduce network bandwidth consumption and recovery latency. We propose a new local recovery approach that co–locates designated repair servers with routers at strategic locations inside the network. We demonstrate the superior performance of our approach over traditional approaches. We address the important issues of repair server placement, repair server resource requirements, and performance degradation due to insufficient resources. We also demonstrate how the repair server functionality can be provided as a dynamically invocable/revocable service with minimal router support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Many applications have the need to transmit data reliably from a sender to a group of receivers. These applications include one–to–many file transfer, information dissemination (e.g., stock quotes and web cache updates), distance learning and shared whiteboards. These applications can have potentially thousands of receivers spanning wide area networks, and thus efficient data delivery is of paramount importance. Multicast technology provides this efficiency. Multicast is an efficient paradigm for transmitting data from a sender to a group of receivers. Multicast incurs lower network and end–system costs than broadcast to all receivers or multiple unicasts to individual receivers.

The Internet protocol IP [18, 21, 62] supports multicast communication by allowing the transmission of an IP datagram from one host to a set of hosts that form a "multicast group." Membership in an IP multicast group is dynamic. A host may join or leave a multicast group at any time. Based on the location of group members and the sender, multicast routing protocols [4, 19, 20] construct distribution trees for distributing data from the multicast sender to the group members. Each multicast group has a unique IP multicast (class D) address. Packets sent from the sender to a multicast group are addressed to the IP multicast address of that group and routed to the group members using a multicast routing distribution tree. IP multicast has been deployed in the Internet in the form of an overlay multicast capable network called the MBone (Multicast backBone) [11, 48]. It has also been deployed in several corporate Intranets.

Although IP multicast provides support for data forwarding, it guarantees only "best–effort" delivery, as in the unicast case. That is, some of the packets sent from the sender

1

might get lost in the network and not reach all the receivers. Carefully designed multicast loss recovery mechanisms on top of IP multicast must be provided for the reliable delivery demanded by the applications noted above.

The goal of this dissertation is to design and evaluate large scale multicast loss recovery architectures and protocols for IP multicast capable networks, that make efficient use of both the network and end–system resources and scale to applications that can have thousands of receivers spanning wide area networks. We next overview some of the challenges in designing such large scale multicast loss recovery mechanisms.

1.1 Challenges

The challenges in designing large scale multicast loss recovery arise from the fact that the multicast sender must ensure reliable delivery to a large number of receivers instead of one receiver, as in the unicast case. The following three subsections overview three important challenges.

1.1.1 Feedback Implosion

In order to ensure reliable transmission from the sender to the receivers, it is important for the receivers to send feedback either by acknowledging every packet that is received by sending an acknowledgement message (ACK) or by sending a negative acknowledgement message (NAK) for a packet that is detected as lost. In case of a large number of receivers, an ACK for every packet from every receiver would lead to a so–called "ACK implosion" at the sender, which would be busy processing a large number of ACKs and would have little time to transmit data. Another problem with an ACK–based approach is that the sender has to maintain a list, for each packet, of the receivers from which it received an ACK. This could be an important overhead, especially when the number of receivers is very large. Due to the possibility of ACK implosion and the requirement of maintaining the identity of each receiver, an ACK–based approach for reliable multicast is generally not considered to be

scalable [71]. A NAK–based approach is more practical because the number of NAKs is likely to be much less than the number of ACKs. Also, with a NAK based approach the sender does not need to be aware of the number and identity of receivers; the responsibility of loss recovery is now on the receiver – a receiver wishing a retransmission simply sends a NAK to the sender. However, when the network is large and the loss probabilities are high we can have a NAK implosion too. Hence one of the most important challenges in designing reliable multicast is how to deal with NAK implosion.

In order to deal with NAK implosion two approaches have been proposed in earlier work on reliable multicast. In one approach, random timers are used for the purpose of NAK suppression [22, 63, 71]. A receiver, on detecting a loss, waits for a random amount of time and multicasts a NAK to all the group members. While waiting, if a receiver receives a NAK for the same packet it suppresses its own NAK. In the second approach [43, 47, 68, 78], NAKs are aggregated by building trees or hierarchies of receivers, routers, or servers with the sender at the root of the tree (i.e., at the highest level of hierarchy). Here, receivers send NAKs to their parent node at the next higher level of hierarchy. The parent nodes aggregate NAKs before forwarding them up the tree towards the sender. In this dissertation we use both approaches to reduce NAKs sent to the sender. Our research focuses primarily on the next two problems.

1.1.2 Retransmission Scoping

Recall that when a packet is lost by a receiver, the receiver requests (via a NAK message) a retransmission from the sender. The sender could either unicast the retransmission to that receiver or multicast it to all group members. When losses are high and the number of receivers is large, unicast of retransmissions is not efficient because several identical retransmissions to many receivers might be required. In such cases, multicasting of retransmissions may be more suitable. If the retransmissions are multicast on the same multicast group (i.e., on the group which all receivers have joined), then retransmissions are sent to
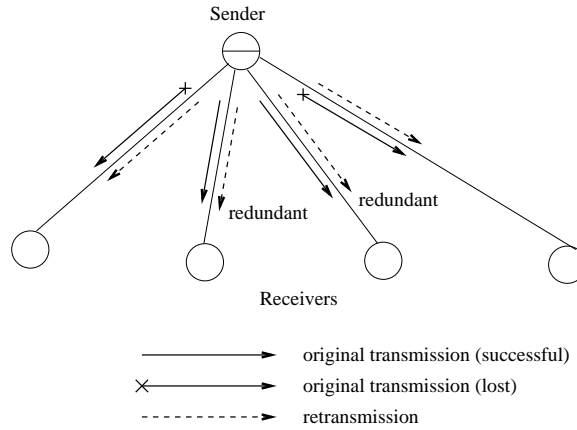
Figure 1.1. Retransmission Scoping Problem

all receivers, not just the ones that have lost the packet. This scenario is shown in Figure 1.1. This causes the unnecessary use of bandwidth on links leading to those receivers who have already received the packet, as well as unwanted packet processing at these receivers. This problem is further aggravated in the presence of heterogeneity, as a large part of the network may receive retransmissions that are needed only by a small fraction of lossy receivers. Hence a fundamental problem in reliable multicast is how to scope retransmissions so as to shield receivers (and the links leading to them) from loss recovery due to other receivers.

In Chapter 2, we present a new approach for solving the retransmission scoping problem. In this approach, a single multicast channel[1] is used for the original transmissions of packets. Retransmissions of packets are sent on separate multicast channels, which receivers dynamically join and leave. This use of multiple multicast channels aims to scope retransmissions so that they are received only by the receivers that require them.

---

[1]We use multicast channel as a more general term than an IP multicast group. A multicast channel can be implemented by an IP multicast group as well as other mechanisms, as shown in Chapter 2.
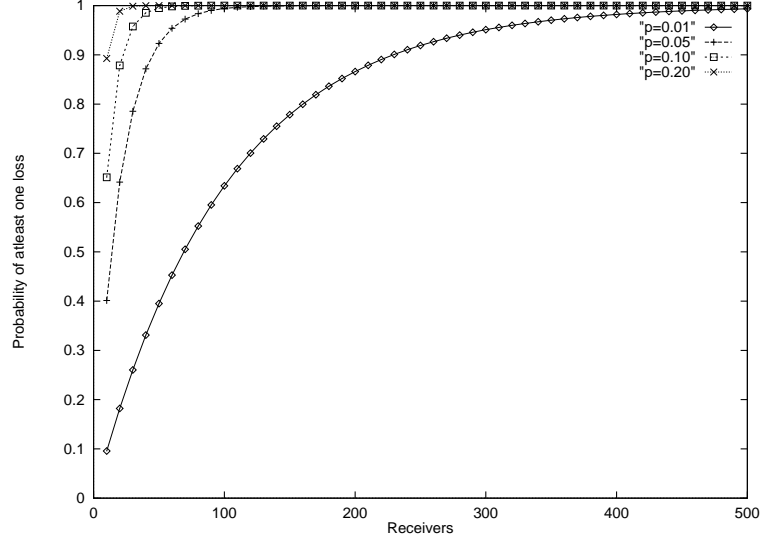
Figure 1.2. Probability of at least one packet loss among the receivers *vs* Number of Receivers

1.1.3 Burden of Recovery

When the number of receivers is large and/or the loss probabilities are high, the probability of at least one receiver losing the packet is very high. If there are R receivers, each of which sees independent packet loss with probability p, the probability of at least one receiver losing a packet is $1 - (1 - p)^R$. In the case that one percent of packets are lost ($p = 0.01$), the probability of at least one loss among the receivers is almost 1 for 500 receivers (as shown in Figure 1.2). Since the actual number of receivers could be much larger than 500, almost every packet needs to be retransmitted, often several times. In this case the sender and the links close to the sender are likely to become bottlenecks, thereby degrading performance. Efficient distribution of the sender's loss recovery burden is thus an important challenge.

Local recovery approaches (as shown in Figure 1.3) [22, 47] in which an entity other than the sender, such as a nearby receiver or server, aids in loss recovery at the receivers can be used to distribute sender's burden of loss recovery. In local recovery, servers or receivers cache packets and themselves supply retransmissions on request, thus sharing the

burden of the sender. Local recovery of lost packets saves network bandwidth over *global* recovery from the sender. The loss recovery latency is also reduced since the location from which a lost packet is recovered is typically closer than the sender to the receiver. When a lost packet cannot be recovered locally, it is recovered from the sender. We call the loss recovery from a nearby server "server–based local recovery" and from a nearby receiver "receiver–based local recovery." An important question to ask here is which of the two approaches is better. In the server–based approach, servers could be placed at the edges of the network in the receiver domain or they could be placed inside the network co–located with routers. If the servers could be placed inside the network, they could exploit detailed knowledge of the underlying network to enhance their performance. In comparing server–based and receiver–based local recovery, we find (in Chapter 3) that the server–based approach exhibits higher performance when servers are co–located with routers above lossy links.

The server–based approach will perform efficiently only when sufficient buffering and processing resources are available at the servers. The servers require buffers to cache packets in order to be able to later retransmit them to repair losses. Servers also need to process transmissions, retransmissions and retransmission requests. Hence it is important to understand the resources requirements of a server that provides local recovery, the factors that influence these resource requirements, and how performance degrades when there are insufficient resources. We address these issues in Chapters 3 and 4.

Another important issue in server–based local recovery is that, in the Internet, *static* deployment of servers might not be useful due to changing network conditions (e.g., route changes [58], or changes in loss behavior [79, 80]). In order to substantially benefit from the server–based approach, it is important to make it more dynamic and flexible. The required dynamism and flexibility could be provided using the recently proposed *active* services [2] paradigm. Active services advocate the dynamic placement of user defined computations within the network [2], while preserving the routing and forwarding seman-
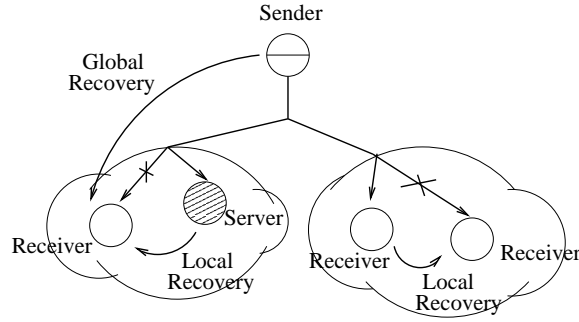
Figure 1.3. Local Recovery Scenarios

tics of the current Internet architecture. Active services could be deployed as application level programs inside routers or, equivalently, on servers co–located with routers. In our server–based local recovery context, the challenge is to provide the server functionality as a dynamically invocable/revocable active service. We address this issue in Chapter 5. We refer to the dynamically invocable/revocable server functionality as active repair service.

1.2 Contributions of this Dissertation

The following is a summary of the contributions of this dissertation.

- We present a new approach to scoping retransmissions in which a single multicast channel is used for the original transmission of packets and separate multicast channels are used for scoping retransmissions to "interested" receivers only. We find that a small to moderate number of multicast channels can be recycled to achieve almost perfect retransmission scoping for a wide range of system parameters. We also propose two mechanisms for implementing retransmission channels, one using multiple IP multicast groups and the other using a single IP multicast group in conjunction with additional router support. We show that the second approach reduces both host processing costs and network bandwidth usage.

7

- We propose a new local recovery approach in which designated servers are co–located with routers at strategic locations inside the network. We compare the performance of this server–based approach with two receiver–based approaches in which only the end–hosts, sender and receivers, are involved in loss recovery. We first demonstrate that the three local recovery approaches yield significantly higher throughput and lower bandwidth usage than an approach that does not use local recovery. We then demonstrate that server–based local recovery yields higher throughput and lower bandwidth usage than receiver–based local recovery.

- We analyze the processing and buffering requirements of a server in our server–based local recovery approach, and determine the dependence of performance of server–based local recovery on server buffer size. In determining the buffer requirements, we examine three buffer replacement policies including "oldest–first," "least–recently–used" and a new policy, which we propose, that replaces packets using "oldest–first" policy but ensures that every packet is buffered for a minimum amount of time before it is replaced. Based on our performance study and noting that it could be easily implemented, we recommend the use of the simple "oldest–first" replacement policy.

- We design a multicast loss recovery architecture and protocol that uses active repair services to show how the challenges of NAK implosion, retransmission scoping and distribution of loss recovery burden can be handled flexibly and efficiently with minimal router support. We also show how active repair services could be located and invoked/revoked through a simple signaling mechanism.

1.3   Structure of the Dissertation

In this introduction we have overviewed the important challenges in the design of large scale multicast loss recovery and summarized our contributions that address these challenges. In Chapter 2 we present a new approach for solving the retransmission scoping

problem by using multiple multicast channels. We also propose and examine specific mechanism for implementing this approach. In Chapter 3 we propose a new server–based local recovery approach and demonstrate the higher performance that this approach achieves in comparison to the approaches that either do not use local recovery or use receiver–based local recovery. We also examine the processing requirements of servers in the server–based approach. In Chapter 4 we examine the buffering requirements of the server–based approach and show how network and end–host performance degrades as the server buffer size is reduced. We also examine several buffer replacement policies and demonstrate that a simple "oldest–first" policy performs nearly as well as the other more complex policies. In Chapter 5 we present the design of an architecture and protocol that use active repair services and show how the challenges of large scale multicast loss recovery can be addressed flexibly and efficiently with minimal router support. Each of the Chapters 2–6 also contains the related work on the problems described therein. The summary of this dissertation and directions for future work are contained in Chapter 6.

CHAPTER 2

RETRANSMISSION SCOPING USING MULTIPLE MULTICAST CHANNELS

In this chapter we present an approach for providing retransmission scoping for scalable, reliable multicast communication with the goal of reducing network and receiver processing overhead. This approach (first suggested in [13] and [16], but never studied) involves the use of multiple multicast channels. To illustrate the retransmission scoping problem, consider a reliable multicast scenario using a single multicast channel. All packet transmissions and retransmissions are done over the single multicast channel. Each receiver therefore receives all of the retransmissions of a packet, even after correctly receiving the packet. This imposes unnecessary receiver processing overhead, especially as the number of receivers increases.

The approach that we examine in this chapter allows one to overcome this *problem of a receiver processing unwanted redundant packets in a multicast scenario*. The approach can be informally described as follows. Consider a system with a sender and R receivers such that data is transmitted reliably from the sender to the receivers using $G + 1$ channels. One channel is used for the original transmission of packets from the sender. A lost packet is retransmitted on one of the remaining G channels. Packets are assigned in a deterministic manner to the channels. After detecting a packet loss, a receiver "joins" the appropriate channel for that packet's retransmission. Once the lost packet is received, the receiver "leaves" the appropriate channel. Previous work on reliable multicast has focussed on the the case $G = 0$, in which case all transmissions and retransmissions are sent on the same multicast channel.

We begin by showing that protocols using an infinite number of multicast channels incur much less processing overhead at the receivers compared to protocols that use only a single multicast channel. Next, through analysis, we derive an expression for the number of unwanted packets at a receiver due to using only a finite number of multicast channels, for a specific negative acknowledgment (NAK)–based protocol. We then study the behavior of the minimum number of multicast channels required to keep the cost of processing unwanted packets to a sufficiently low value (i.e., to achieve most of the benefit of using an infinite number of multicast channels). For an application consisting of a single sender transmitting reliably to many receivers (referred to as a *one–to–many* application [71]) we find that only a small number of multicast channels are required for a wide range of system parameters. In the case of an application where all participants simultaneously act as both senders and receivers (referred to as a *many–to–many* application [71]) a larger but still a moderate number of multicast channels is needed.

It is important to note that the use of multiple multicast channels reduces the network bandwidth consumption as well as receiver processing cost. This is because retransmissions are only forwarded along the links leading to subscribers of the retransmissions channels. However, in practice the realizable bandwidth savings is tied to the mechanism used to implement multiple multicast channels. In this chapter, we explore two mechanisms, one using multiple IP multicast groups and the other using a control–on–demand active networking architecture (see [25]). We identify the advantages and disadvantages of both mechanisms and discuss their impact on performance in terms of end–host and network resources. We find that due to practical limitations, including potentially high join and leave latency, achieving bandwidth savings in the network may be difficult using current IP multicast. The active networking approach of implementing multiple multicast channels, offers better opportunities for saving both sender/receiver processing costs and network bandwidth.

Even though the use of multiple multicast channels helps to reduce end–host processing and network bandwidth, it is unlikely to have any significant impact on the loss recovery latency. This is because lost packets must still be recovered from the sender. Hence, we do not study loss recovery latency in this chapter.

The remainder of the chapter is organized as follows. In the next section we examine existing work on the subject. In Section 2.2 we present three protocols that use multiple multicast channels. In Section 2.3, we analyze the processing cost performance of these protocols. In Section 2.4, we present numerical results to show that use of multiple multicast channels leads to reduction of processing overhead at the receivers. In Section 2.5 we derive the number of multicast channels required to keep the processing overhead due to unwanted packets at a receiver to a sufficiently low value for a specific NAK based protocol. In Section 2.6 we present mechanisms for implementation of multiple multicast channels. Conclusions and suggestions for future work are contained in Section 2.7.

2.1   Related Work

Reliable multicast has been an active research area in the last few years. Several architectures and protocols have been proposed. In this section we present some of the existing and on–going research in reliable multicast.

One of the most popular existing reliable multicast protocols is scalable reliable multicast, SRM [22]. SRM is a NAK–based protocol that has been implemented for a shared whiteboard application. In its basic form, SRM suffers from the problem of unwanted redundant packets being sent to, and processed at, receivers. Local recovery enhancements in SRM [22] are likely to scale down this problem but not solve it. Local recovery helps to isolate the domains of loss and, thereby, reduce global retransmissions. For a multicast application with thousands of local neighborhoods, unless receivers are arranged in a hierarchy with a small bounded degree and retransmission of packets is properly scoped to remain within the neighborhood, the receivers will still receive unwanted retransmissions

if only one channel is used for both transmissions and retransmissions. Log–based reliable multicast, LBRM [31], and reliable multicast transport protocol, RMTP [47], are two hierarchical approaches[1] in which designated receivers (or loggers) at a certain level supply repairs to lower level designated receivers or loggers. The problem of placing these designated receivers and determining their processing and storage requirements is still being studied.

Recently, there has been an increasing interest in using processing and storage inside the network for enhancing reliable multicast performance. Control–on–demand [25], pragmatic general multicast, PGM [68], and active reliable multicast, ARM [43], propose to maintain retransmission state for selective forwarding of retransmissions and for duplicate NAK suppression. In this chapter, we propose to use the control–on–demand [25] architecture for implementing multiple multicast channels. The analysis presented in this chapter, with minor modifications, is also applicable to PGM. ARM, control–on–demand and L1 [37] propose the use of buffering "current–data" at strategic locations inside the network for providing local retransmissions. Even though it has been shown in [37] that buffering data inside the network for the purpose of retransmissions, results in higher performance, the issues of where to place buffers and when and how to invoke "repair service" remain open questions. We will address some of these issues in Chapters 3 and 5. In this chapter we focus on error recovery only from the sender.

In [22] an approach based on IP *Time–to–live* or TTL has been proposed for scoping retransmissions. There are two problems with TTL based scoping. First, TTL based scoping limits packets within a radius and is not suitable for tree structures as in the case of multicast. Second, it is hard to approximate a good TTL value. In [46] an approach that assigns group–relative routing and distance labels to the routers in a multicast group has been proposed. In this approach retransmission scoping is achieved with the help of positional

---

[1]Several other hierarchical approaches such as [29, 45, 78] also exist.

routing within the multicast group. The problem with this approach is that in addition to requiring substantial changes to the routers it also incurs the overhead of distributing and updating positional and distance labels. This overhead could be high, especially in the presence of highly dynamic groups.

We now look at existing work that uses multiple multicast groups or channels. Cheung *et al.* [14], McCanne [50] and Vicisano [74] have proposed to use multiple multicast groups for flow and congestion control, but not for error recovery. In [22] the possibility of using separate multicast groups for defining "local groups" for local recovery has been suggested. The problem with this approach is that there is no easy way to group receivers so a large number of multicast groups will be required when there is a multi–level hierarchy. Cheriton [13] and Crowcroft [16] first suggested the use of multiple multicast groups for error recovery in reliable multicast, in a discussion on the end–to–end mailing list. Holbrook [31] proposes the use of separate retransmission channels for error recovery as future work.

Even earlier, Ammar and Wu [1] proposed the idea of destination set splitting for improving the throughput of some specific positive acknowledgment based point–to–multipoint protocols. They suggested that receivers could be divided into groups based on their capabilities and that the sender would carry out as many simultaneous independent conversations as the number of groups. Our work, inspired by Cheriton and Crowcroft's suggestion, differs from Ammar's work in three significant ways. First, we do not group receivers based on their capabilities. Rather we group packets such that retransmissions of packets belonging to each group is done on a separate multicast channel. Second, we have considered generic NAK–based protocols instead of specific ACK–based protocols. Third, we have also considered the multipoint–to–multipoint scenario.

Among the existing analytical work on reliable multicast, the work of Towsley, Kurose and Pingali presented in [61, 71] provides a simple analytical framework for studying the performance of reliable multicast protocols. They have used this framework for a quanti-

tative demonstration of the superiority of receiver–initiated NAK–based approaches over sender–initiated ACK–based approaches. This work has subsequently been used in several performance analyses such as [37, 44, 45, 51]. It also forms the basis of our analyses.

2.2 Protocols and System Model

We now present three generic NAK–based protocols for using multiple multicast channels for reliable multicast from a sender to several receivers. Based on arguments presented in [22] and [61, 71], receiver–based reliability schemes (i.e., NAK–based schemes) outperform schemes employing sender–based reliability in providing reliable multicast for many applications of interest. Hence we focus only on receiver–based recovery, or negative acknowledgment (NAK)–based schemes in our work. The section ends with a description of the applications and the network model.

2.2.1 Protocol Description

The protocols described below are modified versions of the generic protocols, N1 and N2, proposed in [61, 71]. In [61, 71], the authors have considered only one multicast channel for both transmissions and retransmissions. Protocols N1 and N2 are described in the appendix. The reliable multicast protocols we will consider will be denoted P1, P2 and P3. We initially make the assumption that we have an infinite number of multicast channels at our disposal so that each packet can be recovered on a separate channel, i.e., $G = \infty$. Later, we will observe that only a small number of multicast channels are required to achieve almost the same effect as can be obtained with an infinite number of channels. The discussion below is based on an IP multicast like network scenario, where the sending of a packet on a multicast channel causes that packet to be sent (potentially with some loss) to the members subscribing to that channel.

Protocol P1 is a modified version of protocol N1 [61, 71]. It exhibits the following behavior

- the sender sends all original transmissions on a multicast channel A_{org}.

- when required, the sender retransmits a packet with sequence number i on multicast channel A_i, where $i = 0, 1, 2, ...$

- whenever a receiver detects a lost packet i, it subscribes to the multicast channel A_i and transmits a NAK to the sender over a point–to–point channel and starts a timer.

- the expiration of a timer without prior reception of the corresponding packet serves as the detection of a lost NAK or retransmission, a NAK is retransmitted for the associated packet, and a timer is started again.

- on receiving packet i on A_i, a receiver unsubscribes from A_i.

Protocol P1 is equivalent to N1 in [61, 71] when only A_{org} is used for both transmissions and retransmissions.

Protocol P2 exhibits the following behavior

- the sender sends all the original transmissions on a multicast channel A_{org},

- the sender retransmits packet i on multicast channel A_i, where $i = 0, 1, 2, ...,$

- whenever a receiver detects a lost packet (say i), it subscribes to multicast channel A_i, waits for a *random period of time* and then multicasts a NAK on the channel A_i, and starts a timer,

- upon receipt of a NAK for a packet that a receiver has not received, but for which it (the receiver) has initiated the random delay prior to sending a NAK, the receiver sets a timer and behaves as if it had sent that NAK,

- the expiration of a timer without prior reception of the corresponding packet serves as the detection of a lost NAK or retransmission.

- on receiving packet i on A_i, a receiver unsubscribes from A_i.

16

Protocol P2 is equivalent to N2 in [61, 71] when only A_{org} is used for both transmissions and retransmissions. Protocol P3 exhibits the same behavior as P2 except that a receiver sends a NAK for packet i on the original multicast channel A_{org} instead of on A_i. The important similarity between P2 and P3, which distinguishes them from P1, is that both suppress NAKs [22] to the sender. They attempt to ensure that at most one NAK is sent out to the sender per packet by delaying the generation of the NAKs and multicasting them to all the participating receivers. This suppression of NAKs to the sender does not come for free, however. The price is paid in terms of extra NAK processing at the receivers as the NAKs are now multicast instead of being unicast to the sender. P2 reduces the NAK processing cost at the receivers by sending NAKs for packet i on A_i. Only those receivers that have not received packet i, subscribe to A_i. Hence NAKs are processed only at a few receivers. In comparison, in P3, NAKs are retransmitted on A_{org} and are received by all the receivers that have subscribed to A_{org}. It should be noted that the sender has to subscribe to all retransmission channels in P2 and only subscribe to channel A_{org} in P3. A sender does not have to subscribe to any retransmission channel in P1.

Before quantitatively evaluating the performance of P1, P2 and P3, let us first qualitatively examine their behavior. In P2, NAKs are received *by only* those receivers that have lost packet i. Thus a lost packet can only be recovered from the sender and *not* from a *local* receiver. Additional mechanisms would have to be provided for local recovery. In contrast, in P3, NAKs are received by all the receivers participating in the multicast session. Some of these receivers might have received packet i correctly and could then retransmit the NAKed packet. Thus P3 could be easily modified to include local recovery from other receivers. Another difference is that the performance of P2 is sensitive to the latency associated with detecting packet loss. If two receivers incur very different latencies in detecting the loss of the same packet, it is possible for one to return a NAK prior to the other joining the appropriate group. Consequently the second receiver will miss that NAK and may transmit

its own redundant NAK. For P3, since all NAKs are sent to A_{org}, and since all receivers subscribe to A_{org}, this situation would not occur.

2.2.2 System Model

When examining the performance of P1, P2 and P3, we will study two different system models, corresponding roughly to two broad classes of applications that use reliable multicast. In the first model, corresponding to the *one–to–many* application (e.g., telelecturing), we assume that one sender transmits a continuous stream of packets to R identical receivers. In the second model, corresponding to the *many–to–many* application (e.g., distributed interactive simulations [33]), we assume that there are $R + 1$ identical nodes in the system. All nodes can function as both a sender and receiver. In this model we assume that for each packet there is a single sender and each node is equally likely to be the sender. That is, a node is a sender with a probability $1/(R + 1)$ and is a receiver with probability $R/(R + 1)$ [71]. For both models we assume that all loss events at all receivers for all transmissions are mutually independent and that the probability of packet loss, p, is independent of receiver. Our analysis can be generalized to the case where the probability of packet loss is distinct for different receivers. We further assume that NAKs are never lost. This assumption can be relaxed by following the analysis given in [72].

As noted earlier, we will begin our analysis below by assuming an infinite number of available multicast channels and later consider the case when there is a limited number of channels.

2.3 Processing Cost Analysis

For the purpose of understanding the improvement in processing costs of protocols P1, P2 and P3 over protocols N1 and N2 [71], we now analyze processing costs for protocols P1, P2 and P3. The bandwidth improvement is tied to the implementation and hence will be studied in the section (Section 2.6) on implementation mechanisms. For analyzing the

processing costs, we start with the *one–to–many* model and later use the results of the *one–to–many* model to obtain processing costs for the *many–to–many* model.

The receive processing cost is determined by computing the processing involved in *correctly* receiving a randomly chosen packet. This includes the time required to receive those copies of this packet (i.e., the original copy plus any retransmissions) that arrive at the receiver, the time required to send/receive any NAKs associated with this packet, and the time needed to handle any timer interrupts associated with this packet. The send processing cost is determined by the processing involved in correctly transmitting a packet to all receivers. This includes the cost required to process the original transmission of the packet, process any received NAKs, and process retransmissions that are sent out in response to these NAKs.

We now derive expressions for receiver processing requirements for protocols P1, P2 and P3. Table 2.1 describes the notation used in the analysis. Most of the notation has been reintroduced from [71]. We assume that the processing times have general distributions and that they are independent of each other.

Following an approach similar to the one in [71], the mean per packet processing time for a randomly chosen packet at a receiver for P1 can be expressed as,

$$E[Y^{P1}] = E[Y_p] + pE[Y_j] + E[(M_r - 1)^+]E[Y_n] + E[(M_r - 2)^+]E[Y_t] \qquad (2.1)$$

where $(x)^+ = \max\{0, x\}$. The first term corresponds to the processing required to correctly receive a packet. A receiver will only receive one copy of the packet (either on A_{org}, or on A_i for packet i). The next term represents the processing required for joining and leaving a multicast channel. Note that the join and leave processing times have been multiplied by the loss probability, p, because this cost is incurred only when a packet is lost and a NAK is transmitted. Although several NAKs might be sent from a receiver to recover a lost packet, a receiver needs to join and leave the corresponding multicast channel at most once per packet recovery. The third term represents the processing required to prepare and return

19

X_p	–	sender time to process the transmission of a packet
X_n	–	sender time to process a NAK
Y_p	–	receiver time to process a newly received packet
Y_t	–	receiver time to process a timeout
Y_n	–	receiver time to process and transmit a NAK
Y_n'	–	receiver time to receive and process a NAK
Y_j	–	receiver time to process a join and the corresponding leave
X^ω, Y^ω	–	the send and receive per packet processing time in protocol $\omega \in \{$P1, P2, P3, N1, N2 $\}$
Z^ω	–	the sum of send and receive per packet processing time at a node in protocol $\omega \in \{$P1, P2, P3, N1, N2$\}$ (used in the many–to–many case)
p	–	the loss probability at a receiver
R	–	the total number of receivers
M_r	–	the number of transmissions necessary for receiver r to successfully receive a packet
M	–	the number of transmissions for all receivers to receive the packet correctly; $M = max_r\{M_r\}$

Table 2.1. Notation Used in the Processing Cost Analysis

NAKs. The last term corresponds to the processing of the timer when it expires. Since M_r, the number of times a message must be sent before a given receiver receives the packet, is independent of the number of multicast channels (and indeed is independent of whether an ACK or NAK protocol is used), we can use the results in [71] directly:

$$E[M_r] = 1/(1-p), \tag{2.2}$$

$$E[(M_r - 1)^+] = p/(1-p), \tag{2.3}$$

$$E[(M_r - 2)^+] = p^2/(1-p). \tag{2.4}$$

Substitution of (2.3) and (2.4) into (2.1) yields

$$E[Y^{P1}] = E[Y_p] + pE[Y_j] + pE[Y_n]/(1-p) + p^2 E[Y_t]/(1-p). \tag{2.5}$$

For P2 the mean per packet processing time can be expressed as,

$$
\begin{aligned}
E[Y^{P2}] &= E[Y_p] + pE[Y_j] \\
&\quad + (E[(M_r - 1)])(E[Y_n]/R + (R-1)E[Y_{n'}]/R) \\
&\quad + E[(M_r - 2)^+]E[Y_t] \tag{2.6}
\end{aligned}
$$

Here, the first two terms are the same as in the case of P1. The third term consists of two parts. The first one results from NAK generation and the second results from NAK reception. A receiver generates a NAK with probability $1/R$ and receives a NAK with probability of $(R-1)/R$. The last term is the same as that for P1. Substitution of (2.2) and (2.4) into (2.6) yields

$$E[Y^{P2}] = E[Y_p] + pE[Y_j] + p/(1-p)(E[Y_n]/R + (R-1)E[Y_{n'}]/R) + p^2 E[Y_t]/(1-p) \tag{2.7}$$

21

Using similar arguments, the mean per–packet processing time at a receiver for the P3 protocol can be expressed as:

$$E[Y^{P3}] = E[Y_p] + pE[Y_j] + (E[M] - 1)(E[Y_n]/R + (R-1)E[Y_{n'}]/R)$$
$$+ p^2 E[Y_t]/(1-p) \qquad (2.8)$$

The only difference between the expressions for P2 and P3 is the replacement of $E[M_r]$ in P2 by $E[M]$ in P3. Recall that in P2 a receiver sends a NAK for packet i to address A_i. A receiver r, that is trying to recover packet i, will subscribe to A_i for only the time until it receives i. Since all NAKs are sent to address A_i, the number of NAKs it will receive during this time is M_r. On the other hand, in protocol P3, a NAK is sent to the original transmission address A_{org} and all the NAKs sent for packet i are received by all receivers. $E[M]$ can be expressed in terms of R and p as follows.

$$E[M] = \sum_{m=1}^{\infty} mP(M = m) = \sum_{m=1}^{\infty} P(M \geq m),$$
$$= 1 + \sum_{m=1}^{\infty} (1 - (1 - p^m)^R)$$

In Section 2.6.2 we will observe that when we use the control–on–demand active networking to implement multiple channels the receivers do no incur any join/leave processing cost as above. The mean per–packet processing times at a receiver, for the protocols N1 and N2 are given by the following expressions from [71],

$$E[Y^{N1}] = E[M](1-p)E[Y_p] + pE[Y_n]/(1-p) + p^2 E[Y_t]/(1-p) \qquad (2.9)$$
$$E[Y^{N2}] = E[M](1-p)E[Y_p] + (E[M] - 1)(E[Y_n]/R + (R-1)E[Y_n']/R)$$
$$+ p^2 E[Y_t]/(1-p) \qquad (2.10)$$

22

The sender processing costs of protocols P1 and P2 are the same as that for protocol N1, the sender processing cost of protocol P3 is the same as that for protocol N2. This is because the use of multiple multicast channels only reduces the processing of redundant packets at the receivers. There is no change in the number of NAKs received by the sender and hence there is no change in the number of packets sent out by the sender. There is no per–packet join/leave processing cost at the sender. In P1 and P3, the sender does not subscribe to any retransmission channel because it does not receive anything on the retransmission channels. In P1, NAKs are received point–to–point and in P3, NAKs are received on the original transmission multicast address. In P2 the sender must subscribe to all of the retransmission channels before it starts transmitting packets because it receives NAKs on these retransmission channels. It remains a member until the multicast session ends. Hence the sender does not incur any join/leave processing costs during the session. We can thus use the expressions of sender processing costs from [71]. The mean *sender* processing time needed to successfully transmit a packet to all the receivers, for protocols P1, P2 and P3 is given by the following expressions.

$$E[X^{P1}] = E[X^{N1}] \;\; = \;\; E[M]E[X_p] + RpE[X_n]/(1 - p) \quad (2.11)$$

$$E[X^{P2}] = E[X^{P3}] = E[X^{N2}] \;\; = \;\; E[M]E[X_p] + (E[M] - 1)E[X_n] \quad (2.12)$$

Later in Section 2.6.2, we will see that when we use the control–on–demand active networking architecture to implement multiple multicast channels, the number of NAKs received by the sender in P1 reduces to that received by the sender in P2. This is because the control–on–demand active networking architecture allows duplicate NAK elimination at routers.

Recall that under the *many–to–many* scenario, each of the $R + 1$ end system nodes are equally likely to be the sender of the randomly chosen packet. Hence the mean packet processing time is expressed as

$$E[Z^\omega] \;\; = \;\; E[X^\omega]/(R+1) + R\,E[Y^\omega]/(R+1) \qquad (2.13)$$

where $\omega \epsilon \{P1, P2, P3\}$.

- **Heterogeneous Receivers**

The above analysis can be extended to include the case when the receivers are heterogeneous. If receiver r loses a packet with probability p_r, then the mean receiver processing costs for receiver r, under protocols P1, P2 and P3, can be obtained by replacing p by p_r in equations (2.5)–(2.8). The mean receiver processing costs for receiver r, under protocols N1 and N2, can be obtained by replacing p by p_r in equations (2.9) and (2.10). Equations (2.2)–(2.4) also remain the same except that p is replaced by p_r. $E[M]$ is now given by the following equation.

$$E[M] \;\; = \;\; 1 + \sum_{m=1}^{\infty} \left(1 - \prod_{r=1}^{R} (1 - p_r^m)\right)$$

The expressions for $E[X^{P2}]$, $E[X^{P3}]$ and $E[X^{N2}]$ remain the same as given by equations (2.11) and (2.12). The expression for $E[X^{P1}]$ and $E[X^{N1}]$ is now given by the following equation.

$$E[X^{P1}] = E[X^{N1}] \;\; = \;\; E[M]E[X_p] + \sum_{r=1}^{R} p_r E[X_n]/(1 - p_r)$$

2.4   Numerical Results

We now examine the relative performance of protocols P1, P2 and P3 and N1 and N2. In order to do so, we need to know the processing times associated with sending/receiving a data packet and a NAK packet, as well as their interrupt processing times. In order to measure these values, we instrumented a Linux kernel version 1.2.13 on a 150 MHz Pentium PC. As we had complete control over the processes running on PC, we ensured that the PC had the minimum possible load and performed all of our measurements inside

the Linux kernel. We considered two packet sizes, 1024 bytes for data packets and 32 bytes for NAK packets. The results of our measurements are summarized in the Table 2.2. Each

Operation	Mean Processing Time (μs)	Standard Deviation
Send–Data	502	13.8
Send–NAK	87	5.8
Recv–Data	487	12.6
Recv–NAK	86	5.1
Join	75	6.6
Leave	74	6.8
Timer	32	1.5

Table 2.2. Processing Time(in microseconds)

processing time, in microseconds, was measured 1000 times and an average reported. The timer processing time includes the time to set, execute and delete the timer. To determine the join and leave processing time, we sequentially performed 20 join operations of distinct IP multicast groups followed by 20 leave operations. The join and leave operations were "local" meaning that no IGMP reports were sent out as part of these operations. Higher processing costs will incur if IGMP reports are also sent out. The concept of "local" join and leave is discussed later in the section on local filtering.

It is worth noting that several measurement studies of per–packet processing times have been reported in the literature, notably [35] and [56]. However, neither of these included measurements of join and leave processing times, thus necessitating the series of measurements described above.

2.4.1 One–to–many Model

Using the measured processing times, we can compute the mean send and receive processing costs(times) for the protocols P1, P2, P3, N1 and N2 using equations (2.5)–(2.12), for several values of R and p.

Figures 2.1–2.3 show how the ratios of receiver processing costs obtained under the P and N protocols vary with R and p. It can be seen in each of these graphs that the family
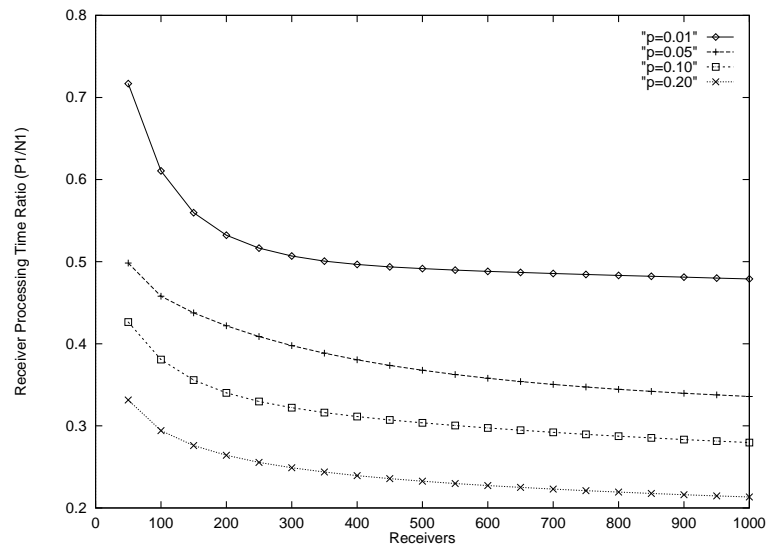
Figure 2.1. Receiver processing cost reduction of P1 over N1
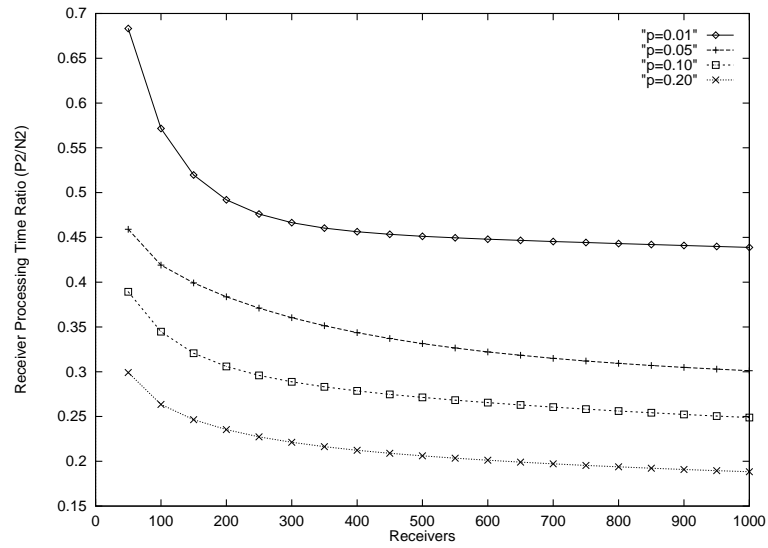
Figure 2.2. Receiver processing cost reduction of P2 over N2
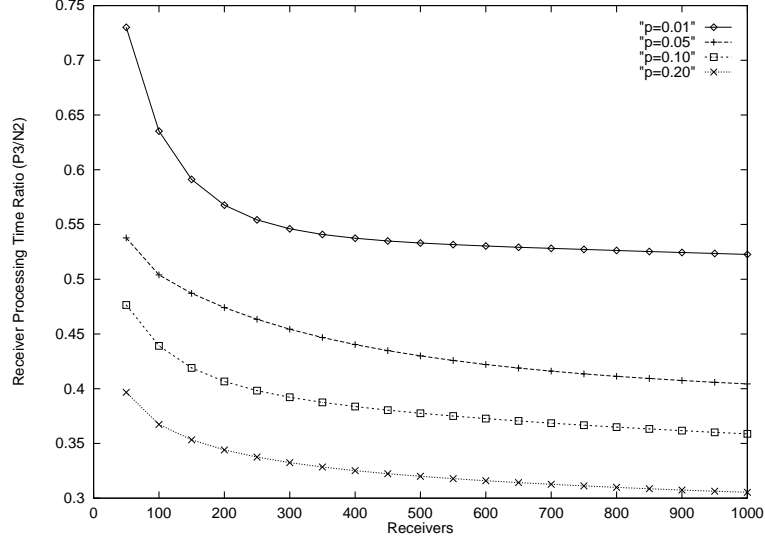
Figure 2.3. Receiver processing Cost Reduction of P3 over N2

of P protocols always perform better. This is because the P protocols, by using multiple multicast channels, eliminate the reception of unwanted and redundant data packets at the receivers and hence receiver processing costs decrease. Although there is a slight increase in processing costs due to the processing of extra joins and leaves at the receivers, this increase is much less than the benefit obtained by reducing the processing of unwanted packets. Further, the benefit increases as the loss probability, p, and number of receivers, R increases. There are two factors contributing to this behavior. The first factor is that the cost of processing a data packet is much higher than the cost of processing a join or a leave operation; Table 2.2 shows that the cost of processing a data packet at a receiver is over six times more costly than the processing a join or leave. The second factor is that only one join and one leave are required to recover a lost packet at a receiver, whereas in the case of protocols N1 and N2 the number of unwanted packets per packet recovery, $(E[M] - 1)(1 - p)$, at a receiver is greater than one even for a small loss probabilities and small number of receivers. This number increases with p and R.

Figures 2.1 and 2.2 show the receiver processing reduction of P1 over N1 and P2 over N2 respectively. Observe that the relative performance of P2 over N2 is better than that of

27
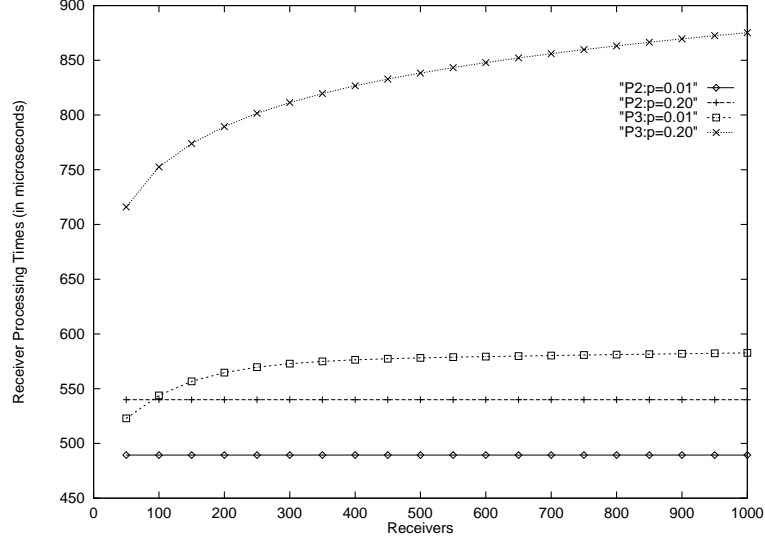
Figure 2.4. Receiver Processing Costs of P2 and P3

P1 over N1. This is because the receiver processing time of N2 is higher than that of N1 [71]. On the other hand the receiver processing times of P1 and P2 are almost same. In fact, from equations (2.5) and (2.7) it can be seen that these are exactly same if $E[Y_n] = E[Y'_n]$, i.e., if the processing time to send a NAK is the same as the processing time to receive a NAK. From Table 2.2 we see that $E[Y_n] = 87$ microseconds and $E[Y'_n] = 86$ microseconds. Hence the receiver processing times of P1 and P2 are same for all practical purposes.

Figure 2.3 shows the receiver processing performance of P3 over N2. Although P3 involves more processing than P2, it still substantially outperforms N2. Note that in Figures 2.1–2.3, the reduction in receiver processing cost flattens out with increasing R. This is because the receiver processing cost of protocols N1 and N2 increases as $\log R$ [71]. The receiver processing cost of P1 and P2 is independent of R. The receiver processing cost of P3 also increases as $\log R$ but much more slowly than N2.

Figure 2.4 shows the receiver processing times of P2 and P3. As expected, we observe that P3 incurs higher receiver processing costs than P2 (and therefore P1), as expected. Recall from Section 2.3 that P3 must perform extra NAK processing in comparison to P1 and P2.

28
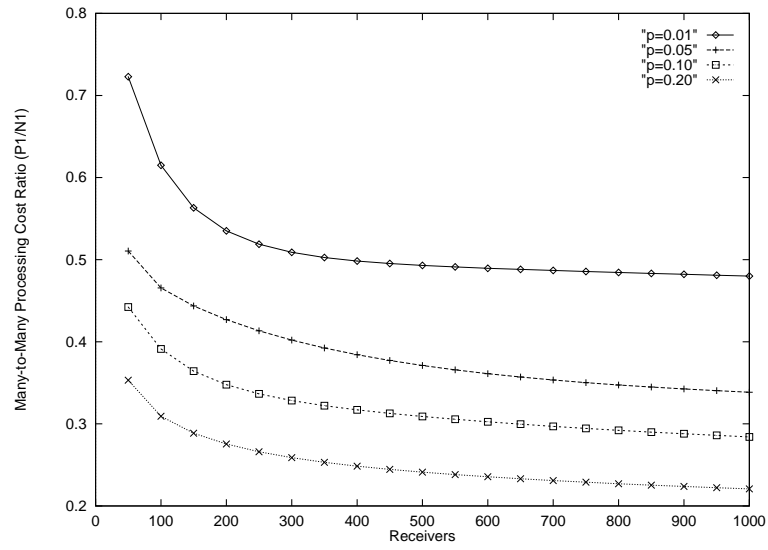
Figure 2.5. Many–to–Many: Overall processing cost reduction of P1 over N1
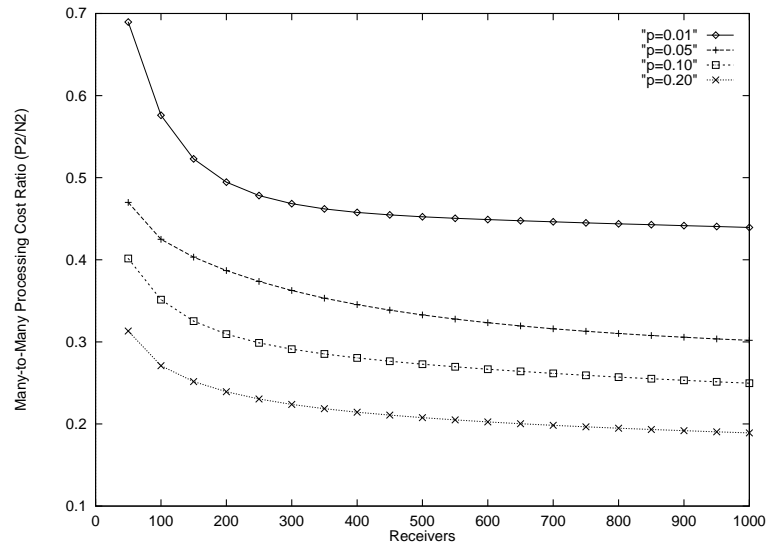
Figure 2.6. Many–to–Many: Overall processing cost reduction of P2 over N2
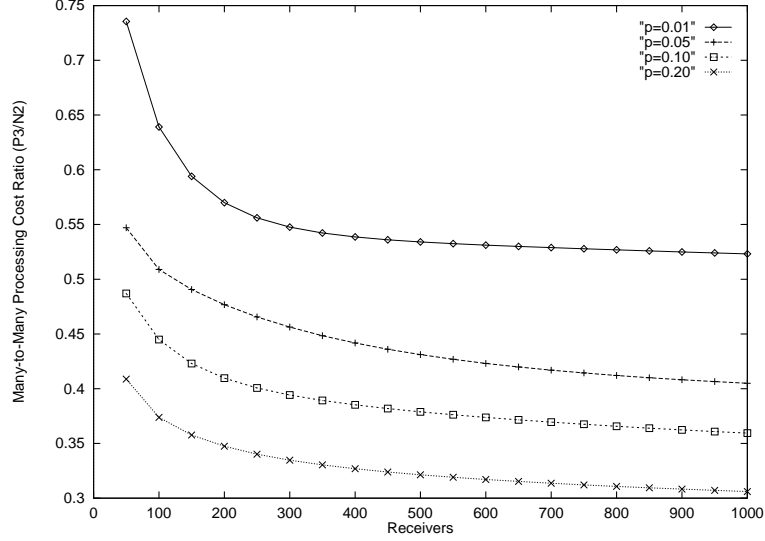
Figure 2.7. Many–to–Many: Overall processing cost reduction of P3 over N2

2.4.2 Many–to–Many Model

In this section, we examine the *many–to–many* model, using equation (2.13) to compute the mean per–packet overall processing cost at a node for protocols P1, P2 and P3 and N1 and N2. Figures 2.5–2.7 show how the *many–to–many* overall processing cost ratio of the P and the N protocols varies with the number of receivers for different loss probabilities. We see in Figure 2.5 that P1 outperforms N1 and in Figures 2.6 and 2.7 that P2 and P3 outperform N2. This follows from the fact that a participant in a *many–to–many* application is much more likely (with probability $R/(R+1)$) to perform receive processing of a packet than send processing. Consequently the mean per packet overall processing cost ratios exhibited in Figures 2.5, 2.6, and 2.7 exhibit behavior nearly identical to the mean per packet receive processing ratios for those protocols that we saw previously in Figures 2.1, 2.2, and 2.3. Among the P protocols, the mean per packet processing cost under P2 is only slightly lower than that of P1. This is because the receiver processing cost is the same for P1 and P2 and the sender processing cost, even though lower for P1 due to NAK suppression, influences the mean per packet processing cost only slightly as R increases. Both P1 and P2 have lower overall processing cost than P3, as they have lower receiver processing cost.

If end host processing were to determine the protocol throughput (i.e., other factors such as network bandwidth are not bottlenecks) then protocol throughput for the *one–to–many* model could be defined as $\min\{1/E[X^\omega], 1/E[Y^\omega]\}$, where $\omega\epsilon\{N1, N2, P1, P2, P3\}$ (see [71]). Similarly, the protocol throughput for the *many–to–many* model could be defined as $1/E[Z^\omega]$. As noted in Section 2.3, the use of multiple multicast channels does not change the sender processing costs. From [71] we know that the sender processing cost is greater than the receiver processing cost. Therefore, the protocol throughput in the *one–to–many* model, determined by the sender processing cost, does not change even when we use multiple multicast channels. However, in the case of *many–to–many* model, because each node acts like a receiver most of the time, the reduction in receiver processing cost obtained by using multiple multicast channels reduces overall processing cost at a node thereby increasing protocol throughput. Thus Figures 2.5, 2.6, and 2.7 also show that the P protocols achieve higher protocol throughput in comparison to the N protocols.

In summary, we observe a significant reduction of receiver processing costs by using multiple multicast channels. For the *many–to–many* application, we also see a substantial reduction in in overall processing costs at a node.

2.5   Finite Number of Retransmission Multicast Channels

In the previous section we made the assumption that the number of available multicast channels was infinite. This is unrealistic and, even if a very large number of multicast channels was available, practical considerations such as the size of routing and forwarding tables within the network would argue in favor of a smaller number of multicast channels. Recall that the main purpose of choosing a multicast–channel–per–packet is to avoid receiving unwanted redundant packets. In this section we demonstrate through analysis that only a small (finite) number of multicast channels is required to keep the overhead of processing redundant packets extremely low – approaching that achievable with an infinite number of channels.

We analyze the *one–to–many* model for protocol P1. The retransmission of packet i is now done on multicast address $A_{i \bmod G}$ where G is the number of retransmission multicast channels. That is, instead of subscribing to A_i, as in the previous section, a receiver needing to recover packet i subscribes to $A_{i \bmod G}$. Unlike the case of $G = \infty$, retransmissions of distinct packets may now be interleaved on the same multicast channel. This interleaving depends upon the retransmission rate of lost packets with respect to transmission rate of *new* (first time) packets. For this reason, we must model the manner in which retransmissions are sent in relation to the new packets. We assume that the sender multicasts new packets periodically with a fixed time interval Δ, and retransmits a packet periodically with a fixed interval Δ' as long as there is a pending NAK for that packet[2]. The value of Δ/Δ' depends upon many factors, such as application requirements, network topology, and network behavior. We will observe that Δ/Δ' is a key parameter in our protocol performance.

2.5.1 Analysis

Our goal in this section is to determine the number of unwanted redundant packets received by a receiver due to the use of only a finite number (G) of multicast channels by protocol P1.

In order to illustrate the factors that will impact performance, let us consider an infinite stream of packets. We pick one packet randomly and label it 0. Suppose that a receiver r' does not receive the original transmission of packet 0. On detecting that it has not received this packet, it joins a retransmission channel corresponding to address $A_{0 \bmod G} = A_0$. It sends a NAK to the sender and sets its NAK retransmission timer to Δ'. The sender retransmits packet 0 on A_0. If this packet is lost again then receiver r''s NAK retransmission timer expires and it retransmits the NAK. While r' is listening to channel A_0, other receivers may use the retransmission channel A_0 to recover packets $\ldots, -3G, -2G, -G, G, 2G, 3G, \ldots$.

---

[2]Realistically, both Δ and Δ' are random variables.
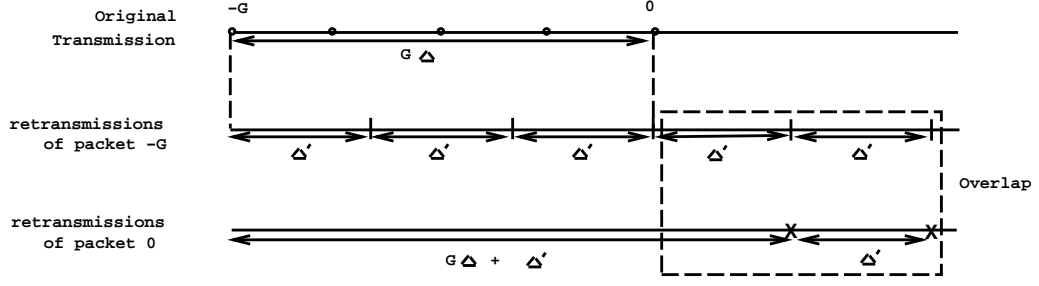
Figure 2.8. Sender timeline

If r' has already received these other packets, then any retransmissions of these packets received by r' are unwanted.

We focus on the number of these unwanted packets received by r'. For this purpose we need to consider the overlap between the retransmissions of packets $\ldots, -3G, -2G,$ $-G, G, 2G, 3G, \ldots$ and retransmissions of packet 0 during the period of time when r' is using A_0 to receive packet 0. Figure 2.8 shows the overlap between retransmissions of packets $-G$ and 0. For simplicity we have shown $G\Delta$ to be an integral multiple of Δ' in Figure 2.8, although this is not a requirement for our analysis. We introduce the random variable U to denote the number of unwanted packets received by r' while it is attempting to receive packet 0, given that it requires at least one retransmission. Its expectation, $E[U]$, can be expressed as,

$$E[U] = (1 - p) \left(\sum_{j=1}^{\infty} (E[Z(-jG)] + E[Z(jG)]) - \sum_{j=1}^{\infty} (E[Z'(-jG)] + E[Z'(jG))] \right)$$

(2.14)

where the random variable $Z(k)$, $k = \pm 1, \pm 2, \ldots$, denotes the number of retransmissions of packet k that overlap with the retransmissions of packet 0 to r' and the random variable $Z'(k)$, $k = \pm 1, \pm 2, \ldots$, denotes the number of retransmissions of packet k that overlap with the retransmissions of packet 0 before r' leaves channel A_0. In the remainder of this section we derive expressions for $E[Z(k)]$ and $E[Z'(k)]$, $k = \pm 1, \pm 2, \ldots$. The notation used in the analysis is described in Table 2.3.

33

$Z(k)$	–	the number of retransmissions of packet k, due to R receivers, that overlap with the retransmissions of packet 0 to receiver r'
$Z'(k)$	–	the number of retransmissions of packet k, due to r', that overlap with the retransmissions of packet 0 to r'
N	–	number of NAKs generated by receiver r' for packet 0
N_r	–	number of NAKs generated by receiver r for any packet
U	–	number of unwanted packets received at a receiver per packet recovery
p	–	the loss probability at a receiver
G	–	the number of retransmission multicast channel
Δ	–	the time interval between successive transmissions from the sender
Δ'	–	the time interval between successive retransmissions from the sender
R	–	the total number of receivers

Table 2.3. Notation Used in the Analysis of Unwanted Retransmissions

For $k = -1, -2, -3, \ldots$, $Z(k)$ is determined by considering the minimum of the number of retransmissions of packet 0 and number of retransmissions of packet k, after r' starts using A_0 for recovering packet 0. The quantity $\lfloor -k\Delta/\Delta' \rfloor$ determines the number of possible retransmissions of k before r' starts using A_0 to recover packet 0. Therefore,

$$Z(k) = \min(N, \max(0, \max_{1 \leq r \leq R}(N_r - \lfloor \frac{-k\Delta}{\Delta'} \rfloor))). \tag{2.15}$$

For $k = 1, 2, 3, \ldots$, i.e., for packets transmitted after packet 0, $Z(k)$ is determined by considering the maximum number of retransmissions of packet k, over all receivers, until r' finishes recovering packet 0.

$$Z(k) = \min(\max(0, N - \lfloor \frac{k\Delta}{\Delta'} \rfloor), \max_{1 \leq r \leq R} N_r). \tag{2.16}$$

In equations (2.15) and (2.16), $P(N = n) = (1 - p)p^{n-1}$ for $n \geq 1$ and $P(N_r = n) = (1-p)p^n$ for $n \geq 0$. The difference in $P(N = n)$ and $P(N_r = n)$ arises from the fact that N can only take values greater than or equal to 1 because we start from the assumption that r' loses the original transmission of packet 0. On the other hand, N_r can be zero when receiver

34

r receives the original transmission of packet k and thus does not need a retransmission of k. These two probabilities can be used to derive the following two relations,

$$
\begin{aligned}
P(N \le n) &= 1 - p^n, & n \ge 1 \\
P(N_r \le n) &= 1 - p^{(n+1)}, & n \ge 0
\end{aligned}
$$

Let Z_1 and Z_2 be two independent random variables. We have the following useful relations.

$$
\begin{aligned}
P(\min(Z_1, Z_2) \le z) &= 1 - (1 - P(Z_1 \le z))(1 - P(Z_2 \le z)) \\
P(\max(Z_1, Z_2) \le z) &= P(Z_1 \le z)P(Z_2 \le z)
\end{aligned}
$$

Based on these relations it is easy to derive

$$
\begin{aligned}
P(Z(k) \le z) &= 1 - p^z \left(1 - (1 - p^{z + \lfloor -k\Delta/\Delta' \rfloor + 1})^R \right), \quad k = -1, -2, \ldots \\
E[Z(k)] &= 1 - (1 - p^{\lfloor -k\Delta/\Delta' \rfloor + 1})^R \\
&\quad + \sum_{z=1}^{\infty} p^z (1 - (1 - p^{z + \lfloor -k\Delta/\Delta' \rfloor + 1})^R), \quad k = -1, -2, \ldots \quad (2.17)
\end{aligned}
$$

Using a similar approach, we can obtain the following expression for $E[Z(k)]$.

$$
E[Z(k)] = p^{\lfloor k\Delta/\Delta' \rfloor}(1 - (1-p)^R) + \sum_{z=1}^{\infty} p^{z + \lfloor k\Delta/\Delta' \rfloor}(1 - (1 - p^{z+1})^R), \quad k = 1, 2, \ldots \quad (2.18)
$$

By setting $R = 1$ in (2.17) and (2.18) we obtain

$$
E[Z'(k)] = \frac{p^{\lfloor |k|\Delta/\Delta' \rfloor + 1}}{1 - p^2}, \quad k = \pm 1, \pm 2, \ldots \quad (2.19)
$$

The expressions in (2.18) and (2.19) can be substituted into (2.14) to yield $E[U]$.
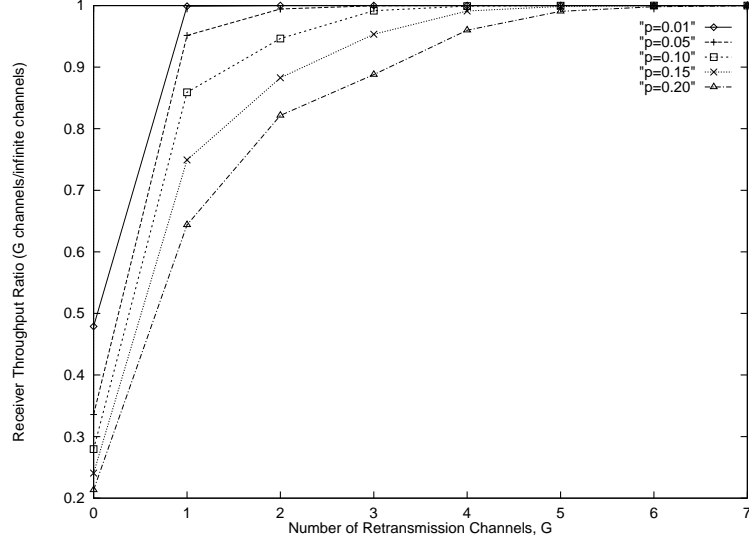
Figure 2.9. $\frac{\Delta}{\Delta'} = 1$ and $R = 1000$

2.5.2 Numerical Results

We now examine the number of retransmission channels required to achieve a receiver throughput "close" to the receiver throughput obtained by using an infinite number of retransmission channels. We also see how this number changes with loss probability p, the ratio Δ/Δ', and the number of receivers R. For this purpose, we define a performance metric γ, which is the ratio of receiver throughput[3] for the case of G retransmission channels ($G \geq 0$) to that for the case of an infinite number of channels. For P1, γ is defined as

$$\gamma = \frac{E[Y^{P1}]}{E[Y^{P1}] + pE[U]E[Y_p]}$$

Recall that $E[U]$ depends on the values of G, p, R and Δ/Δ' and thus γ depends on these variables too.

Figure 2.9 shows how γ varies with the number of retransmission channels, g, for several loss probabilities. In this figure $\Delta/\Delta' = 1$ and $R = 1000$. We find that as g increases,

---

[3]Receiver throughput is defined to be the inverse of receiver processing cost, $E[Y^{P1}]$.
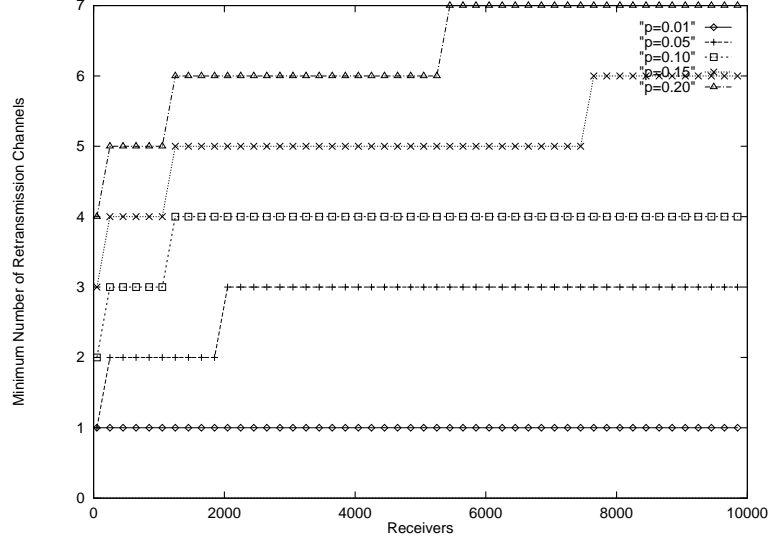
Figure 2.10. $G^*(0.99)$ vs R when $\frac{\Delta}{\Delta'} = 1$

γ approaches 1 very fast. This is because with more multicast channels, the separation (in time) between recovery of packets mapped to the same retransmission channel becomes larger causing $E[U]$, the mean number of unwanted packets, to fall very sharply to zero. Let $G^*(\epsilon)$ be the minimum number of retransmission channels required to make $\gamma > \epsilon$. We find that $G^*(0.99) = 3$ when $p = 0.10$ and $G^*(0.99) = 5$ when $p = 0.20$. Smaller values of epsilon reduce $G^*(\epsilon)$. We find that $G^*(0.90) = 2$ when $p = 0.10$ and $G^*(0.90) = 4$ when $p = 0.20$.

Figure 2.10 shows how $G^*(0.99)$ varies with the number of receivers for different loss probabilities, with $\Delta/\Delta' = 1$. As expected, $G^*(0.99)$ is an increasing function of the number of receivers. We observe however, that this growth is very slow. For $p = 0.20$, $G^*(0.99) = 5$ when $R = 1000$, $G^*(0.99) = 6$ when $R = 5000$ and $G^*(0.99) = 7$ when $R = 10000$. On the other hand, as the number of receivers decreases, the minimum number of retransmission channels does not drop considerably. $G^*(0.99) \geq 4$ when $p = 0.20$ even for 50 receivers.
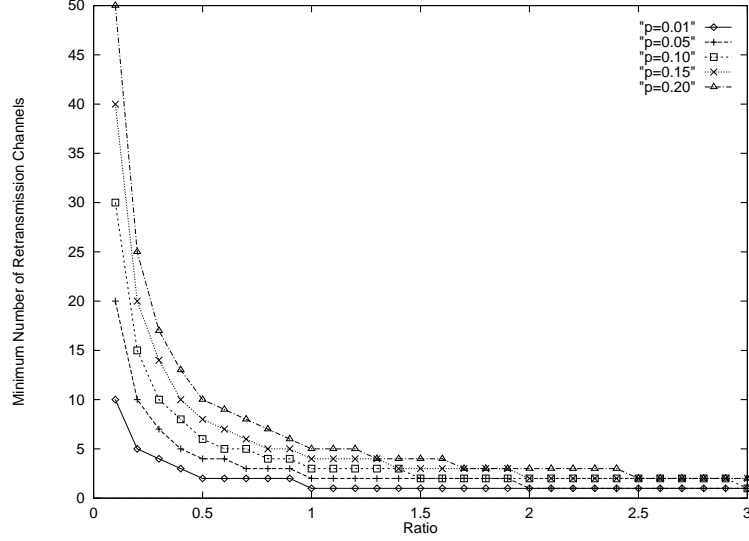
Figure 2.11. $G^*(0.99)$ vs $\frac{\Delta}{\Delta'}$ when $R = 1000$

These results suggest that only a small number of channels are needed to achieve a receiver throughput that is close to the the receiver throughput with $G = \infty$ even when the loss probability is high and the number of receivers is large. These results were obtained when the ratio Δ/Δ' is one. Next we observe how this ratio affects G^*.

Figure 2.11 shows the behavior of $G^*(0.99)$ as a function of Δ/Δ', for several loss probabilities, when $R = 1000$. We see that G^* is very sensitive to small values of Δ/Δ'. This is because the likelihood of overlap between retransmissions corresponding to the same retransmission channel increases as Δ/Δ' decreases. This behavior diminishes as Δ/Δ' increases. In fact, beyond a certain value of Δ/Δ', G^* becomes insensitive to Δ/Δ', as seen in Figure 2.11. Figure 2.11 also shows that for a given Δ/Δ', G^* is higher for higher loss probabilities. This increase can be attributed to the fact that more retransmissions are required to recover lost packets, thereby increasing the chance of overlap between recovery of packets mapped to the same retransmission channel. With an increase in Δ/Δ', G^* becomes insensitive to p. This is because the retransmissions of different packets using the same channel are significantly separated.

38

In summary, we observe that one can achieve a throughput within 1% of the maximum achievable throughput using a very small number of multicast channels and that this holds for a wide range of system parameters. If we choose a less stringent requirement and can tolerate unwanted processing slightly greater than 1% of the ideal case, then even fewer channels are required.

2.5.3 The Multiple Sender Case

In Section 2.5.1 we analyzed the mean number of unwanted packets at a receiver due to a finite number of multicast channels for the *one–to–many* model. We now extend the analysis of P1 to the *many–to–many* scenario, where each node is a receiver as well as a sender. Hence, instead of a single sender, we now have multiple senders. As before, there are G retransmission channels in addition to the channel for original transmissions. On losing a packet i from sender j, a receiver joins the multicast channel $A_{i \bmod G}$ and sends a NAK to sender j. Sender j retransmits the packet on $A_{i \bmod G}$. After correctly recovering packet i from sender j the receiver leaves $A_{i \bmod G}$.

For the convenience of analysis we model the multiple senders as a single global sender. That is, we assume that all of the transmissions and retransmissions originate from a fictitious global sender. We also assume that the global sender multicasts new packets periodically with a fixed time interval Δ, and retransmits a packet periodically with a fixed interval Δ' if there is a pending NAK for that packet[4]. A stream of original packet transmissions from the global sender would look like $(s_1, n_1), (s_2, n_2), (s_3, n_3), \ldots$ where s_i is the source of the ith packet and n_i is the sequence number given to that packet by sender s_i.

Let us assume that the sender's traffic streams are mutually independent. Hence a packet in the combined packet stream from the global sender is equally likely to be mapped to any of the retransmission channels. In other words, the probability that a packet corresponding

---

[4]We have made this assumption for simplifying the analysis. Our goal is to obtain an estimate of the number of multicast groups required in the multiple sender case without complicating the analysis. The combined stream of packets from several senders is unlikely to have a constant inter–packet interval.
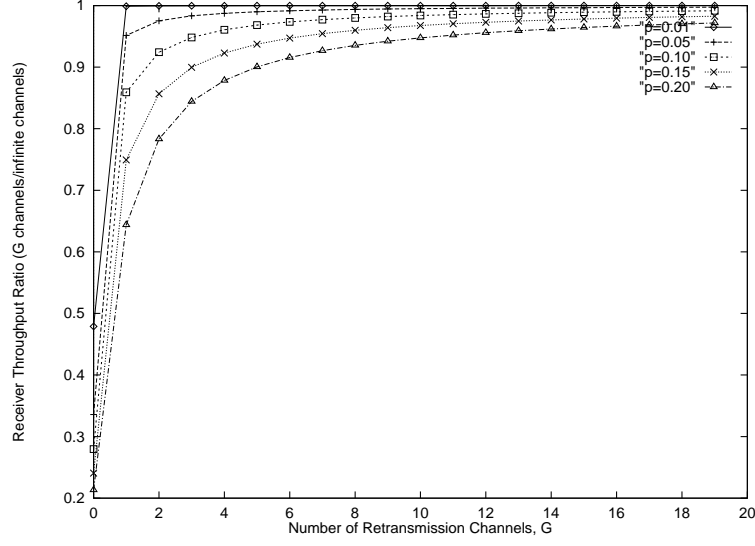
Figure 2.12. Many–Many Scenario, $\frac{\Delta}{\Delta'} = 1$ and $R = 1000$

to a certain retransmission channel is $1/G$. As before, we randomly choose a packet from the global stream and call it 0. The expected number of unwanted packets received at a randomly chosen receiver, say r', when it tries to recover packet 0 can be expressed as

$$E[U] = (1-p)\left(\frac{1}{G}(\sum_{k=1}^{\infty}(E[Z(k)] + E[Z(-k)]) - \sum_{k=1}^{\infty}(E[Z'(k)] + E[Z'(-k)]))\right) \quad (2.20)$$

Using equations (2.17)–(2.20) we can compute the value of $E[U]$ for different values of G, p, R and Δ/Δ'. We compute γ, as defined in Section 2.5.2, for different values of G. Figure 2.12 shows how γ varies with G when $\Delta/\Delta' = 1$ and $R = 1000$. We observe that γ rapidly increases when G is small, but then approaches a horizontal asymptote of 1 as G increases. We define $G_m^*(\epsilon)$ to be the minimum number of retransmission channels required to keep $\gamma > \epsilon$. Due to the asymptotic behavior of γ, choosing values of ϵ very close to 1 results in large $G_m^*(\epsilon)$. For example, $G_m^*(0.99) = 17$ when $R = 1000$ and $p = 0.10$. Choosing a somewhat smaller value of ϵ is more practical. For example, $G_m^*(0.90) = 2$ when $R = 1000$ and $p = 0.10$. This means that with 2 retransmission channels we could

40
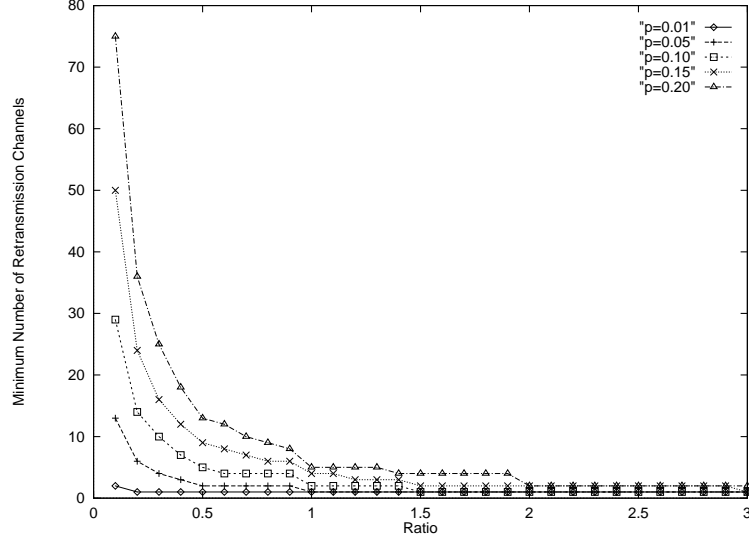
Figure 2.13. Many–Many Scenario, $G_m^*(0.90)$ vs $\frac{\Delta}{\Delta'}$ when $R = 1000$

get 90% of the benefit that we would have gained with an infinite number of retransmission channels.

We now study how the ratio Δ/Δ' affects $G_m^*(\epsilon)$. Figure 2.13 shows how $G_m^*(0.90)$ varies with Δ/Δ' for several loss probabilities when $R = 1000$. We observe from Figure 2.13 that $G_m^*(0.90)$ takes moderate to low values for a wide range of values of Δ/Δ'. Thus, even in the presence of multiple senders, we can obtain 90% of the benefit of infinite channels by using only a moderate or small number of channels. However, now the receivers must be prepared to tolerate a slightly larger number of unwanted packets than in the *one–to–many* case.

We end this section by observing that in a pathological case, all of the traffic streams from the senders might synchronize in transmitting packets such that packets with the same sequence number get bunched together. If $A_{i \bmod G}$ is the retransmission channel of packet i irrespective of the sender, then a bunch of $1s$ (or $2s$ or $3s \ldots$), from different senders, would correspond to the same retransmission channel and there is likely to be a great deal of overlap in their recovery. This would result in a large number of unwanted packets at the

41

receivers involved in recovering these packets. To reduce the likelihood of such a synchronization we could use different rules for mapping packets to retransmission channels. An example of one such rule is that each sender uses the retransmission channels in a different order, i.e., if there are, say, 6 retransmission channels then sender 1 might use the retransmission channels in the order $[1, 4, 3, 0, 5, 2]$ and sender 2 might use a different order such as $[5, 3, 2, 0, 4, 1]$ and so on. This means that sender 1 retransmits packet 0 on channel 1, packet 2, on channel 4, packet 3 on channel 3 and so on. The receivers know the order of retransmission channel usage for each sender and hence join the appropriate channel, depending on the sender, for recovering packet i, rather than joining $A_{i \bmod G}$ for all the senders.

2.6 Implementation Mechanisms

We now look at mechanisms to implement scalable reliable multicast with end–to–end recovery using multiple multicast channels. The use of multiple retransmission channels reduces receiver processing costs. By allowing the forwarding of retransmissions to only those branches leading to receivers who need the retransmitted packets, network bandwidth consumption is also reduced. The key implementation issue is how to efficiently manage the multiple channels. We consider two different implementation options, one using existing IP multicast mechanisms, the other using the control–on–demand active networking architecture [25].

2.6.1 Using IP Multicast Groups

One mechanism to implement the required multicast channels is to use multiple IP multicast groups. Here, each multicast channel is implemented as an IP multicast group, and joining and leaving a multicast channel corresponds to joining and leaving an IP multicast group (using the IGMP protocol). The basic scheme is simple. Whenever a packet is lost, the receiver determines the retransmission group on which the lost packet will be retrans-

42

mitted, sends a join request for that multicast group (unless it is already a member of that retransmission group), and then generates and sends a NAK to the sender. When all losses associated with a particular group have been successfully recovered, the receiver leaves the retransmission group (recall, due to the sharing of a finite number of groups to recover multiple packets the receiver may be waiting for multiple packets on the same group).

Even though the number of multicast groups is likely to be relatively small, there are potential concerns with using IP multicast groups. First, there are overheads for processing join and leave operations at routers, as receivers dynamically add and delete themselves from multicast groups (i.e., generation, forwarding and processing of join and leave signaling messages). Second, join and leave messages (reliable unicast transmissions between routers) use some additional bandwidth. In current networks supporting IP multicast routing, a join or leave from a receiver is detected by the nearest multicast router, the one attached to the subnet of the receiver, through the IGMP protocol [21]. This information is then propagated to the nearest branching point of the multicast tree, rooted at the sender [19], or to a suitable core, in the case of core–based trees [4]. Each join and leave message results in processing overhead at all the intermediate routers. The significance of this processing burden is highly topology dependent. In general, the closer the branching point is to a receiver, the lower the join or leave overhead on the network.

Another problem with using IP multicast is the high leave latency. When a receiver sends a leave request to its subnet router, the router usually waits for a few seconds [65, 74] before it actually "leaves" the group and sends a leave message upstream. This means that, even after a receiver has left a multicast group, the subnet of the receiver continues to receive packets sent to that multicast group. Therefore, a leave operation at a receiver can only save receiver processing but not network bandwidth. When the join latency is high, it is also possible that a receiver will join the same multicast group for recovering a different packet even before the subnet multicast router leaves the group for an earlier packet. If this happens frequently, then, as far as the multicast router is concerned a receiver

never leaves any retransmission multicast group and the network bandwidth consumption becomes the same as in the case of using a single multicast group for both transmissions and retransmissions.

High join latencies can also lead to inefficient performance. The path along which join messages traverse is slower[5] than the path along which NAKs propagate towards the sender. If the join latency for a multicast group is high, the multicast route setup can take longer than the time required by the NAK to reach the sender and generate a retransmission from the sender. This could result in the loss of the retransmission.

In order to minimize the impact of signaling processing due to join and leave operations, and to be able to use IP multicast, we have designed a scheme that does *local filtering* at a receivers network interface. This introduces no additional signaling (i.e., join/leave packets to be sent to/from routers) in the network. Instead, all packets belonging to both the original transmission and retransmission groups are always allowed to reach the local network to which the receiver is attached. All unwanted packets are then filtered out by the network interface hardware at the receiver.

In our scheme we distinguish between two kinds of join/leave operations. The first kind, which we call non–local join and non–local leave, is the regular join and leave as described above. The second kind, which we call local join and local leave, concerns only the receiver's local network interface. A local join or leave message, from the reliable multicast protocol layer, travels only to the receiver's network interface hardware. No IGMP messages are sent to the nearest IP multicast router. A local join or leave is simply an indication to the hosts network interface to filter packets locally.

We now describe the scheme for local filtering. As a first step, a receiver non–locally joins both the original transmission and all of the retransmission multicast groups. This

---

[5]The join messages are sent unicast between hops. They are processed by each router for the purpose of updating the multicast routing table. On the other hand, the NAKs are sent directly to the sender's address and are forwarded by the fast switching fabric of the routers.

results in the transmission of IGMP messages to the nearest IP multicast router and, subsequently, the propagation of graft messages towards the branching point. Subsequently, the receiver locally leaves all of the retransmission groups. From this point on, whenever the receiver needs to recover a packet it performs a *local* join to the appropriate multicast group. Once the packet has been received correctly, the receiver performs a local leave. Hence, as far as the network routing tables are concerned the receiver is always a member of all the retransmission groups and all packets to these groups are duly forwarded to the local interface of the receiver. It is left to the local interface to filter out the unwanted packets, based on the information provided by the local join and leave operations, to save the receiver from processing these packets. When a receiver wants to drop out of the multicast session, it non–locally leaves all of the multicast groups.

Since the filtering is done locally at a receiver, this scheme does not reduce the data traffic in the network, and hence does not reduce network bandwidth consumption. At the same time it does not introduce any additional traffic in comparison to the scenarios that use a single multicast group.

Note that our local filtering scheme does not require any changes to the network routers. There is, however, a need to modify the networking code at the receiver. This change appears not to be excessive. It is clear that for this scheme to work, the network interface hardware at a receiver must provide support for filtering. The Ethernet interface provides multicast filtering in the hardware[6]. Although it is best to perform the local filtering in the network interface hardware, benefits are also possible by implementing an efficient software packet filter. Finally, it should be noted that even with local filtering it is still necessary for the routers to set up routes and include entries in the multicast routing table for

---

[6]IP multicast addresses are mapped to Ethernet multicast addresses. Filtering is done based on Ethernet addresses and not IP multicast address. Such a filtering is desirable in the Ethernet hardware according to [18]. Unfortunately, many Ethernet cards do not provide appropriate multicast filtering.

each of the multicast groups. We still need multiple IP multicast addresses to be allocated to each reliable multicast session.

2.6.2 Using Active Networking

Local filtering reduces the processing requirements at the receiver end–system by moving the filtering from the host's CPU to its network interface card. However as we have seen, it does not save on bandwidth. Moving filtering to a point inside the network does not adequately economize on network bandwidth either, because of the high leave latency associated with this move.

In order to reduce signaling overhead and speed up join/leave latency, we propose using an active networking approach. The control–on–demand architecture proposed in [25] supports router programmability while retaining basic IP forwarding. While having the capability of acting in the data path (processing every data packet), control–on–demand also supports control plane programmability where the user installed program manages connectivity and router resources without interfering with data forwarding. We use the stream thinning capability of the control–on–demand architecture at an active router to actively discard data on unwanted branches. This active discard mechanism implements multiple (potentially infinite) retransmission channels.

With this approach, all transmissions and retransmissions are sent to the same IP multicast group. Active routers along the path from the receivers to the sender process NAKs sent from the receivers to the sender and maintain state for selective forwarding of subsequent data retransmissions from the sender[7]. The active routers "snoop" retransmissions from the sender and, based on the NAK state, attempt to stop the forwarding of the retransmissions on links that do not lead to receivers desiring the retransmission. The snooping mechanism is executed asynchronously of data forwarding and, hence, does not reduce data

---

[7]The reverse path for sending NAKs from the receivers to the sender through the active routers could be established by using Source path messages (SPMs) as shown in Chapter 5

forwarding performance. At the same time, snooping provides only a probabilistic filtering of undesired retransmissions. There is no guarantee that a retransmission could be halted from being forwarded even after it has been snooped.

In the ideal case, a retransmission is forwarded only on links leading to receivers that have requested the retransmission. This is equivalent to recovering every lost packet on a separate channel. Only one retransmission is sent on a branch for each NAK. After a retransmission corresponding to a NAK is forwarded downstream, the active node removes all the state associated with that NAK so as to conserve network resources. This also avoids the need for an explicit receiver leave operation. An interesting consequence of the above data–driven management of NAK state is that it allows the network to exploit our mechanism "under the hood" transparently to the receivers. The receivers would see improved service quality as mostly only wanted packets would be delivered.

Instead of using snooping, one could use IP options in retransmissions (as suggested in Chapter 5) to get the attention of the active node for selective forwarding of retransmissions. This approach has been used in PGM [68]. The analysis presented in this chapter, with minor modifications, is also applicable to PGM. The problem in using IP options is that retransmissions with IP options move along a slower path instead of the fast data forwarding path, even though the filtering is ideal.

2.6.2.1 Performance Benefits

We have observed above that implementing multiple multicast groups using active networking will improve performance by reducing bandwidth consumption. We now present some numerical examples to substantiate this observation. We will assume that active nodes perform ideal filtering.

Our bandwidth measure, termed *bandwidth consumption*, is defined to be the sum of the expected number of transmissions (including retransmissions) per successful transmission of a packet, along all of the links to all of the receivers. For simplicity, we assume that

47
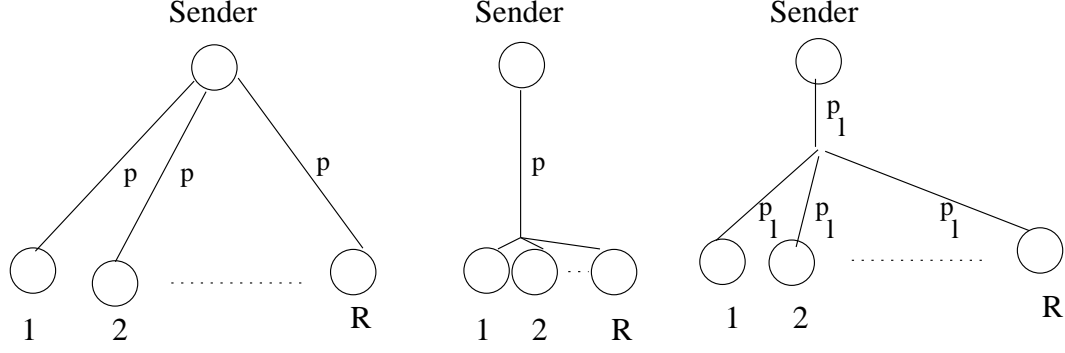
Figure 2.14. Topologies

all receivers are identical and suffer losses with probability p. Bandwidth consumption is dependent upon the multicast tree topology. We consider three topologies as shown in Figure 2.14. In the first topology, a star, receivers have disjoint paths from the sender. In the second topology, all receivers share the same path from the sender. In the third topology, a modified star [79], a portion of the path is common to all receivers. The loss probability along the common path and each of the disjoint paths in the modified star is denoted by p_l, where $p = 1 - (1 - p_l)^2$.

We now determine the ratio of the bandwidth consumed by an active approach for implementing multiple multicast channels to the bandwidth consumed when only a single multicast channel is used for both transmissions and retransmissions. For the star topology the bandwidth consumption ratio can be expressed as $E[M_r]/E[M]$, where $E[M_r]$ is the mean number of sender transmissions required for correctly transmitting a packet to one receiver and $E[M]$ is the mean number of sender transmissions required for all receivers to correctly receive a packet. From the expressions for $E[M_r]$ and $E[M]$ derived in Section 2.3, we have

$$E[M_r]/E[M] \;=\; 1/(1-p)(1 + \sum_{m=1}^{\infty} (1 - (1 - p^m)^R))$$

48

For the second topology, since all receivers have the same path from the sender, the bandwidth consumption ratio is $E[M_r]/E[M_r] = 1$. For the modified star, the bandwidth consumption ratio is expressed as

$$(E[M] + E[M_r']R)/(E[M] + E[M](1 - p_l)R)$$

where $E[M_r'] = 1/(1 - p_l)$ is the mean number of transmissions required by a receiver to recover loss on one link (with loss probability p_l). Now, $E[M]$ can be expressed as follows:

$$E[M] = (1/(1 - p_l))(1 + \sum_{m=1}^{\infty} (1 - (1 - p_l^m)^R))$$

Figure 2.15 shows the reduction in bandwidth consumption due to the use of an infinite number of multicast channels over a single multicast channel as the number of receivers increases. Here $p = 0.1$. The star topology and the identical–path topology are the two extreme cases. The reduction in bandwidth consumption for other topologies will lie in between these two. The reduction in bandwidth consumption of the modified star topology is shown as an example.

In addition to reducing bandwidth consumption, the implementation of multiple multicast channels further improves the end–host processing costs. The receiver processing cost is the one derived in Section 2.3 for infinite groups minus the join and leave processing costs, as no explicit join or leave operations are required. As the active networking architecture provides NAK suppression, there is no need for explicit NAK suppression. This means that ideally, for each packet, the sender would receive only as many NAKs as the number of transmissions required by the sender to successfully transmit the packet to all receivers. Thus the sender throughput of P1 increases to that of P2.

2.6.2.2   Memory Requirement For Storing NAK State

We now estimate the memory required for maintaining NAK state at an active node for a multicast session. We again assume that the sender multicasts new packets periodically
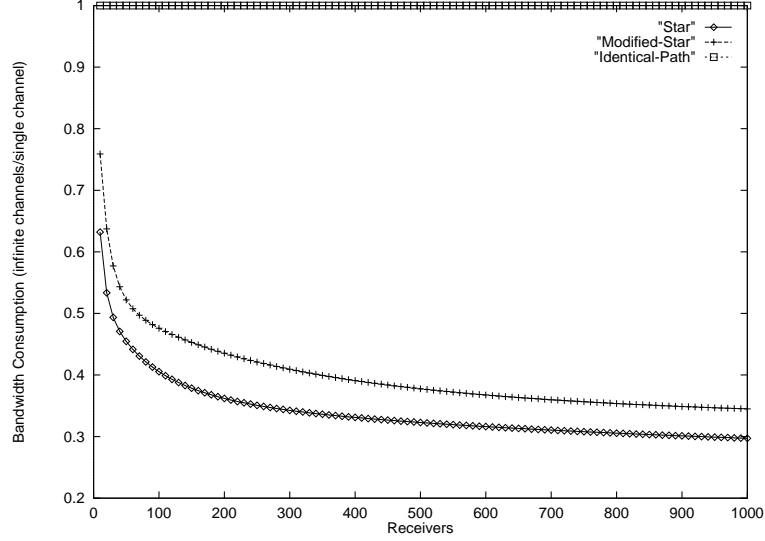
Figure 2.15. Bandwidth Consumption Reduction

with a fixed time interval Δ, and retransmits a packet periodically with a fixed interval Δ' as long as there is a pending NAK for that packet, as in Section 6. We focus on the memory requirements of an active subnet router. Let there be B buffers at this active node, each capable of holding a single NAK state. The probability of loss of a packet between the sender and the active node is denoted by p. Loss events are assumed to be independent. We assume that there are no losses between the active node (subnet router) and the receivers below it [79].

First–time–NAKs (not the retransmitted NAKS) arrive with a time interval Δ/p at the active node[8]. A NAK state is kept from the time the first NAK is received by the active node until the time the packet corresponding to the NAK is forwarded downstream. Due to limited buffer space, it is possible that no buffers will be available when a new NAK arrives. We assume that an *oldest first* buffer replacement policy is used in such a case, i.e., that new NAK state replaces the oldest NAK state. Consider a random lost packet, i. Let q denote the probability of replacing the NAK state corresponding to i, even before i

---

[8]Realistically, NAKs would arrive at random time intervals.

50

is received, by a new NAK state. The maximum number of retransmissions of i before the NAK state for i is overwritten is equal to $\lfloor B\Delta/\Delta' \rfloor$. If N is the number of retransmissions of i from the sender such that it is correctly received by the active node, q is equal to the probability of N exceeding $\lfloor B\Delta/\Delta' \rfloor$. Therefore,

$$
\begin{aligned}
q & = P(N > \lfloor B(\Delta/\Delta')/p \rfloor) \\
& = p^{\lfloor B(\Delta/\Delta')/p \rfloor}
\end{aligned}
$$

We now determine the minimum number of NAK state buffers required to ensure $q \leq 0.001$. Figure 2.16 shows how the minimum NAK state buffer requirement varies with the ratio Δ/Δ' for different values of p. We observe that the required number of NAK state buffers increases in p and decreases in Δ/Δ'. Fewer than 10 buffers are required to ensure $q \leq 0.001$ for a wide range of loss probabilities and ratios Δ/Δ'. Therefore, the active node needs only 10 buffers or less for keeping NAK state per session. Considering that the NAK state comprises of sender address (4 bytes), multicast group address (4bytes), packet sequence number (4 bytes), NAK sequence number (1 byte) required for NAK suppression (see [43]) and another 7 bytes for interface numbers (on which the retransmission needs to be forwarded), flags and any other fields, a buffer sufficient to store the state of 10 NAKs amounts to approximately 200 bytes per multicast session. For small loss probabilities or high Δ/Δ', this memory requirement is much less.

On the rare occasion that the NAK state for a packet at an active node is overwritten, a retransmission is not forwarded downstream. The downstream receiver will eventually timeout and retransmit the NAK which will create a new NAK state at the active node.

In this section we have considered NAK state buffer requirement at active subnet routers. The NAK state required by active routers inside the network per multicast session is likely to be of the same order, though the number of multicast sessions the active router has to support is larger.

51
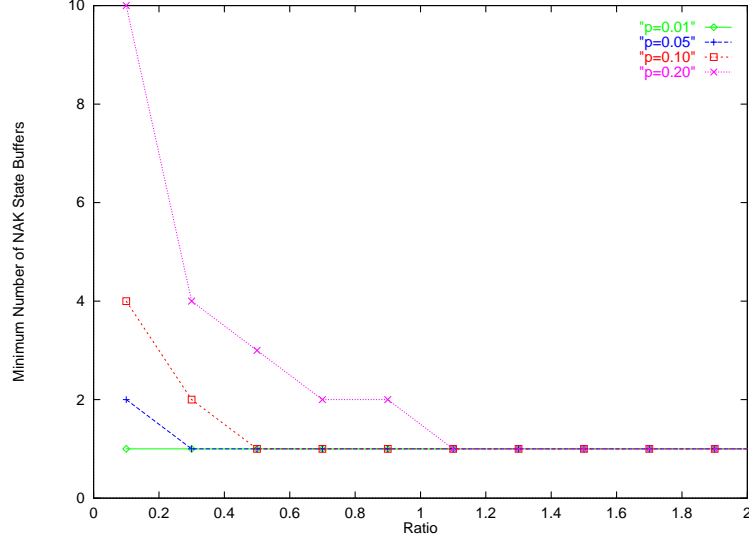
Figure 2.16. NAK State Buffer Requirement

2.7 Conclusions

In this chapter we have examined an approach for providing retransmission scoping for reliable, scalable multicast communication by using multiple multicast channels for recovery of lost packets. In this approach, rather than having all receivers receive all retransmitted packets (regardless of whether a given receiver had already received a given packet correctly), the multiple multicast channels are used to allow only those receivers that actually *want* a particular packet to actually receive that packet.

We considered the idealized case of an infinite number of multicast channels as well as the more realistic scenario of using a small, fixed number of multicast channels. We also considered two different models of sender behavior: the one–to–many scenario and the many–to–many scenario. Our analytic models have demonstrated that significant performance gains (in terms of reduced receiver overhead, and a reduced overall protocol overhead in the case of many–to–many communication) can be realized in such environments, over a range of system parameters.

We discussed two mechanisms for implementing multiple multicast channels, one using multiple IP multicast groups and the other using an active networking approach. With the

current IP multicast support we obtain savings in receiver processing through our local filtering scheme but cannot save on network bandwidth whether or not we move the filtering point inside the network. However, with the active networking approach, by using some processing at routers and a small buffer space (up to 200 bytes per multicast session), we could save on both sender and receiver processing costs as well as network bandwidth.

Our work can proceed in the following directions. Our analyses assume that loss events are independent. In [79] and [80] Yajnik *et al.* have observed temporal correlation in losses on the Internet. They have also observed some spatial correlation in loss on the MBone. The framework developed in [5] could be used to extend our analysis to account for spatial correlation in loss. With more spatially correlated loss among receivers, the benefit of using multiple multicast channels will be less. Indeed, if the losses were 100% spatially correlated, no receivers will receive any unwanted packets even with a single multicast channel. In the future, we plan to study the impact of temporal correlation in loss, especially in the context of reusing a finite number of multicast channels.

It is possible to combine our work with existing work on local recovery. Local recovery helps in isolating the domains of loss, thereby reducing global retransmissions and recovery latencies. If there are several local domains separated by a wide area network, and the domains see a sufficient amount of independent loss, then multiple multicast channels could be used to scope retransmissions to the domains. If the domains themselves have a large number of receivers experiencing high intra–domain losses then multiple multicast channels could also be used for retransmission scoping inside these domains.

In order to be able to use IP multicast for implementing multiple multicast channels we need to clearly understand the processing overheads of join and leave signaling on the routers, determine actual values of join and leave latencies (through actual measurements) and find approaches for reducing leave latency [65]. As far as the active networking approach is concerned, we need to model the processing costs at active routers. We also need

to study the impact of multiple multicast sessions, using an active router at the same time, in terms of its processing and buffering resource requirements.

We have seen that multiple multicast channels can be used to reduce receiver processing costs and network bandwidth consumption. As far the sender is concerned, there is no improvement in processing costs whether we use a single channel or multiple channels. If the sender is the bottleneck, as observed in [71], then there is no improvement in protocol throughput by using multiple channels for the *one–to–many* case. The next chapter shows how local recovery can be used to address this problem.

CHAPTER 3

A COMPARISON OF SERVER–BASED AND RECEIVER–BASED LOCAL RECOVERY APPROACHES FOR SCALABLE RELIABLE MULTICAST

Local recovery approaches for reliable multicast, in which a network entity other than the sender aids in error recovery, have the potential to provide significant performance gains in terms of reduced bandwidth and delay, and higher system throughput. In this chapter, we propose a new server–based local recovery approach that requires additional processing and caching inside the network and compare its performance with two traditional receiver–based local recovery approaches that use only traditional end–to–end means. Our server–based local recovery makes use of specially designated hosts, called *repair servers*, co–located with routers at strategic points inside the network. Each repair server serves the retransmission requests of receivers in its local area by (re)transmitting local repairs. The repair servers themselves recover lost packets from either higher–level repair servers or the sender.

Receiver–based approaches do not require any special repair servers; only the end hosts (sender and receivers) are involved in error recovery. The first receiver–based approach is *self organizing*. Whenever local recovery of a lost packet is attempted, the receiver needing the repair dynamically selects a receiver within a local geographical area to supply the repair. The second receiver–based approach, called the *designated receiver* approach, assigns a specific end–receiver the task of supplying repairs in its local area.

We analyze the end–system throughput and the network bandwidth usage of the above local recovery approaches in the face of spatially correlated loss. We first show that local recovery approaches yield significantly higher end–system throughput and lower bandwidth

55

usage than an approach that does not use local recovery. Next we compare the performance of the three local recovery approaches. Under the assumption that repair servers have sufficiently high processing and buffering capabilities so that they themselves are not bottlenecks, we show that server–based recovery yields higher end–system throughput and lower bandwidth usage than receiver–based recovery. Among the two receiver–based approaches, the self–organizing approach exhibits higher end–system throughput but uses more bandwidth.

Last, we estimate the processing power needed at repair servers so that they do not become bottlenecks. We find that a repair server needs a processing power slightly higher than that of a receiver, per multicast session, for a wide range of loss probabilities.

The remainder of this chapter is structured as follows. In the following section we examine the existing work on local recovery for reliable multicast. In Section 3.2 we present the three local recovery approaches and our system model in Section 3.3. Section 3.4 contains the throughput and bandwidth analysis. We compare the performance of the server–based and the receiver–based approaches in Section 3.5. In Section 3.6 we find the processing power required at the repair servers. Conclusions and directions for future work are contained in Section 3.7.

3.1 Related Work

Many researchers have recently proposed different approaches for local recovery. These approaches can be broadly classified as follows. Log–based reliable multicast, LBRM [31], is server–based, Scalable reliable multicast [22, 23], SRM, with local recovery enhancements is self organizing receiver–based and Reliable multicast transport protocol [47], RMTP, Tree–based multicast transport protocol [78], TMTP, and LORAX [45] are designated receiver–based with pre–constructed logical hierarchy. Local group concept [29, 30], LGC, is a hybrid of the logical tree–based approach and SRM. We add a new dimension

to the earlier work on server–based recovery [31] by considering the placement of repair services inside the network.

Our work is the first to analytically compare the throughput and bandwidth usage of server–based and receiver–based local recovery approaches. Previous throughput analyses have focused on comparing ACK–based versus negative acknowledgment (NAK)–based protocols that do not use local recovery [36, 61, 71] or comparing logical tree–based local recovery with other non–local recovery approaches [44, 45]. In our work, we analyze not only the throughput but also the network bandwidth usage of the local recovery protocols. We also estimate the processing and buffering requirements of the repair servers. All of the previous work mentioned above also assumes spatial independence among losses; we use a system model that considers the spatial correlation [79] present in network losses.

3.2 Protocols

We now present a server–based and two receiver–based approaches to local recovery for reliable transmission of data from a sender to multiple receivers. All approaches are receiver–initiated NAK–based approaches. The server–based approach places much of the burden for ensuring error recovery on repair servers placed inside the network. In the self organizing receiver–based approach, the error recovery burden is shared among the sender and the participating receivers. In the designated receiver approach, a designated receiver is assigned the task of supplying repairs in its local area. In this section we assume that mechanisms are available for scoping retransmissions within a "local" region. Some of the mechanisms for scoping retransmissions are described in Chapters 2 and 5.

3.2.1 A Server–Based Local Recovery Protocol

We now describe a generic server–based reliable multicast local recovery protocol, L1, that assumes the presence of a *repair tree*. This repair tree is the physical multicast routing tree constructed by the routing protocols with the sender as the root, receivers as leaves

and repair servers co–located with routers. The receivers recover lost packets from repair servers and repair servers recover lost packets from either upper level repair servers or the sender. Protocol L1 exhibits the following behavior:

- The sender multicasts packets to a multicast address (say A) that is subscribed by all receivers and repair servers.

- On detecting a loss, a receiver, after waiting for a random amount of time, multicasts[1] a NAK to the receivers in its neighborhood and the repair server (for the purpose of NAK suppression among receivers in the neighborhood) and starts a NAK retransmission timer. If the receiver receives a NAK from another receiver for this packet, it suppresses its own NAK and sets the NAK retransmission timer as if it had sent the NAK.

- On detecting a loss, a repair server, after waiting for a random amount of time, multicasts a NAK to the sender (or its upstream repair server) and the other repair servers at the same level in the tree (for the purpose of NAK suppression among repair servers) and starts a NAK retransmission timer. While waiting to send out a NAK for the lost packet, if the repair server receives a NAK for the same packet from another repair server, then it suppresses its own NAK and sets the NAK retransmission timer as if it had sent the NAK.

- If a repair server has the packet for which it received a NAK from a member of its group (of receivers) for which it is responsible, it multicasts the packet to the group. If the repair server does not have the packet, it starts the process of obtaining the packet from the sender (or its upstream repair server) if it has not already done so (as described above).

---

[1]Assume that this multicast address, different from A, is known to the receiver.

58

- The sender, on receiving a NAK from the repair servers, remulticasts the requested packet to all receivers and repair servers.

- The expiration of the NAK retransmission timer at a repair server (or a receiver) without prior reception of the corresponding packet serves as the detection of a lost packet for the repair server (or the receiver) and a NAK is retransmitted.

We end this section by distinguishing between a log service and a repair service. A log service, as mentioned in [31], provides secondary storage for packets transmitted from the sender. The entire data is stored at the log servers for repairs and "late–comers." Log service should be distinguished from the repair service, particularly if the repair servers are considered network resources that will be shared over several multicast sessions. Logging entire multicast sessions at these repair servers may not scale with the number of multicast sessions because of the storage requirement. In our work, we consider storing only "recent" data at the repair servers for the sole purpose of providing quick repairs.

3.2.2   Receiver–Based Local Recovery Protocols

We now describe the two receiver–based local recovery protocols. We first describe the self organizing receiver–based protocol for local recovery. This protocol, termed L2, is a generic version of SRM [22] with local recovery enhancements. In L2, loss recovery is performed at two levels. At the first level, a receiver tries to recover packets locally from only within its local neighborhood. We define[2], a receiver's local neighborhood to consist of all the receivers in the subtree (of the multicast routing tree) emanating from its nearest backbone router at the edge of the backbone. If the receiver is unsuccessful in recovering the packet locally, it tries to recover packets from the sender ("global recovery"). Each receiver alternates between local and global recovery until it receives the missing packet. Protocol L2 exhibits the following behavior

---

[2]This definition and other definitions of local neighborhood can be found in [23].

- the sender multicasts all packets on an address (A) which is subscribed by all participating receivers.

- on detecting a loss, a receiver waits a random amount of time and multicasts a local NAK addressed only to the receivers in its neighborhood and starts a local NAK retransmission timer (first–level recovery); if the receiver does not receive the lost packet before the local NAK retransmission timer expires, it waits for a random amount of time and multicasts a global NAK to address A and starts a global NAK retransmission timer (global recovery). The random waiting times and the NAK retransmission timeout values are different for the two levels of recovery. The first–level retransmission timeout will be smaller. If the receiver does not recover the lost packet before the global NAK retransmission timer expires, it reinitiates local recovery.

- on receiving a local NAK for which it has the associated packet, a receiver multicasts the packet only to its neighborhood; before sending this repair packet, a receiver waits for a certain random amount of time and suppresses its own transmission if another receiver sends out the repair packet in this time. On receiving a global NAK for a packet, a receiver suppresses its own NAK if it is planning to send one for the same packet.

- upon receiving a global NAK, the sender remulticasts the packet to A.

The above protocol attempts to ensure that only one NAK and only one repair are generated at either recovery level. Unlike SRM [22], L2 generates *global repairs only from the sender* thereby avoiding the risk of global repair explosion. In SRM, receivers and the sender use a random back–off strategy to generate a repair. Here, all receivers and the sender " listen" for a repair. If they do not receive the repair within a certain time, then they generate a repair if they have it. Note that a repair has to be processed at each node whether it is sent or received. Hence the overhead of processing a global repair under SRM and

L2 will be comparable. For local recovery, no particular host is likely to have the repairs and hence, as in SRM, L2 uses a random back–off strategy for generating local repairs. Protocol L2 has been described above in the case of two–level recovery. It could be easily generalized to multiple levels of recovery.

The designated receiver local recovery protocol, termed L3, is similar to protocol H2 described in [45] with some minor differences. The repair tree in L3 (and H2) is a logical tree [44, 45] constructed with end hosts (the sender and the receivers). This logical repair tree is different from the physical multicast routing tree. Instead of special repair server hosts, each non–leaf receiver is assigned the task of providing repair service to its children. In protocol L3, the sender multicasts the original transmissions of a packet to all the receivers but unlike L1, it multicasts retransmissions only to its children in the logical tree. This could be done by using a separate multicast group to which all of the designated receivers and the sender belong. The NAKs from designated receivers are also sent on this group for the purpose of NAK suppression. Each designated receiver recovers lost packets from its parent and supplies repairs to its children in the logical tree. In L3 (unlike H2) a node does not send any periodic acknowledgments to its parent.

3.3   System Model

We now present the network loss and system model that will be used for the performance analysis of protocols L1, L2 and L3. In [79], Yajnik *et al.* have observed that most of the losses take place in the links from the source site to the backbone (we call this the source link) and in the links (called "tail links") from the backbone to the individual sites (termed stub domains in [10]), as shown in Figure 3.1. The backbone and the individual sites have been observed to be mostly loss free. It has also been noted in [31] that tail links are likely to be bottlenecks for the foreseeable future. Hence, in our loss model we consider loss only at the source and tail links. Losses at the source link will be seen by all receivers, whereas losses at a tail link will be seen only by the receivers attached to that tail

Source

p_1 Source link

loss free backbone

Backbone links

Tail links

p_1

site

receivers
in each tail link
(no loss within a site)

local neighborhood

■   Backbone Router
⊖   Source
○   Receiver
⬤   Repair Server
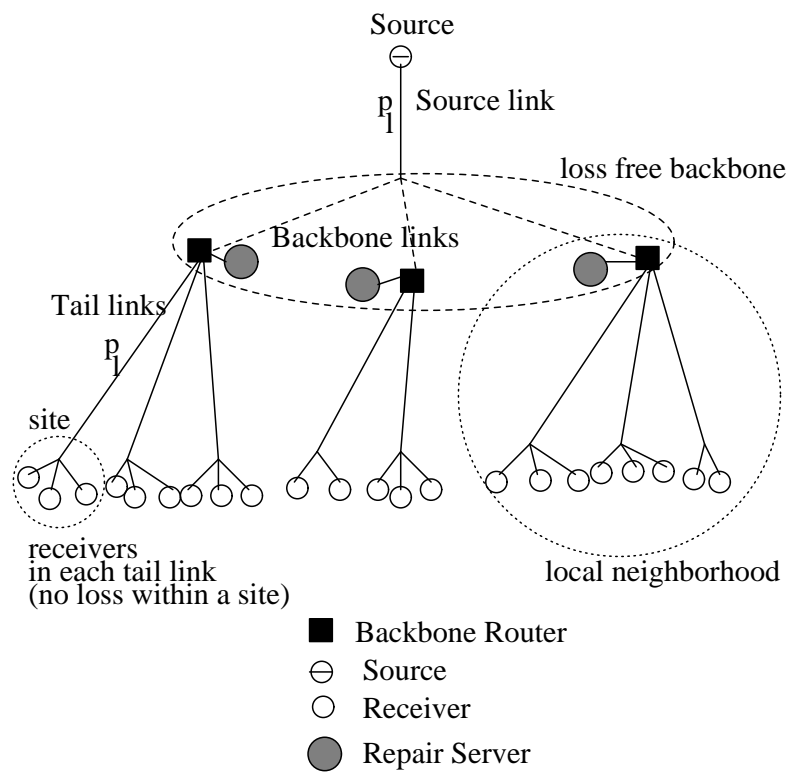
Figure 3.1. System Model

link. Losses at tail links are assumed to be mutually independent. We also assume that loss events are temporally independent. Let the loss probability in the source link and each tail link be p_l. The end–to–end loss probability as seen by a receiver, p, is equal to $1 - (1 - p_l)^2$. We also assume that NAKs are never lost. This assumption can be relaxed by following the analysis given in [72].

We assume that well–defined local neighborhoods exist and that the mechanisms described at the end of the previous section could be used to address packets to specific neighborhoods. Let there be one source link and T tail links where each tail link has multiple receivers. Let each upstream backbone router, at the edge of the backbone, have k tail links. The number of such backbone routers, W is equal to T/k. We assume that communication between two receivers on different tail links takes place only through a backbone router(s). Direct site–site links (termed stub–stub edges in [10]) are likely to be very rare.

The above loss model has implications for the placement of repair servers inside the network. We argue that the repair servers should be placed with the routers at the edge of the backbone. To be able to take advantage of local recovery, a repair server should be both upstream of, and as close to, the point of loss as possible. Figure 3.1 shows such a placement of repair servers.

We end this section by noting that for our system model, two–level recovery is sufficient for protocol L1. In analyzing L3, we assume that there is a two–level hierarchy where one receiver in each neighborhood is assigned the task of supplying repairs within the neighborhood. These designated receivers recover lost packets directly from the sender.

3.4 Performance Analysis

In this section we analyze the end–system processing and network bandwidth requirements of protocols L1, L2 and L3.

Following the approach first suggested in [61], the receive processing cost (time) is determined by computing the total amount of processing involved in *correctly* receiving a

randomly chosen packet. This includes the time required to receive those copies of this packet (i.e., the original copy plus any retransmissions) that arrive at the receiver, the time required to send/receive any NAKs associated with this packet, and the time needed to handle any timer interrupts associated with this packet. The send processing cost is determined by the total sender processing involved in correctly transmitting a packet correctly to all receivers. This includes the cost required to process any received NAKs, and process retransmissions that are sent out in response to these NAKs.

Unlike other analyses (e.g., [36, 44, 45, 61, 71]) we also analyze the *bandwidth* requirements of the various protocols. For analyzing bandwidth we focus on the following two bandwidth metrics.

1. *Total bandwidth usage*: the total number of bytes sent per successful packet transmission along all of the links to all the receivers[3].

2. *Traffic concentration*: the ratio of average bandwidth usage of the link with the highest average bandwidth usage to the average bandwidth usage over all links in the network. This measure has been used to compare the performances of routing protocols[7, 20].

3.4.1 Processing Cost Analysis

We now derive expressions for sender and receiver processing requirements for protocols L1, L2, and L3. Table 3.1 describes the notation used in the analysis. Some of the notation has been reintroduced from [71, 36]. We assume that processing times have general distributions and that they are independent of each other.

---

[3]More generally we could have a three dimensional table with the dimensions being type of packet (data–packet or NAK), type of link (source link, tail link or backbone link) and direction on the link (upstream or downstream), to determine the bandwidth associated with the transmission of a packet over a certain link in a certain direction. Then we could sum over the bandwidth used over all links.

$$
\begin{array}{lll}
X_p & - & \text{the time to process the transmission of a packet at a sender} \\
X_n & - & \text{the time to process a NAK at a sender} \\
Y_p & - & \text{the time to process a newly received packet at a receiver} \\
Y_t & - & \text{the time to process a timeout at a receiver} \\
Y_n & - & \text{the time to process and transmit/receive a NAK at a receiver} \\
p & - & \text{the end–to–end loss probability at a receiver} \\
p_l & - & \text{the source link or tail link loss probability} \\
T & - & \text{the total no. of tail links} \\
k & - & \text{number of tail links per backbone router/repair server} \\
X^\omega, Y^\omega & - & \text{the send and receive per packet processing} \\
& & \text{time in protocol } \omega \in \{\text{L1, L2, L3}\}
\end{array}
$$

Table 3.1. Notation Used in the Performance Analysis

3.4.1.1   Analysis of L1

Notation specific to L1 is described in Table 3.2. Following the approach described in [71, 36], the mean per–packet processing time for a randomly chosen packet at the sender for protocol L1 can be expressed as,

$$
E[X^{L1}] = E[M_s]E[X_p] + (E[M_s] - 1)E[X_n] \tag{3.1}
$$

The first term corresponds to the mean processing required to transmit a packet. The second term corresponds to the mean NAK processing required at the sender.

The mean processing requirement at a receiver for a randomly chosen packet is

$$
E[Y^{L1}] = E[M_R](1 - p_l)E[Y_p] + (E[M_R] - 1)E[Y_n] + E[(M_r - 2)^+]E[Y_t] \tag{3.2}
$$

The first term corresponds to the processing required to correctly receive a packet. The second term corresponds to the mean processing time required to send and receive NAKs[4].

---

[4]In counting the number of NAKs and the number of timeouts there is an implicit assumption that the lost packet is available at the repair server. For this assumption to hold good a receiver has to cleverly choose NAK retransmission timeout values. Otherwise, the number of NAKs will be more than what we have counted.

$$M_s \quad - \quad \text{number of transmissions from the sender required for all repair}$$
servers to successfully receive a packet

$M_r \quad - \quad$ number of transmissions from a repair server required for
a receiver to successfully receive a packet

$M_R \quad - \quad$ number of transmissions from a repair server to all
the recvrs for which it is responsible successfully
recv a pkt – assuming that no pkts are sent from the sender

$Z^{L1} \quad - \quad$ the per packet processing time at the repair server

Table 3.2. Notation specific to L1

The third term corresponds to the processing required to execute routines due to expiration of NAK retransmission timer. Here, $(x)^+ = \max\{0, x\}$. The mean processing time at a repair server for a randomly chosen packet is

$$E[Z^{L1}] \;=\; E[M_s](1 - p_l)E[Y_p] + (E[M_R] - 1)E[X_p] + (E[M_s] - 1)E[Y_n]$$
$$+ (E[M_R] - 1)E[Y_n] + E[(M_s - 2)^+]E[Y_t] \tag{3.3}$$

The first two terms correspond to the processing of transmissions and retransmissions, received from the sender and the retransmissions sent out to the receivers, respectively. The third term corresponds to processing of NAKs sent to the sender or received from other repair servers. The fourth term corresponds to the processing of NAKs received from the receivers and the fifth term represents the processing cost of the NAK retransmission timeout routine. The terms $E[M_s]$, $E[M_R]$, $E[M_r]$ and $E[(M_r - 2)^+]$ are computed as follows:

We have

$$P(M_s \leq m) = 1 - p_l^m, m = 1, 2, 3 \dots$$

and therefore,

M – number of transmissions from the sender for all receivers
to receive the packet correctly if there were no local recovery

M_n – number of transmissions from the sender for all receivers
in a local neighborhood to receive the packet correctly if there
were no local recovery

M' – number of transmissions from the sender to ensure that at
least one receiver in each neighborhood receives the packet correctly

M'_n – number of transmissions from the sender to ensure that at
least one receiver in a neighborhood receives the packet correctly
if there were no local recovery

M_t – number of transmissions (global or local) required
for any receiver r to correctly receive a packet

Table 3.3. Notation specific to L2

$$E[M_s] = 1/(1 - p_l)$$

We know that for any random variable X, such that $X = 1, 2, 3, \ldots,$

$$E[X] \;=\; 1 + \sum_{m=1}^{\infty} (1 - P(X \leq m)), \qquad (3.4)$$

Now $P(M_R \leq m) = (1 - p_l^m)^k$, hence

$$E[M_R] \;=\; 1 + \sum_{m=1}^{\infty} (1 - (1 - p_l^m)^k)$$

Now $P(M_r \leq m) = 1 - p_l^m$, hence $E[M_r] - 1 = p_l/(1-p_l)$ and $E[(M_r - 2)^+] = p_l^2/(1-p_l)$.

3.4.1.2 Analysis of L2

We now analyze the sender and receiver processing costs of protocol L2. Notation specific to this analysis is described in Table 3.3. We assume that the processing times needed to send and receive a packet are the same (it has been observed in [36] that these are almost same). We also make the optimistic assumption that only one retransmission

is generated per NAK during local recovery in L2. This assumption will result in lower receiver processing costs for L2. The mean processing time at the sender for a randomly chosen packet is

$$
\begin{aligned}
E[X^{L2}] \;=\; & (E[M'] + E[\lfloor (M - M')/2 \rfloor])E[X_p] \\
& (E[M'] + E[\lfloor (M - M')/2 \rfloor] - 1)E[X_n]
\end{aligned}
\tag{3.5}
$$

The first term corresponds to the processing required for transmissions and retransmissions of the packet. This term is made up of two parts. The first part is the expected processing cost of (re)transmissions of the packet so that at least one receiver in every neighborhood receives the packet. The second part is the expected processing cost of additional retransmissions from the sender when receivers cannot locally recover the lost packet, even though it is available in the local neighborhood. Recall that a receiver alternates between local and global recovery in that order. Therefore, $\lfloor (M - M')/2 \rfloor$ additional transmissions are required from the sender to ensure that all receivers correctly receive the packet. The second term corresponds to the processing of global NAKs received by the sender. This number of global NAKs is equal to one less than the number of (re)transmissions from the sender because only one NAK is generated per loss due to NAK suppression[5].

The mean processing time at the receiver for a randomly chosen packet is

$$
\begin{aligned}
E[Y^{L2}] \;=\; & (E[M'] + E[\lfloor (M - M')/2 \rfloor])(1 - p_l)^2 E[Y_p] \\
& + (E[\lceil (M_n - M'_n)/2 \rceil])(1/k + (k - 1)(1 - p_l^2)/k)E[Y_p] \\
& + (E[M'] + E[\lfloor (M - M')/2 \rfloor] - 1)E[Y_n] \\
& + (E[M'_n] + E[\lceil (M_n - M'_n)/2 \rceil] - 1)E[Y_n] \\
& + (E[(M_t - 2)^+] + E[(M'_n - 2)^+])E[Y_t]
\end{aligned}
\tag{3.6}
$$

---

[5]Here the assumption is that NAK suppression works perfectly.

The first two terms correspond to the processing of global and local transmissions at the receiver. The number of local retransmissions is determined by considering a local neighborhood in isolation and determining the number of transmissions needed from the sender to ensure that all the receivers in the neighborhood receive the packet correctly. Then the number of transmissions required to ensure that at least one receiver in the neighborhood receives the packet is found. Half of the difference in these quantities yields the number of local retransmissions. We take the ceiling of this number to account for the fact that a recovery sequence always begins with local recovery. A local transmission is generated within a tail with a probability $1/k$. A receiver receives a local transmission generated in another tail link in its neighborhood with probability $(k-1)(1-p_l)^2/k$. Here we optimistically assume that only one local repair is generated per local NAK.

The third term in (3.6) corresponds to the mean processing time required to send and receive global NAKs. The fourth term is the mean processing time required to send and receive local NAKs. It is important to note here that m' local NAKs are generated even before the packet is available in the local neighborhood. The last term is the mean processing time required for timer routine executions. This consists of two parts. The first part is the expected processing cost of the timer routine executions when the packet is transmitted (globally or locally) and not received. The second part is the expected processing cost of the timer routine executions when local recovery attempts are being made in between global recovery attempts even before the packet is available in the local neighborhood. We determine the expressions for $E[M]$, $E[M']$, $E[M_n]$ and $E[M'_n]$ as follows:

We use the recursive technique described in [5] to obtain

$$
\begin{aligned}
P[M \leq i] &= \sum_{u=0}^{i-1} \binom{i}{u} p_l^u (1-p_l)^{i-u} (1-p_l^{i-u})^T \\
P[M_n \leq i] &= \sum_{u=0}^{i-1} \binom{i}{u} p_l^u (1-p_l)^{i-u} (1-p_l^{i-u})^k \\
P[M' \leq i] &= \sum_{u=0}^{i-1} \binom{i}{u} p_l^u (1-p_l)^{i-u} (1-p_l^{(i-u)k})^W
\end{aligned}
$$

69

$$P[M'_n \leq i] \quad = \quad \sum_{u=0}^{i-1} \binom{i}{u} p_l^u (1 - p_l)^{i-u} (1 - p_l^{(i-u)k})$$

$E[M]$, $E[M']$, $E[M_n]$ and $E[M'_n]$ can be determined using (3.4) and can be substituted into (3.5) and (3.6). Now, $P(M_t \leq i) = 1 - p^i$. Hence,

$$E[(M_t - 2)^+] = \sum_{i=2}^{\infty} (1 - P(M_t \leq i)) = p^2/(1-p)$$

$$
\begin{aligned}
E[(M'_n - 2)^+] \quad &= \quad \sum_{i=2}^{\infty} (1 - P(M'_n \leq i)), \\
&= \quad E[M'_n] - 2 + P(M'_n \leq 1), \\
&= \quad E[M'_n] - 2 + (1 - p_l)(1 - p_l^k)
\end{aligned}
$$

Further, we introduce two random variables X_1 and X_2 such that, $X_1 = 0.5$ when $(M - M')$ is odd, otherwise $X_1 = 0$, and $X_2 = 0.5$ when $(M_n - M'_n)$ is odd, otherwise $X_2 = 0$. Then,

$$
\begin{aligned}
E[\lfloor (M - M')/2 \rfloor] \quad &= \quad E[(M - M')/2 - X_1] \\
&= \quad E[(M - M')/2] - E[X_1] \\
&= \quad (E[M] - E[M'])/2 - P((M - M')\text{is odd})/2
\end{aligned}
$$

and,

$$
\begin{aligned}
E[\lceil (M_n - M'_n)/2 \rceil] \quad &= \quad E[(M_n - M'_n)/2 + X_2] \\
&= \quad E[(M_n - M'_n)/2] + E[X_2] \\
&= \quad (E[M_n] - E[M'_n])/2 + P((M_n - M'_n)\text{is odd})/2
\end{aligned}
$$

Since random variables M and M' (and also M_n and M'_n) are not independent it is not easy to obtain simple expressions for the loss probabilities $P((M - M')\text{is odd})$ and $P((M_n -$

M_d –   number of transmissions from the sender required for all designated
receivers to successfully receive a packet

M_l –   number of transmissions from a designated receiver to all
the recvrs for which it is responsible successfully
recv a pkt – assuming that no pkts are sent from the sender

Z^{L3} –   the per packet processing time at the designated receiver

Table 3.4. Notation specific to L3

M'_n)is odd). Noting that the values of these two probabilities will be significant only when
the number of tails is small and loss probabilities are low, we bound $E[\lfloor(M - M')/2\rfloor]$ and
$E[\lceil(M_n - M'_n)/2\rceil]$ by

$$E[\lfloor(M - M')/2\rfloor] > (E[M] - E[M'])/2 - 1/2$$

$$E[\lceil(M_n - M'_n)/2\rceil] > (E[M_n] - E[M'_n])/2$$

3.4.1.3   Analysis of L3

We now analyze the sender, receiver and designated receiver processing costs of proto-
col L3. Notation specific to this analysis is described in Table 3.4. The mean processing
requirement at the sender for a randomly chosen packet is

$$E[X^{L3}] = E[M_d]E[X_p] + (E[M_d] - 1)E[X_n] \tag{3.7}$$

The mean receive processing requirement at a leaf receiver that is not located at the
same site as the designated receiver is

$$E[Y^{L3}] = E[M_l](1 - p)E[Y_p] + (E[M_l] - 1)E[Y_n] + p^2/(1 - p)E[Y_t] \tag{3.8}$$

71

The mean receive processing requirement at a leaf receiver located at the same site as the designated receiver is

$$E[Y^{L3}] = E[M_l]E[Y_p] + (E[M_l] - 1)E[Y_n] + p^2/(1-p)E[Y_t] \qquad (3.9)$$

The mean receiver processing requirement at a designated receiver is

$$\begin{aligned}
E[Z^{L3}] &= E[M_d](1-p)E[Y_p] + (E[M_d] - 1)E[Y_n] + p^2/(1-p)E[Y_t] \\
&\quad (E[M_l] - 1)(E[X_n + E[X_p]]) \qquad (3.10)
\end{aligned}$$

It is important to note here that the number of NAKs sent from a leaf receiver is only one less than the number of transmissions from its parent. This means that if the designated receiver does not have the repair packet, an end–receiver waits for sufficiently long time before sending a NAK, so that the designated receiver is itself able to recover the packet. Otherwise, the receiver is likely to send many more NAKs to its designated receiver than what we have counted in equations (3.8) and (3.10).

Using the recursive technique described in [5],

$$\begin{aligned}
P[M_d \le i] &= \sum_{u=0}^{u=i-1} \binom{i}{u} p_l^u (1-p_l)^{i-u} (1-p_l^{i-u})^W \\
P[M_l \le i] &= \sum_{u=0}^{u=i-1} \binom{i}{u} p_l^u (1-p_l)^{i-u} (1-p_l^{i-u})^{k-1}
\end{aligned}$$

$E[M_d]$ and $E[M_l]$ can now be derived using equation (3.4).

The maximum processing rate at the sender is $\Lambda_s^\omega = 1/E[X^\omega]$ and the maximum processing rate at the receiver is $\Lambda_r^\omega = 1/E[Y^\omega]$, where $\omega \epsilon \{L1, L2, L3\}$. Under the assump-

tion that the repair server is never a bottleneck, the overall end–system throughput for L1 is given by the minimum of the per–packet processing rates at the sender and the receiver.

$$\Lambda_o^{L1} = \min\{\Lambda_s^{L1}, \Lambda_r^{L1}\} \tag{3.11}$$

Similarly, the overall end–system throughput for L2 is given by the minimum of the per–packet processing rates at the sender and the receiver

$$\Lambda_o^{L2} = \min\{\Lambda_s^{L2}, \Lambda_r^{L2}\} \tag{3.12}$$

The overall end–system throughput for L3 is given by the minimum of the per–packet processing rates at the sender, the receiver and the designated receiver

$$\Lambda_o^{L3} = \min\{\Lambda_s^{L3}, \Lambda_r^{L3}, \Lambda_d^{L3}\} \tag{3.13}$$

Here the maximum processing rate at a designated repair server, $\Lambda_d^{L3} = 1/E[Z^{L3}]$.

3.4.2 Bandwidth Analysis

To analyze bandwidth usage, we consider the network to be made up of three types of links, a source link, backbone links and tail links. Then the mean total bandwidth usage, denoted by B^ω, where $\omega \epsilon \{L1, L2, L3\}$, can be expressed as,

$$E[B^\omega] \;=\; E[B_s^\omega] + E[B_b^\omega]W + E[B_t^\omega]T \tag{3.14}$$

Here $E[B_s^\omega]$, $E[B_b^\omega]$ and $E[B_t^\omega]$ are the mean bandwidth usage on the source link, a backbone link and a tail link respectively, for protocol ω. To determine these quantities, we must need to find the mean number of packets flowing in each of the links per successful transmission of a packet from the sender to all the receivers. For simplicity, we assume

73

that the source, backbone and tail links each consists of a single physical link. A packet is counted for bandwidth usage computation if it is to be offered to a link.

The traffic concentration, denoted by c^ω, can be expressed as,

$$c^\omega = \max(E[B_s^\omega], E[B_t^\omega])/(E[B_s^\omega] + E[B_t^\omega]T/(T+1)) \qquad (3.15)$$

We exclude the backbone links from equation (3.15) because we cannot identify the backbone links that are used for distributing packets across the backbone. This exclusion does not affect our results however, because the backbone links do not become bottlenecks. However, we do obtain somewhat lower values of traffic concentration because the bandwidth usage in the backbone links tends to reduce the average bandwidth usage over all links. For protocol L1,

$$E[B_s^{L1}] = E[M_s]E[B_d] + (E[M_s] - 1)E[B_n]$$

$$E[B_b^{L1}] = E[M_s](1 - p_l)E[B_d] + (E[M_s] - 1)E[B_n]$$

$$E[B_t^{L1}] = E[M_R]E[B_d] + (E[M_r] - 1)E[B_n]$$

Here, B_d is a random variable representing the bandwidth required for a data–packet transmission or retransmission and B_n is another random variable representing the bandwidth associated with a NAK over any of the above three types of links. We assume that B_d and B_n have general distributions and are independent of each other. For protocol L2,

$$E[B_s^{L2}] = (E[M'] + E[\lfloor (M - M')/2 \rfloor])E[B_d]$$
$$+ (E[M'] + E[\lfloor (M - M')/2 \rfloor] - 1)E[B_n]$$

$$E[B_b^{L2}] = (E[M'] + E[\lfloor (M - M')/2 \rfloor])(1 - p_l)E[B_d]$$

$$+ (E[M'] + E[\lfloor (M - M')/2 \rfloor] - 1)E[B_n]$$

$$
\begin{aligned}
E[B_t^{L2}] &= (E[M'] + E[\lfloor (M - M')/2 \rfloor])(1 - p_l)E[B_d] \\
&\quad + (E[\lceil (M_n - M'_n)/2 \rceil])(1/k + (k - 1)(1 - p_l)/k)E[B_d] \\
&\quad + (E[M'] + E[\lfloor (M - M')/2 \rfloor] - 1)E[B_n] \\
&\quad + (E[M'_n] + E[\lceil (M_n - M'_n)/2 \rceil] - 1)E[B_n]
\end{aligned}
$$

For protocol L3,

$$
\begin{aligned}
E[B_s^{L3}] &= E[M_d]E[B_d] + (E[M_d] - 1)E[B_n] \\
E[B_b^{L3}] &= E[M_d](1 - p_l)E[B_d] + (E[M_d] - 1)E[B_n]
\end{aligned}
$$

Now the tails leading to sites with designated receivers have more traffic flowing through them in comparison to tails leading to sites without any designated receivers. For tails leading to sites with designated receivers,

$$
\begin{aligned}
E[B_{t_1}^{L3}] &= E[M_d](1 - p_l)E[B_d] + (E[M_d] - 1)E[B_n] \\
&\quad + (E[M_l] - 1)E[B_d] + (E[M_l] - 1)E[B_n]
\end{aligned}
$$

For tails not leading to sites with designated receivers,

$$
E[B_{t_2}^{L3}] = E[M_l](1 - p_l)E[B_d] + (E[M_l] - 1)E[B_n]
$$

Therefore, for protocol L3, the contribution of the tails links to the bandwidth usage is $E[B_{t_1}^{L3}]W + E[B_{t_2}^{L3}] * (T - W)$.

3.5   Throughput and Bandwidth Comparisons

In this section we first compare the throughput and bandwidth usage of protocols L1, L2 and L3 with a protocol N2 [71] that does not use local recovery[6] to establish the benefits of local recovery. N2 is a receiver oriented protocol that uses global NAK suppression and is shown to be the best of three global recovery schemes (proposed in [71]). Next we compare the performances of L1, L2 and L3. In computing L1's throughput we assume that the repair server has sufficient processing power and is never a bottleneck. The required processing power for this to be true is determined in the next section.

In order to compute the send and receive processing costs we use the measurements reported in Chapter 2 for processing time associated with sending/receiving a data packet and a NAK packet, as well as the timeout processing times. We use $E[X_p] = E[Y_p] = 500\mu$secs, $E[X_n] = E[Y_n] = 85\mu$secs and $E[Y_t] = 32\mu$secs. Here a data packet is of size 1024 bytes and a NAK packet is of size 32 bytes. The number of tails per neighborhood (as defined in Section 3.2), k, is set to 8. For bandwidth comparisons, we choose $E[B_d] = 1024$ and $E[B_n] = 32$.

Figures 3.2 and 3.3 show the throughput increase and bandwidth reduction due to local recovery. The performance of L1 is significantly higher than N2. Further, this behavior becomes more pronounced as the number of tails, T, and the loss probability, p increase. L2 also performs much better than N2. However, the ratio of throughput obtained under L2 and N2 remains equal to 2 and does not change significantly with increase in T or p. This is because the bottleneck node (a receiver for $p = 0.05$ and the sender for $p = 0.25$) under L2 performs only half the amount of processing in comparison to the bottleneck node (sender) under protocol N2 due to L2's alternate local and global recovery. The throughput obtained under L3 is determined by the the designated receiver's throughput (or by the

---

[6]We analyze N2 for our system model.
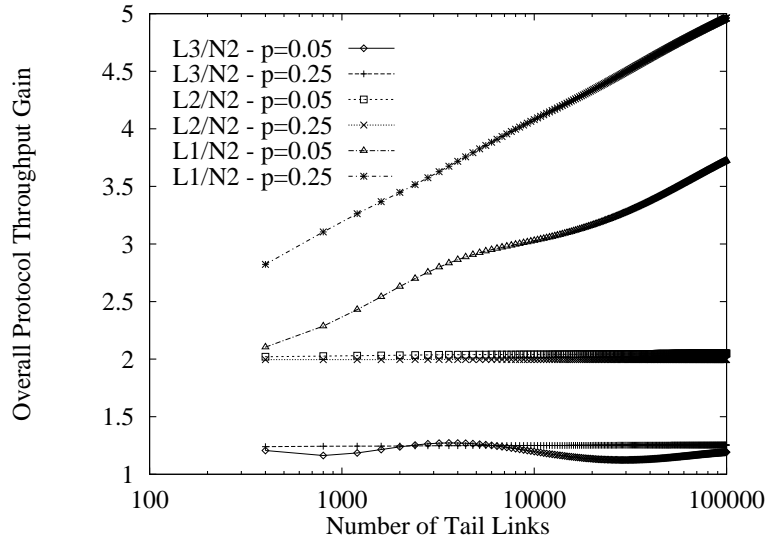
Figure 3.2. Throughput Gain
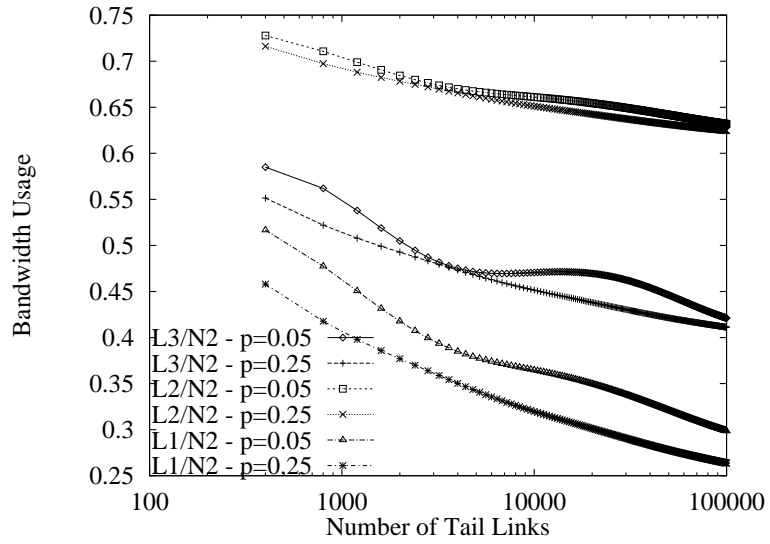
Figure 3.3. Bandwidth Reduction

77

sender throughput for very high p and T) and is only marginally higher than that obtained under N2.

The throughput and bandwidth comparison of protocols L1, L2 and L3 is shown in Figures 3.4 and 3.5. This behavior could also be inferred from Figures 3.2 and 3.3. Protocol L1 has the highest throughput and lowest bandwidth usage in comparison to L2 and L3. This is because the L1 sender is responsible for correctly transmitting packets only over the source link, and an L1 receiver does not receive any redundant packets from the sender. Under L2 the sender is responsible for making sure that at least one receiver in each neighborhood receives the packet going through the source and tail links. Since all packets are multicast to all receivers, the receivers receive some redundant retransmissions from the sender. For these reasons the sender and the receivers in the case of L2 do more processing in comparison to the sender and receivers under L1, leading to higher sender and receiver throughput under L1. This results in higher overall end–system throughput under L1. As more data packets and NAKs are multicast to the entire network under L2, L1 uses much less bandwidth than L2.

It is very interesting to note in Figures 3.4 and 3.5 that even though L2 achieves a higher throughput than L3, L2 also uses more bandwidth. This is because the responsibility of providing repairs is concentrated at the sender and the designated receivers under L3 even though less packets flow on the links. The sender in L3 must continue sending retransmissions until the designated receiver in each local neighborhood correctly receives the packet. In the case of L2, the sender sends retransmissions only until at least one receiver in each local neighborhood receives the packet correctly (the additional transmissions due to failure of local recovery do not contribute much). Therefore, the sender under L3 sends more retransmissions. Also, the designated receiver under L3 does much more processing than a receiver under L2. The bottleneck under L3 moves between sender and the designated receiver with change in loss probability and number of tail links. Thus L3

78
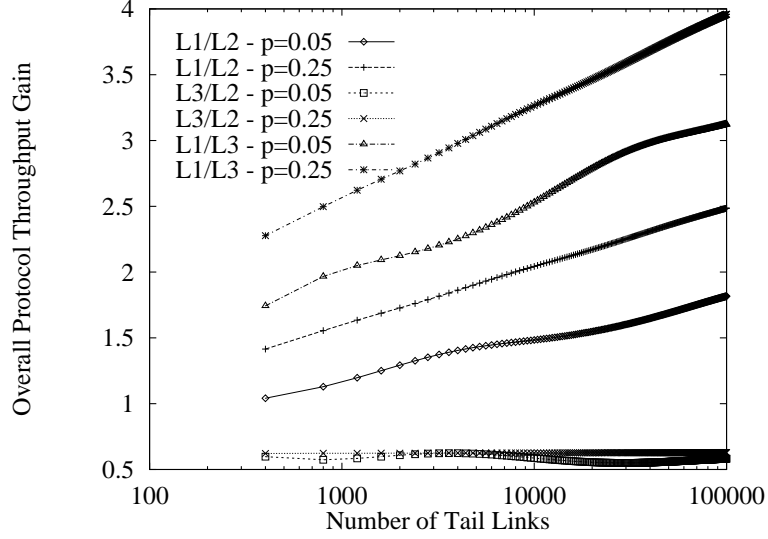
Figure 3.4. Throughput Gain

has lower throughput in comparison to L2. Due to the fact that retransmissions from the sender are sent only to the designated receivers, less bandwidth is used under L3.

We now look at the traffic concentration under protocols L1, L2 and L3. Figure 3.6 shows how the traffic concentration obtained under protocols L1, L2 and L3 varies with the loss probability for 10000 tails. It can be seen that the traffic concentration under L3 increases with loss probability. This is because traffic is concentrated on the tail links leading to the designated receivers (and the source link for high values of p and T). For $p = 0.25$ the traffic concentration under L3 is 250% higher than that under L1 and L2. The traffic concentration under L1 and L2 is always almost 1.

In order to take the processing costs at the repair servers into account in our analysis, let us assume that the repair servers are the same kind of machines as the sender or a receiver. Hence we consider $E[X_p]$, $E[Y_p]$, $E[X_n]$, $E[Y_n]$, $E[Y_t]$ for a repair server to be the same as that for a receiver or sender. We then find the mean total per–packet processing costs at all nodes (sender, repair servers, receivers) under protocol L1. We also find the mean total per–packet processing costs at all nodes (sender, receivers) under L2 and L3. For the case of a single receiver per tail circuit, Figure 3.7 shows how the mean total per–packet

79
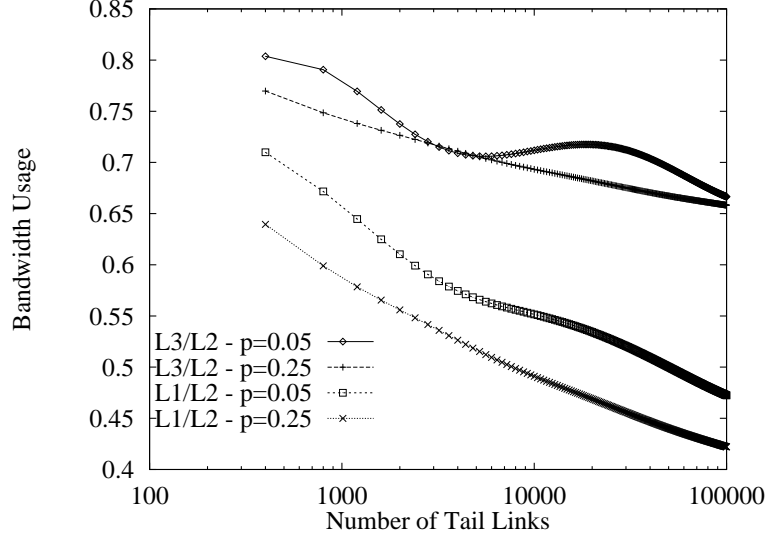
Figure 3.5. Bandwidth Reduction

processing ratio varies with T and p. It can be seen that for $p = 0.05$, the mean total per–packet processing cost under L1 is more than that of L3. This is because for low loss probabilities, the repair server does not need to supply many repairs but must still perform the extra work of receiving all packets. This behavior is also observed when comparing L1 and L2 for $p = 0.05$ and less than thousand tails. With an increase in the number of tails and loss probability, the mean total per packet processing cost under L2 and L3 starts increasing relative to that under L1. The mean total per packet processing cost under L3 is less than L2. Noting the fact that L1 incurs the smallest receiver processing costs, adding more receivers per tail will further reduce, relatively, the mean total per processing cost under L1.

In summary, we see that local recovery leads to higher end–system throughput and lower bandwidth usage. While comparing three local recovery protocols L1 exhibits highest throughput and uses least bandwidth, provided repair servers are not bottlenecks. Among L2 and L3, L2 has higher throughput but L3 has lower bandwidth usage. Interestingly, if we consider the repair servers to have the same processing power as sender or receivers,

80
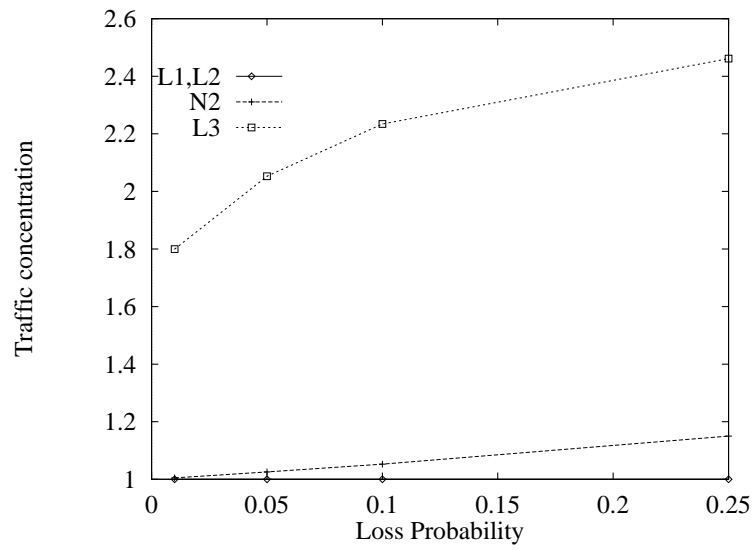
Figure 3.6. Traffic Concentration
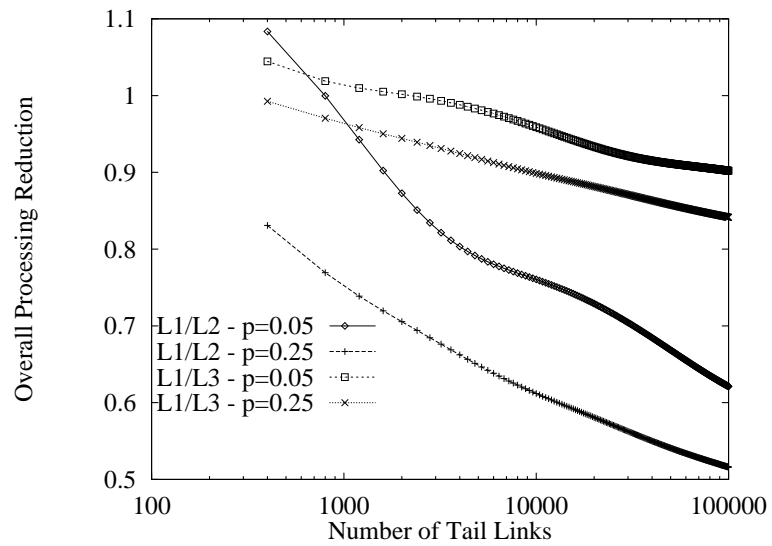
Figure 3.7. Mean Total Per Packet Processing Ratio

81

then there is an overall reduction in processing costs in the repair–server–based approach with increasing loss probabilities, number of tail links and number of receiver per tail link.

The sender throughput under L3 is only slightly greater than the designated receiver throughput. In fact, for higher values of p ($\geq$ 0.25) and large values of T the sender throughput is lower than the designated receive throughput. If protocol L3 is redesigned such that all the transmissions and retransmissions were multicast to every receiver and not just the designated receiver, then there will be less burden on the designated receivers to supply the repairs. Still, the throughput under L3 will improve only slightly (the retransmission processing cost at the sender still remaining the same). At the same time the bandwidth usage and mean total per packet processing will increase more.

This also suggests that using faster machines as designated receivers will not help much because the sender throughput constraint will be dominant. L2 will still have a higher end–system throughput than L3. One way to improve the throughput without increasing bandwidth is to construct a logical tree [45] with a small and constant branching factor. Instead of two levels (a sender sending retransmissions to a large number of designated receivers, and designated receivers supplying repairs to other receivers), this tree would have several levels depending upon the number of tails links. The performance of this approach is very sensitive to the branching factor of the logical tree. We extend the analysis in [45] using our loss model to find that the logical tree approach provides excellent throughput and bandwidth performance when the branching factor is very small. However, for a large number of tails, a smaller branching factor increases the depth of the tree resulting in longer recovery paths. This can potentially lead to high delays. A meaningful comparison of the logical tree–based approach of [45] with the local recovery approaches examined in this chapter is possible only by modeling the delay behavior in addition to modeling the throughput and bandwidth usage.
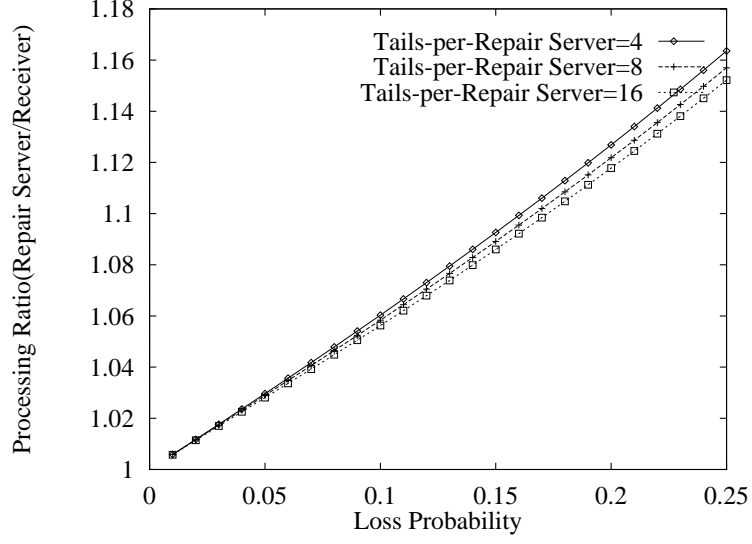
Figure 3.8. Ratio of Repair Server to Receiver Processing Cost

3.6 Processing Power

A repair server must perform receive–side processing of packets and NAKs, and send–side processing of repairs. We now determine the processing power required at a repair server, per multicast session, so that it does not become the bottleneck. Since a receiver performs more processing than the sender under L1, we determine the processing power of the repair server in terms of that receiver (using equations (3.2) and (3.3)). If a repair server can match the receiver throughput then it will never be a bottleneck in a multicast session that uses protocol L1.

In Figure 3.8 we plot the ratio of mean repair server processing cost (assuming that it has the same processing power as a receiver) and the mean receiver processing cost for varying end–to–end loss probability, p. A repair server needs a processing power that is approximately 1.16 times that of a receiver when the number of tail links per repair server equal to 8 and loss probabilities are less than or equal to 0.25. Any reduction (or increase) in tail links per repair server does not change this requirement significantly because even though the repair server has to supply fewer (or more) repairs, the processing cost at a

receiver is also reduced (or increased) because it now receives fewer (or more) retransmissions of a packet.

Thus a repair server need only be slightly faster than a receiver to avoid becoming a bottleneck itself for reasonable loss probabilities. This result applies to the case of a single multicast session. If a repair server is to handle K multicast sessions, then it must be K time faster than a receiver, assuming a receiver belongs to only a single multicast session.

3.7   Conclusions

In this chapter we have investigated one server–based and two receiver–based local recovery approaches for scalable reliable multicast. In the server–based approach, designated hosts, placed inside the network, are used as repair servers. In the receiver–based approaches, loss recovery only involves the participating receivers and the sender. We analyzed and compared protocols developed for these approaches. Using analytical models, we demonstrated the performance gains in using the server–approach in terms of end–system throughput, network bandwidth and overall processing costs. The performance gains increase as the size of the network and the loss probability increase, making the server–based approach more scalable with respect to these parameters. We also estimated the processing power required at the repair servers per multicast session for achieving the performance gains.

A server–based approach could be deployed to improve performance in intranets where services requiring reliable multicast are offered, placing the repair servers at appropriate locations. Depending on the processing power and buffer space that can be provided on the repair servers, the network provider could restrict the use of repair servers for only certain number of multicast sessions (by assigning separate multicast addresses and limiting the use of repair servers only for those addresses). Additional multicast sessions could be allowed with the increase in processing power and buffer space at the repair servers. As far as the Internet is concerned, *static* deployment of repair servers might not be useful

due to the changing network conditions (e.g., route changes [58], changes in loss behavior [79, 80]. Given the performance improvements of the server–based approach, it is worth studying how to make it more dynamic and flexible. We address this issue in Chapter 5.

In this chapter, we have considered the mean total per–packet processing cost at all nodes (sender, receivers and repair servers) as a performance metric for taking into account the processing cost of repair servers. We need to find better cost metrics to take into account the processing and buffering resources used inside the network. We also need to study the performance degradation due to insufficient processing and buffering resources at the repair servers, a topic we address in the following chapter.

We argued that the receiver–based approach that builds logical trees is likely to incur higher delays. Hence there is an immediate need to analyze the delay of this approach (and also L1 and L2 for comparison) and study the tradeoff between delay, throughput and bandwidth usage. Our system model could be extended to take into account temporal correlation in the network loss[79] and to also consider more complex topologies with heterogeneity. It is easy to extend the analysis of L1 and L3 to any complex topology. The analysis of L2 is slightly more complicated.

CHAPTER 4

BUFFER REQUIREMENTS AND REPLACEMENT POLICIES FOR MULTICAST REPAIR SERVICE

In the previous chapter we saw that server–based local recovery yields higher through-put and lower bandwidth usage in comparison to receiver–based local recovery. However, server–based local recovery will perform efficiently only when sufficient processing and buffering resources are available at the repair servers. Since these repair servers are likely to be used by a large number of reliable multicast sessions, as well as by other applications including web caching and multimedia gateways, it is extremely important to understand their resource requirements. In this chapter we examine the buffer and processing require-ments of repair servers for reliable multicast.

One function of a repair server is to buffer sender's transmissions in order to be able to later retransmit them to repair downstream losses. Theoretically, in a NAK–based approach such as ours, a repair server needs to buffer each packet for an infinitely long amount of time so that it can retransmit a packet whenever a NAK for that packet is received. Such perfect local recovery requires an infinite buffer space at the repair servers. Realistically, a multicast session will be allocated a certain number of buffers; when all its buffers are full, a buffer replacement policy will be used to replace old packets occupying the buffer with new packets arriving from the sender. Given the replacement of buffered packets, it is possible that some of the packets will be removed from the buffer at the repair server before they are successfully received downstream. These packets must now be recovered from upstream of the repair server.

Our work in this chapter examines the buffer requirements of repair servers; it divides into three parts:

- First, for a simple FIFO (*oldest–first*) buffer replacement policy, we use analytical models to determine the amount of buffer space required at a single repair server to ensure that it can almost always supply a repair when required. We show how the repair server buffer requirements depend upon the packet arrival process at the repair server and the duration of time during which a packet needs to be held in its buffer. This latter quantity depends upon the required number of retransmissions and the length of time between retransmissions of the packet.

- Next, we extend the FIFO buffer analysis of the single repair server case to a topology consisting of multiple repair servers serving different loss domains (a loss domain consists of a repair server and its downstream receivers) to determine how the end–system throughput and network bandwidth consumption will vary with buffer size. In the same context, we study the impact of the number of repair servers in the multicast tree on the mean number of additional retransmissions required from the sender. We show that for homogeneous loss domains it is necessary to populate almost all of the domains with repair servers before a marked reduction is observed in the mean number of additional retransmissions from the sender.

- Third, for the purpose of improving buffer utilization and ensuring graceful performance degradation when only a small number of buffers are available, we propose a new buffer replacement policy. This policy, called FIFO–MH, replaces packets in a FIFO manner only after they have been held for a specified minimum amount of time. We examine the effect of different retransmission request arrival patterns on the performance of FIFO–MH. We also simulate the LRU (*least recently used*) replacement policy. Based on our performance study and noting that it could also be easily implemented, we recommend the use of the simple FIFO buffer replacement policy.

The rest of this chapter is structured as follows. In Section 4.1 we examine the related work on buffer requirements for reliable multicast. In Section 4.2 we present our network model. In Section 4.3 we study the buffer requirements of a single repair server when a simple FIFO buffer replacement policy is used. In Section 4.4 we consider a network consisting of a sender, receiver and multiple repair servers and extend the analysis of Section 4.3 to determine the dependence of the end–system throughput and network bandwidth consumption on buffer size. In Section 4.5, we consider more complex buffer management policies for improving buffer utilization. Our conclusions and directions for future work are described in Section 4.6.

4.1 Related Work

Most of the earlier performance studies on reliable multicast have focussed upon network bandwidth and end–host resources. There has been very little work on memory requirements for caching packets for reliable multicast. In our earlier work [37], we made the assumption that a repair server can store an infinite number of packets and determined the average buffer occupancy at the repair servers; we did not study any buffer replacement policies. Our work on buffer occupancy has been extended in [66] to determine the buffer occupancy at repair servers for protocols that provide parity encoding or forward error correction service. Our buffer occupancy formalism has also been used to study the activation/deactivation of repair servers in [53].

4.2 Network Model

We adopt the same network model as in Chapter 3. A sender multicasts data to a group of receivers. In addition, there are designated hosts, called repair servers, that are co–located with routers, typically at strategic points (i.e., above lossy links) inside the network. The sender and the repair servers conceptually constitute a *repair tree* rooted at the sender. The repair servers join the multicast group on which data is being transmitted

and receive data packets in exactly the same manner as other receivers. These packets are cached for the purpose of supplying future repairs. On detection of loss, a receiver recovers the packet from the closest upper–level repair server. The repair servers in turn recover lost packets from upper–level repair servers. The sender can be considered the highest level repair server.

Note that the repair tree is the same as the physical multicast tree. Therefore, a packet lost at a certain level cannot be received at a lower level. The findings of this chapter can also be applied to logically hierarchical repair trees [47, 78, 44], although we do not consider that application here. We assume that the reliable multicast protocol used for error recovery is L1 (or L1–like), as described in Chapter 3.

4.3   Buffer Requirements of a Repair Server

In this section, we consider a single repair server that has the responsibility of supplying repairs to its downstream receivers, as shown in Figure 4.1. The repair server buffers packets that arrive from the sender in order to supply future repairs. When the repair server's buffers are full, a buffer replacement policy is used to replace old packets occupying the buffer with new packets arriving from the upstream sender. It is possible that some of the packets will be removed from the buffer at the repair server before they are successfully received by all of the downstream receivers. These packets must then be recovered from an upstream repair server or from the sender. Additional retransmissions will thus be required from an upper–level repair server or sender. Our goal in this section is to determine the average number of these additional retransmissions as a function of the buffer size at the repair server. We then determine the buffer size required to maintain the average number of such additional retransmissions from upstream repair servers below a threshold.

Before proceeding with our buffer requirement study, we introduce a key parameter, the retransmission interval, that is used in our study. We define *retransmission interval* to mean the following two kinds of time intervals; the time interval from when a packet is
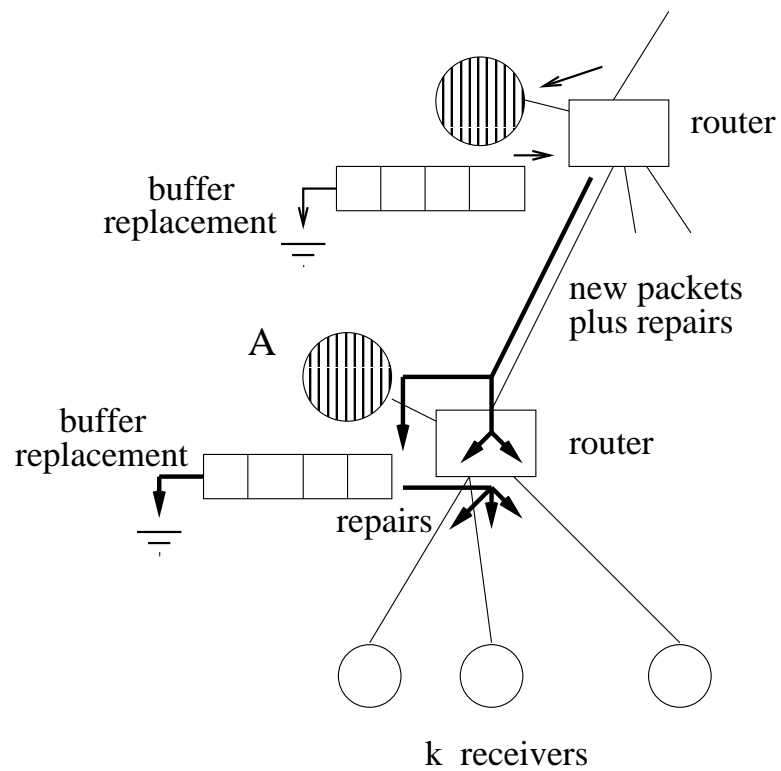
Figure 4.1. Repair server serving k receivers

first buffered at a repair server until it is first retransmitted and the time interval between successive retransmissions of the packet from the repair server.

In NAK–based protocols, a repair server retransmits a packet on receiving a NAK for that packet from downstream receivers. NAKs can arrive at a repair server at variable time intervals due to the following reasons.

- The round trip times from the repair server to the receivers are variable.

- The time to detect loss at receivers is also variable (e.g., depends on the inter–packet separation[1]).

- NAKs are sent after a random delay for the purpose of NAK suppression.

If the repair server retransmits a packet whenever and as soon as it receives a NAK for that packet, the retransmission interval is a variable quantity and the repair server has no control over its values. Alternatively, a repair server protocol might batch NAKs for a packet and send out retransmissions of the packet periodically as long as there is a pending NAK for that packet. In the second case, the retransmission interval is a constant and its value can be set by the repair server.

We now determine, through analysis, the mean number of additional retransmissions that an arbitrary packet incurs from an upstream repair server (or the sender) in order to ensure that it is correctly received by all receivers below the repair server. In the analysis, we make the following assumptions. Note that these are modeling assumptions, not protocol requirements. We focus on a single repair server (e.g., A in Figure 4.1) that supplies repairs to k downstream receivers and itself receives packets from upstream repair servers and/or the original sender[2].

---

[1]Losses are usually detected by observing gaps in the sequence numbers of the packets received and hence the time to detect loss also depends upon the inter–packet separation time.

[2]In this chapter, when we use the term "upstream," it is taken to mean upstream with respect to the repair server, unless otherwise qualified

- Packets arrive at the repair server according to a Poisson process with arrival rate $\lambda = 1/\delta$. These also include the retransmissions of packets that were lost upstream of the repair server. These packets are buffered at the repair server on arrival. The repair server can buffer up to B packets. When all of the buffers are occupied, packets are replaced in simple FIFO (or oldest first) order.

- The repair server will periodically retransmit a packet every δ' time units, as long as it receives NAKs for that packet, i.e., the retransmission interval is a constant[3] denoted by δ'.

- The repair server is responsible for supplying repairs to k receivers. Receiver i sees independent loss on the link between the repair server and itself with probability p_i, $i = 1, 2, \ldots, k$.

- A packet, once buffered and subsequently overwritten, is not buffered again at the repair server. The Poisson arrivals described above do not include these packets. There are two reasons for this assumption. First, a scheme that buffers packets that were once buffered and subsequently overwritten by replacing other packets in the repair server buffers will do well only when certain packets are more "desirable" than the others. This might be true in those contexts where there is *locality in reference* (for example, web caches) but unlikely in the reliable multicast case (see the discussion on using *least recently used* buffer replacement policy in Section 4.5). Second, this assumption greatly simplifies the analysis.

4.3.1 Analysis

Let the random variable H represent the number of possible retransmissions of an arbitrary packet occupying a buffer at the repair server before it is replaced. This quantity is

---

[3]This assumption is justified when a repair server protocol batches retransmission requests (NAKs) and sends out retransmissions of a packet periodically as long as there is a pending NAK for that packet. The case when δ' is a variable is studied in Section 4.5

independent of whether the packet is actually ever retransmitted. As the repair server has B buffers, no retransmissions of the packet are possible if B additional packets arrive subsequent to the packet in time less than δ'. We define the random variable $Q(\tau)$ to denote the number of packet arrivals at the repair server within a time interval of length τ. Therefore,

$$P(H = 0) \;\; = \;\; P(Q(\delta') \geq B)$$

For $i = 1, 2, 3, \ldots$,

$$P(H = i) \;\; = \;\; P(Q(i\delta') < B \leq Q((i+1)\delta'))$$

For $i = 0, 1, 2, \ldots$,

$$P(Q(i\delta') \geq B) \;\; = \;\; 1 - \sum_{n=0}^{B-1} \frac{(i\lambda\delta')^n e^{-i\lambda\delta'}}{n!}$$

Let the random variable N denote the number of additional retransmissions required from an upstream repair server or the sender. Let the random variable M denote the number of transmissions required for all k receivers to correctly receive the packet, if the repair server were to supply all the transmissions. Then,

$$
\begin{aligned}
P(N \leq m) \;\; &= \;\; \sum_{h=0}^{\infty} P(N \leq m | H = h) P(H = h) \\
&= \;\; \sum_{h=0}^{\infty} P(M \leq m + 1 + h) P(H = h) \quad\quad (4.1)
\end{aligned}
$$

where $P(N \leq m | H = h) = P(M \leq m + 1 + h)$ is the conditional probability that the number of retransmissions from upstream is less than or equal to m given $H = h$. We add one to $m + h$ to account for the original transmission of the packet.

The mean number of additional retransmissions from upstream, $E[N]$, is expressed as follows.

$$
\begin{aligned}
E[N] &= \sum_{m=0}^{\infty} (1 - P(N \leq m)) \\
&= \sum_{m=0}^{\infty} (1 - \sum_{h=0}^{\infty} P(M \leq m+1+h) P(H = h)) \\
&= \sum_{m=0}^{\infty} (1 - \sum_{h=0}^{\infty} (1 - p_1^{m+1+h})(1 - p_2^{m+1+h}) \ldots (1 - p_k^{m+1+h}) P(H = h))
\end{aligned}
$$

When $p_1 = p_2 = \ldots = p_k = p$, we have

$$
E[N] = \sum_{m=0}^{\infty} (1 - \sum_{h=0}^{\infty} (1 - p^{m+1+h})^k P(H = h)) \tag{4.2}
$$

The value $E[N]$ is a measure of the inability of the repair server to supply local repairs. In Section 4.5, we will look at ways to minimize $E[N]$. Using (4.2), we can now find the buffer requirements at a repair server as a function of $E[N]$, p, k, $\lambda \ (= 1/\delta)$ and δ'. Note that λ and δ' always occur together as a product in the above expressions.

4.3.2 Numerical Examples

We use (4.2) to find the minimum buffer space required to maintain $E[N]$ below a certain value for different values of p, R and $\lambda\delta'$. In this section, we choose to maintain $E[N] \leq 0.01$, i.e., on average, one additional retransmission or less will be required for every hundred packets.

We first study the impact of the product $\lambda\delta'$, or the ratio δ/δ', on the buffer requirement. The ratio δ/δ' is a key parameter. Its value can be interpreted as the number of retransmissions of a packet from the time it arrives at the repair server until the time the next packet arrives. If only one retransmission were required to ensure that all downstream receivers receive the packet, and, δ/δ' were greater than 1, then only one buffer would be required at the repair server. The buffer requirement goes up as δ/δ' decreases or as the loss probability p increases, as more retransmissions are needed in this latter case.

Figure 4.2 shows how the minimum buffer size varies with the ratio δ/δ' for different values of p. Here, k is set to 10. We see that the minimum buffer size is very sensitive to small values of δ/δ'. When $\delta/\delta' = 0.2$, 13 buffers are required to satisfy $E[N] \leq 0.01$. An increase in the arrival rate or in the time between retransmissions of a packet results in decrease in δ/δ', which results in the need for more buffers. At the same time, moderate to high values of δ/δ' result in extremely small buffer requirements. These results confirm our intuition as discussed above.

In order to obtain realistic buffer requirements, it is important to determine realistic values of the ratio δ/δ'. A very rough value for this ratio can be obtained if we assume that the reliable multicast sender transmits packets at a rate, S, that is "TCP–friendly" with respect to the link loss rate, p. Using the formula $S = 1.22/(RTT\sqrt{P})$ (from [24]) for a reliable TCP connection between a sender and a receiver, where S is the sender rate in pkts/sec, RTT is the round trip time from the sender to the receiver and P is the loss probability between the sender and the receiver, if we set $S = 1/\delta$, $RTT = \delta'$ and $P = p$ then we have $\delta/\delta' = \sqrt{p}/1.22$. We use this formula to select values of δ/δ' and find that 13–18 buffers are required to satisfy $E[N] \leq 0.01$, for a wide range of p (from $0.01 - 0.20$). The buffer requirements corresponding to the estimated values of δ/δ' are shown by a 'x' in Figure 4.2 where the leftmost 'x' corresponds to $p = 0.01$ and the rightmost corresponds to $p = 0.20$.

We now study the impact of the number of receivers, k, on the buffer requirement. Figure 4.3 shows how the minimum buffer size needed to keep $E[N] \leq 0.01$ varies with the number of receivers. Here the arrival rate is 128pkts/sec ($\delta = 7.81$ms) and $\delta' = 40$ms. We observe that the buffer requirement changes very slowly with number of receivers. The same behavior is observed for other values of the δ and δ'. This is because the number of retransmissions required from the repair server, A (Figure 4.1), for all receivers to correctly receive the packet increases slowly as a function of k (O($logk$)) [71]. In both Figures 4.2 and 4.3 the buffer requirement increases with link loss probability, as expected.
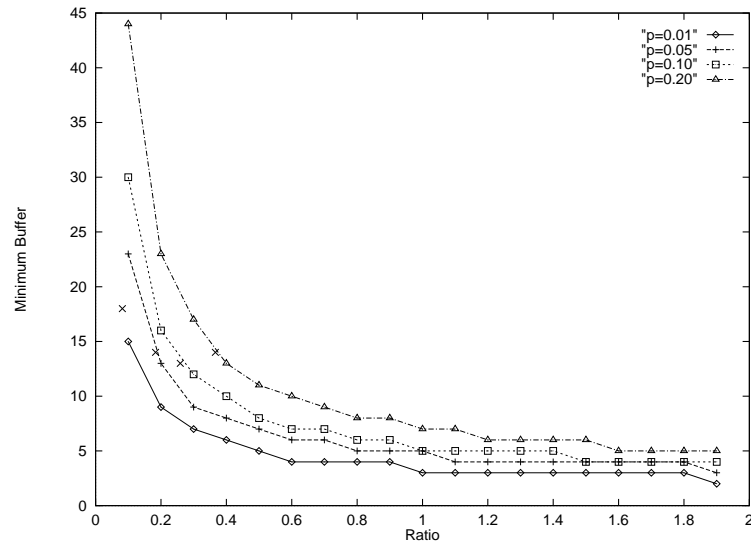
Figure 4.2. Minimum Buffer Size vs δ/δ'
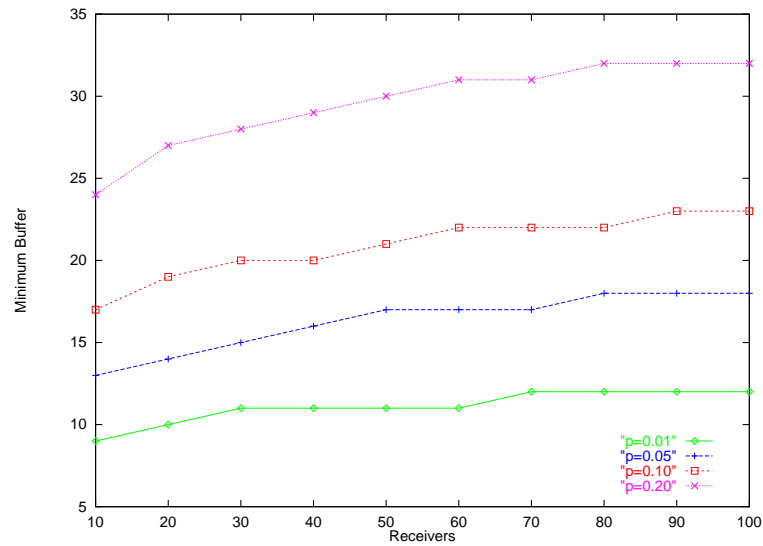
Figure 4.3. Minimum Buffer Size vs Number of Receivers
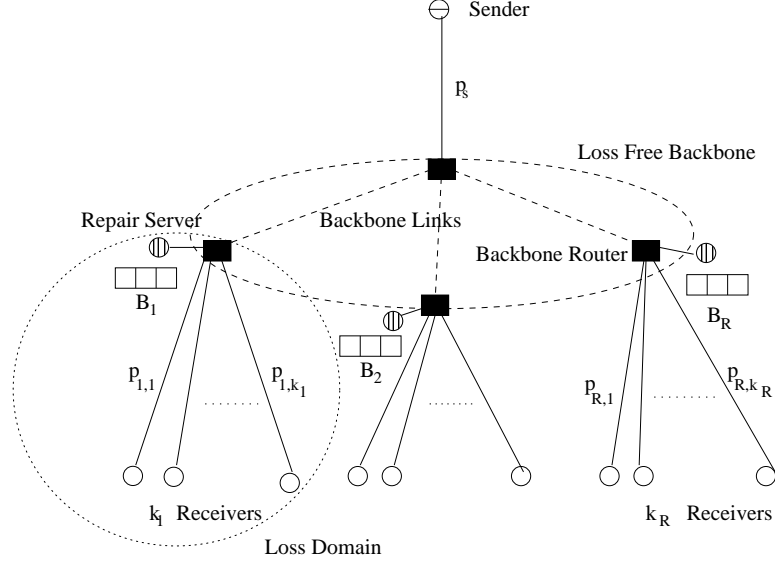
Figure 4.4. Repair server serving k receivers

In summary, we find that the buffer requirements are very sensitive to small values of the number of retransmissions of a packet within the packet inter–arrival time. When this value is large, very few buffers are required at the repair server to meet the demands of local recovery. We also find that the buffer requirements increase very slowly as the number of receivers increases.

4.4   Dependence of Overall Performance on Buffer Size

In the last section we found the mean number of additional retransmissions required, due to finite buffer space at a repair server, from an upper–level repair server (or sender) in the repair tree. In this section we consider the entire repair tree as described in Section 4.1. We determine, through analysis, the mean additional retransmissions required from the sender as a function of buffer space at the repair servers. This metric determines the effect of buffer size on the end–system throughput and network bandwidth usage. To compute this metric, we make the following assumptions.

- Our system model (see Figure 4.4) is similar to the one used in Chapter 3. There are R repair servers. Repair server i has B_i buffers and is responsible for supplying repairs to its k_i downstream receivers, where $i = 1, 2, \ldots R$. We consider loss only in the links from the source to the backbone and from the backbone to the receivers[4]. The loss seen by receiver j in loss domain i on the link(s) from the backbone to itself is denoted by $p_{i,j}$. The loss on the link(s) from the sender to the backbone is represented by p_s. Loss events are assumed to be temporally independent. Each loss domain in Figure 4.4 is the same as the single repair server system studied in Section 4.2.

- As before, packets arrive at each repair server according to a Poisson process with arrival rate $\lambda = 1/\delta$. These arrivals also include the retransmissions of packets that were lost on the link(s) from the source to the backbone. Packets are buffered at the repair server on arrival. Repair server i can buffer up to B_i packets. When all of the buffers at a repair server are occupied, packets are replaced in simple FIFO (or oldest first) order. A repair server retransmits a packet with a fixed time interval δ' as long as it receives NAKs for that packet. A packet once buffered and subsequently overwritten is not buffered again at a repair server.

4.4.1 Analysis

We focus on an arbitrary packet sent by the sender. Let the random variable A_i denote the number of additional transmissions from the sender due to finite buffers at repair server i, and the random variable A denote the total number of additional retransmissions from the sender due to finite buffers at all the repair servers. Then,

$$P(A \leq m) = P(\max(A_1, A_2, \ldots, A_R) \leq m)$$

---

[4]Each site below a tail link (defined in Chapter 3) is assumed to contain only one receiver.

$$= \prod_{i=1}^{R} P(A_i \leq m) \qquad (4.3)$$

Using (4.1) from the previous section, we have

$$P(A_i \leq m) = \sum_{h_i=0}^{\infty} (1 - p_{i,1}^{m+1+h_i})(1 - p_{i,2}^{m+1+h_i}) \dots (1 - p_{i,k_i}^{m+1+h_i}) P(H_i = h_i) \quad (4.4)$$

where H_i is the random variable denoting the number of possible retransmissions of the packet from repair server i before it is replaced.

The mean number of total additional retransmissions from the sender due to finite buffer space at the repair servers can be determined by using (4.3) and (4.4) in the following expression.

$$E[A] = (1/(1 - p_s))(\sum_{m=0}^{\infty} (1 - P(A \leq m))) \qquad (4.5)$$

In equation 4.5, the factor $1/(1 - p_s)$ is used to account for the loss in the link(s) from the sender to the backbone.

4.4.2 Numerical Examples

We use (4.5) to examine how the overall performance depends on buffer size at repair servers. We assume that $k_i = k$, $B_i = B$ and $p_{i,j} = p$ for $i = 1, 2, \dots, R$ and $j = 1, 2, \dots, k$.

Figure 4.5 shows the dependence of the mean additional retransmissions from the sender on the buffer size at the repair servers. Here, the number of repair servers, $R = 100$, the mean arrival rate $\lambda = 1/\delta = 128$pkts/sec, $\delta' = 40$ms, and number of receivers per repair server, $k = 10$. The loss probability between the sender and the backbone, p_s is set to 0.05. When $p = 0.05$, 23 buffers are required at each repair server to ensure that less than 0.01 additional retransmissions are sent from the sender. Beyond a minimum number of buffers, the number of additional retransmissions from the sender falls quickly as
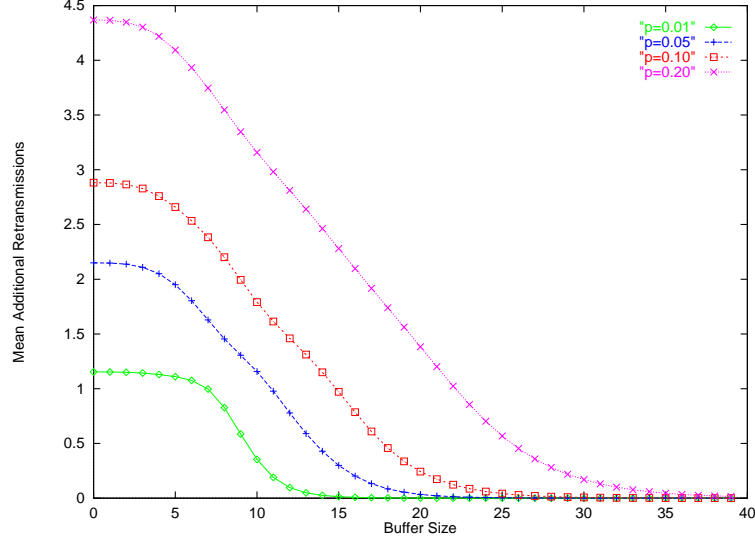
Figure 4.5. Mean Additional Retransmissions vs Buffer

the buffer size at each repair server increases. This is because the mean number of additional retransmissions from the sender depends upon factors such as $(1 - p^{cB})$ where c is a constant.

We also use (4.5) to study the dependence of the mean additional retransmissions required from the sender on the number of repair servers. Since we are interested in looking at the repair server population, we make the best case assumption that a repair server, if there is one in a domain, has enough buffers so that buffering is not a constraint. We set the buffer size to infinity in (4.5) for loss domains that have repair servers and to zero for the loss domains that do not have repair servers.

Figure 4.6 shows the dependence of the mean additional retransmissions from the sender on the number of repair servers. Here, there are 100 loss domains, with 20% of the loss domains experiencing 25% loss and 80% of the loss domains experiencing 1% loss at each link from backbone to a receiver. We populate the higher loss domains with repair servers first. In other words, the first 20 repair servers are placed in the high loss domains. We observe that the mean additional number of sender retransmissions can be reduced by about 50% (from 4 to 2) by populating only the high loss domains with repair servers. A

100
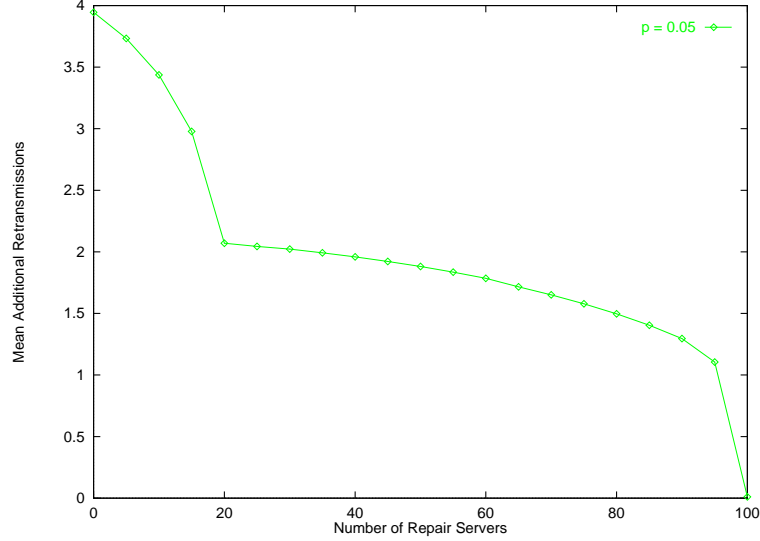
Figure 4.6. Mean Additional Retransmissions vs Number of Repair Servers

more interesting observation is that almost *all* of the remaining 80% loss domains must be populated with repair servers before we see any additional marked reduction in the mean number of retransmissions from the sender. This is because for homogeneous loss domains, the mean number of additional transmissions from the sender varies as $log(R - r)$ where R is the maximum number of loss domains and r is the number of loss domains that have a repair server. A similar behavior is observed for other values of the system parameters as well. These results suggest the obvious fact that repair servers should be placed in high loss domains first. Interestingly, they also suggest that when loss domains are homogeneous, almost all domains need to have a repair server before we see any marked performance improvement in end–system throughput and network bandwidth usage due to local recovery.

4.5 Retransmission Buffer Replacement Policies

In the previous two sections we studied the impact that finite buffers have on performance when old packets occupying the buffer at the repair server are replaced by newly arriving packets in a FIFO manner. The problem with this policy is that when there are in-

101

sufficient buffers, it is possible for a packet buffered at a repair server to be replaced before it gets a chance to be sent as a repair. In such a situation, buffering a packet at a repair server serves no purpose, since lost packets must then be recovered from the upper–level repair server. In this section we propose a new buffer management policy, FIFO–MH, with the goal of making the best use of the finite buffer space, however small, at the repair servers. We study how this policy compares with a simple FIFO buffer replacement policy. We also compare the performance of the simple FIFO policy and LRU (least recently used) buffer replacement policy. We focus on a single repair server and use the mean number of additional retransmissions due to finite buffer space at the repair server, from an upper–level repair server or sender, as a measure for studying and comparing the performance of the different buffer replacement policies.

4.5.1 FIFO with Minimum Holding Time

We propose a new buffer management policy, called FIFO with minimum holding time (or FIFO–MH in short), in which packets are replaced in a FIFO manner but each packet is held in its buffer for a specified minimum amount of time. This minimum amount of time is called the *buffer holding time*. Our heuristic for choosing a buffer holding time is that *a packet should be buffered for at least one retransmission interval before it can be replaced*. In other words, the buffer holding time should be at least one retransmission interval (as defined in Section 4.3). This heuristic is based on the intuition that if a packet is buffered for some finite time, it is better to keep it for at least one retransmission interval, since this is the minimum amount of time needed to determine if a retransmission of that packet will be needed.

In order to use the above heuristic when the retransmission interval is variable, the repair server must maintain a running estimate of the retransmission interval. At any time, the buffer holding time is set to the most current estimate of the retransmission interval. More generally, our FIFO–MH policy can be described as follows.

102

1. A packet occupying a buffer is marked at the time of its first retransmission.

2. When a new packet arrives and there are no free buffers, if the oldest packet in the repair server buffer is marked or has been in the buffer for more than the buffer holding time, then the oldest packet is replaced by the new packet otherwise, the new packet is discarded.

We study the FIFO–MH policy through simulation under the assumption that packets arrive at the repair server according to a Poisson process and for the cases of constant retransmission intervals, retransmission intervals from known distributions, and retransmission intervals based on measurements taken from the Internet. Our simulation model is identical to the analytical model used in Section 4.3. The only differences are that the buffer replacement policy now is FIFO–MH instead of FIFO and the retransmission interval, δ', is not necessarily a constant. To compare the performance of FIFO–MH with FIFO, we also simulate the FIFO policy for the above cases.

4.5.1.1 Constant Retransmission Interval

When the retransmission interval is constant the buffer holding time time under FIFO–MH is simply set to the value of the retransmission interval which is either known to the repair server or can be easily estimated by the repair server. Figure 4.7 shows the benefit of using FIFO–MH over FIFO for three different arrival rates, when the retransmission interval is 40ms, $p = 0.05$ and $k = 10$. We observe that the mean number of additional retransmissions under FIFO–MH is less than that under FIFO. When the buffer size is 5 and arrival rate is 128pkts/sec, the mean additional retransmissions under FIFO–MH is 55% of the mean additional retransmissions under FIFO.

If we increase the arrival rate or reduce δ, and, increase B while keeping $B\delta$ fixed, we find that FIFO–MH performs better for higher values of the arrival rate and larger number of buffers. Figure 4.7 shows the higher performance of FIFO–MH in comparison to simple FIFO when the arrival rate is increased to 256pkts/sec and to 512pkts/sec. When the buffer
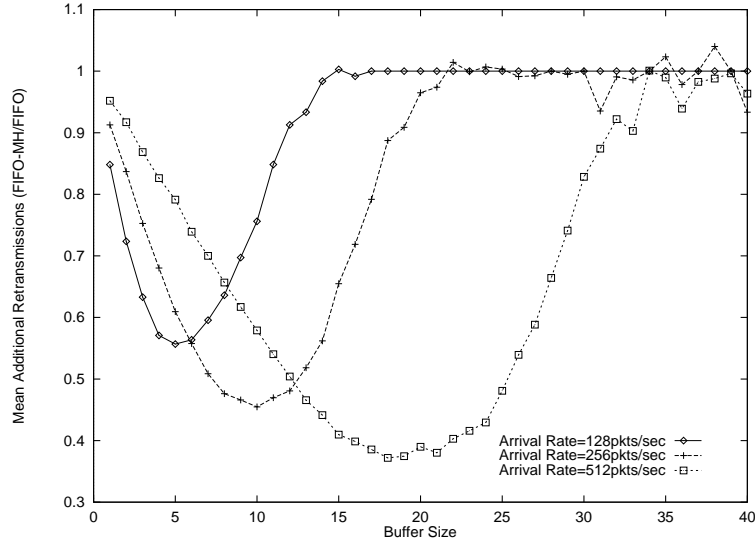
Figure 4.7. Mean Additional Retransmissions Ratio vs Buffer Size

size is 10 and the arrival rate is 256pkts/sec the mean number of additional retransmissions under FIFO–MH is 45% of the mean number of additional retransmissions under FIFO. This is because, under FIFO–MH, when the arrival rate is not very high and the buffer size is small, a packet might be held in its buffer for more than one retransmission interval (but less than two retransmission intervals). This longer holding time wastes the buffer for that amount of time. When the arrival rate and the buffer size are doubled the holding time beyond one retransmission interval is reduced.

Figure 4.7 also shows that the difference in the mean number of additional retransmissions under FIFO–MH and FIFO reduces as the number of buffers increases. This is because when the number of buffers is large enough to allow each packet to be held for at least one retransmission interval, there is no difference between FIFO and FIFO–MH.

Note that holding the packet in the buffer for at least one retransmission interval is not necessarily optimal. For example, when the number of buffers is large enough that each packet is always present in its buffer for at least one retransmission interval before being replaced and when the probability of the receivers requiring more than one retransmission from the repair server is high, then higher performance might be obtained if the buffer

104

holding time is two or more retransmission intervals. The problem of deciding when to increase the buffer holding time to two or more retransmission intervals is an interesting topic for future work.

4.5.1.2 Variable Retransmission Interval

We now consider two cases when the retransmission interval is variable. In the first case, the retransmission interval has a known distribution. In the second case the distribution is derived empirically from round trip time measurements.

- **Retransmission Interval with Exponential Variation**

We represent the retransmission interval as the sum of a constant delay and an exponentially distributed random variable. Let Y be a random variable denoting the retransmission interval, c be the constant minimum delay and Z be an exponentially distributed random variable. We have $Y = c + Z$. Here c models the round trip propagation delay.

We first determine what statistic (including mean, mode, median) of Y best allows the repair server to determine a good buffer holding time under FIFO–MH. The repair server computes and estimate of this statistic and sets the buffer holding time to this estimate. We then compare the performance of FIFO–MH and FIFO. Note that we are not proposing any repair server algorithms for estimation of the retransmission interval.

In order to determine a suitable statistic of Y that can be used as the buffer holding time, we compute the mean number of additional retransmissions under FIFO–MH by setting the buffer holding time to constant values that are taken from the range of values of Y.

Figure 4.8 shows how the mean number of additional retransmissions varies as the buffer holding time increases. Here, the arrival rate is 128pkts/sec, $B = 5$, $p = 0.05$ and $k = 10$. $E[Z]$ and c are so chosen that $E[Y]$ is always 40. We observe from the figure that we can operate FIFO–MH at any buffer holding time as long as it is greater than or equal to c but not much higher than $c + E[Z]$. The same behavior is observed for other values of the arrival rate, B, p, k and $E[Y]$. Some of the candidates for the buffer holding
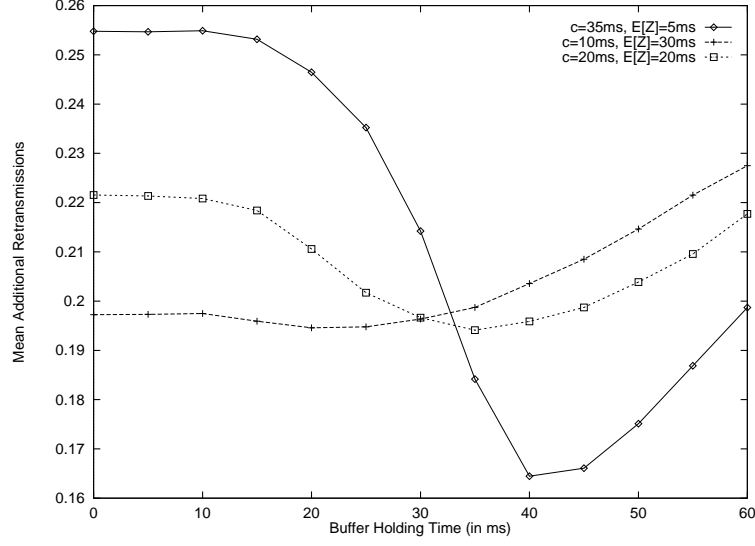
Figure 4.8. Exponentially distributed Z

time are mode(Y), median(Y) and $E[Y]$. We observe that median(Y) is most suitable for all the three curves shown in Figure 4.8 (median(Y) = 38.5, mode(Y) = 35, $E[Y] = 40$, when $c = 35$ and $E[Z] = 5$; median(Y) = 34, mode(Y) = 20, $E[Y] = 40$ when $c = 20$ and $E[Z] = 20$; median(Y) = 31, mode(Y) = 5, $E[Y] = 40$ when $c = 5$ and $E[Z] = 35$).

We now compare the performance of FIFO–MH and FIFO by setting the buffer holding time to median(Y) under FIFO–MH. Figure 4.9 shows how the ratio of mean number of additional retransmissions under FIFO–MH to that under FIFO varies with the number of buffers when $c = 20$ms, $E[Z] = 20$ms and the buffer holding time is median(Y) ≈ 34ms. We observe that the mean number of additional retransmissions under FIFO–MH can at best be about 80% of the mean number of additional retransmissions under FIFO even for high data rates. This reduction in mean number of additional retransmissions is much less than the reduction observed when the retransmission interval is a constant (and set to 40 which is same as $E[Y]$).

- **Retransmission Interval from Round Trip Time Traces**

In order to obtain retransmission intervals that reflect the variation in round trip times from repair servers to receivers, we collected round trip time from traces of Internet delay. We
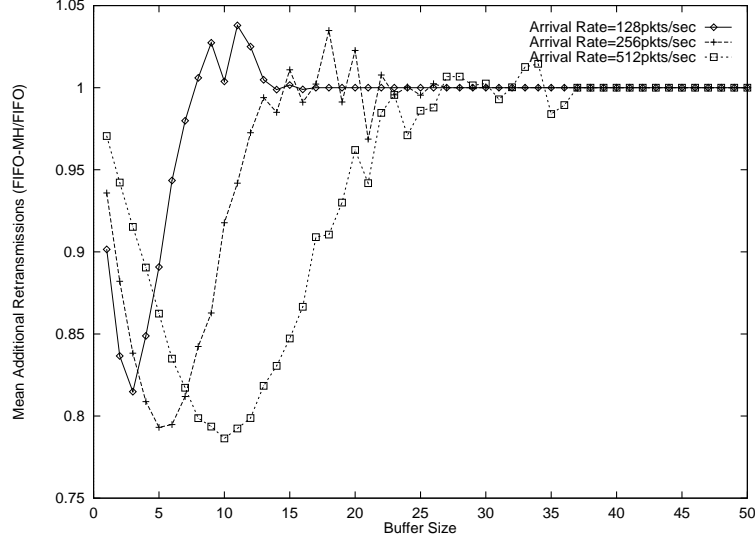
Figure 4.9. FIFO vs FIFO–MH

used these traces in conjunction with an idealized NAK suppression mechanism at the receivers to generate retransmission intervals. In our NAK suppression mechanism, among the receivers that lose a packet, only the receiver with the smallest round trip time from the repair server sends a NAK for that packet to the repair server. In this chapter, we do not worry about how such a NAK suppression can be achieved.

We collected round trip times from a host at UMass (repair server), to three hosts (receivers) at MIT, CMU and Georgia Tech by running simultaneous *ping* commands. Each trace consists of over 12,000 samples. The results in this section are based on these measurements.

Using these traces, we simulate the FIFO–MH policy. In the simulation, we use a sliding window estimate of the median of the retransmission intervals. In order to find a right size for the sliding window we find the mean number of additional retransmissions for different window sizes and for different values of packet arrival rates and buffer sizes under FIFO–MH. Figure 4.10 shows a sample of our experimental results. We observe that a window of size of 10 is appropriate for our traces.
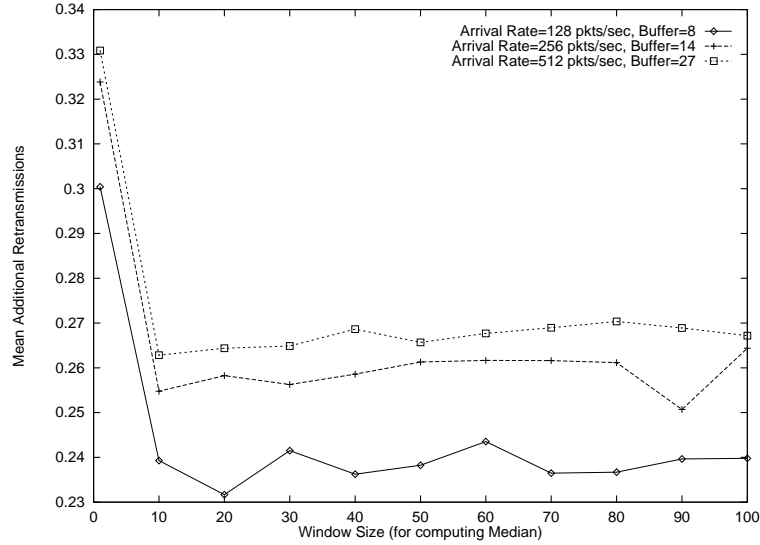
Figure 4.10. Mean Additional Retransmissions vs Window Size

Using a window size of 10, we find the running estimate of the median and use that as the buffer holding time under FIFO–MH and then compare the performance of FIFO–MH with FIFO. Figure 4.11 shows how the ratio of the mean number of additional retransmissions under FIFO–MH to that under FIFO varies as we increase the buffer size. We observe that the mean number of additional retransmissions under FIFO–MH is about 80% of the mean additional retransmissions under FIFO. This result is similar to the result when the retransmission interval has exponential variation.

In summary, the FIFO–MH buffer replacement policy can be used instead of simple FIFO to prevent premature overwriting of buffers and reduce the mean number of additional retransmissions only when the retransmission interval is constant. If the retransmission intervals are highly variable over a large range of values, then FIFO–MH does not significantly improve performance over FIFO. This is because a good buffer holding time cannot be determined in such a case. When the retransmission interval estimate is large, a large number of the packets will be discarded when the buffers are occupied by packets that are not lost downstream and are not required to be retransmitted. On the other hand, when the retransmission interval estimate is small, the packets (including those that are lost
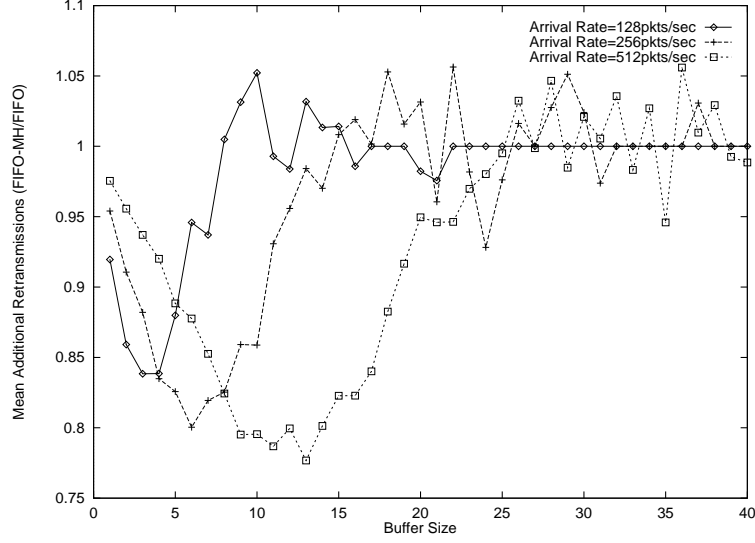
Figure 4.11. FIFO vs FIFO–MH

downstream and are required to be retransmitted) occupying the buffers will get replaced prematurely.

4.5.2 LRU policy

In the LRU policy, when the repair server buffers are all occupied, a new packet, on arrival at the repair server, replaces the packet that was least recently accessed. The LRU policy gives priority to packets that are retransmitted over those that are holding the buffers but are not "in demand." LRU is useful when the following two conditions are met.

- Packets are held in the buffer for sufficiently long time such that those packets whose retransmissions have not been requested can be assumed to have been successfully received and can be safely replaced.

- Lost packets require several retransmissions before they are successfully received by all receivers.

Figure 4.12 compares the performance of LRU with FIFO when the retransmission interval is a constant. In this figure $k = 10$, $p = 0.05$ and the retransmission interval is

109
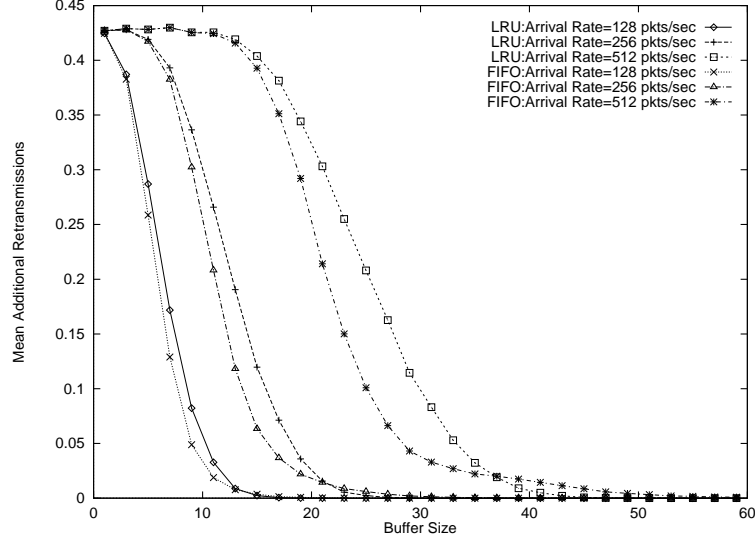
Figure 4.12. FIFO vs LRU, constant retransmission interval

40ms. We observe that when the number of buffers is small, the mean number of additional retransmissions under LRU is equal to or greater than the mean number of additional retransmissions under FIFO. LRU begins to perform better when the number of buffers is increase beyond a certain value. When the arrival rate is 512pkts/sec, LRU incurs lower mean number of additional retransmissions in comparison to FIFO when the buffer size is greater than 37. A similar behavior is observed, as shown in Figure 4.13, when we generate the retransmission intervals from our round trip time traces. From our observations we conclude that LRU cannot help in reducing mean additional retransmissions or improving buffer utilization over FIFO when the number of buffers is small.

4.6   Conclusions

In this chapter, we have studied the buffer requirements and buffer replacement policies at a repair server for providing reliable multicast. We used analytical models to show how repair server buffer requirements depend upon the packet arrival rate at the repair server, the required number of retransmissions, and the duration between retransmissions of a

110
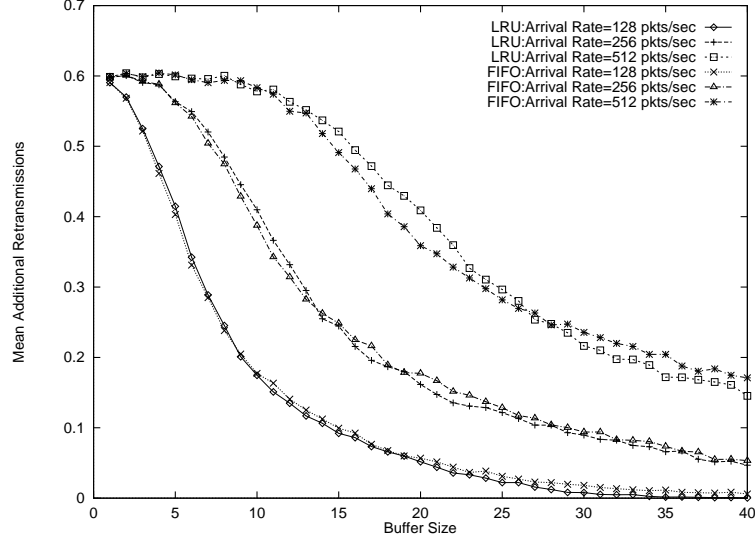
Figure 4.13. FIFO vs LRU, retransmission interval from measurements

packet. We found that buffer requirements are very sensitive to small values of the number of retransmissions of a packet within the packet inter–arrival time but increase very slowly with the number of receivers a repair server is responsible for supplying repairs.

We determined the effect of the number of repair servers and buffer size at a repair server on the end–system throughput and network bandwidth usage. We examined three buffer replacement policies; FIFO, FIFO–MH (that we proposed) and LRU. We showed that a simple FIFO buffer replacement policy performs nearly as well as FIFO–MH and even better than LRU when the number of buffers is small.

Our work can proceed in several directions. We have studied the buffer requirements of one multicast session in isolation. An extremely important future work will be to study the buffer requirements when several multicast sessions share the buffer resources at the repair server. It would also be interesting to extend our buffer study to other applications such as web caching to determine the memory requirements at web caches. In our work we have found how the buffer requirements depend on the various system parameters such as upstream retransmissions, loss probability, packet arrival rate, retransmission interval

111

and number of receivers. We would like to use the knowledge of these dependencies for studying the placement of repair servers in the multicast tree.

CHAPTER 5

RELIABLE MULTICAST USING ACTIVE REPAIR SERVICES

In Chapter 3, we presented a new server–based approach that co–locates designated repair servers with routers at strategic locations inside the network. An important issue in the server–based approach is that, in the Internet, static deployment of repair servers might not be useful due to the changing network conditions (e.g., route changes [58] and changes in loss behavior [79]). Given the performance benefits of the server–based approach, it is worth studying how to make it more dynamic and flexible. The required dynamism and flexibility can be provided by the recently proposed active services paradigm [2].

Active services advocate the dynamic placement of user–specified computations within the network, while preserving the routing and forwarding semantics of the current Internet architecture. Active services could be deployed as application–level programs inside routers or, equivalently, on severs co–located with routers. In our server–based local recovery context, the challenge is to provide the server functionality as a dynamically invocable/revocable active service.

While our focus in Chapter 2 was on the retransmission scoping problem and our focus in Chapter 3 was on a generic server–based approach and a study of its performance using analytical models, in this chapter, we present a detailed design of a complete multicast loss recovery architecture and protocol that use active repair services. We show how the challenges of NAK implosion, retransmission scoping and distribution of loss recovery burden can be flexibly and efficiently addressed using only minimal router support. We also show how active repair services could be located, and invoked/revoked through a simple signaling mechanism.

The rest of this chapter is structured as follows. In the next section we describe the design goals and principles for our reliable multicast architecture. In Section 5.2 we survey the existing reliable multicast architectures. In the earlier chapters our survey was specific to only certain aspects of these architectures. We find that none of the existing architectures meets all of our design goals and principles. In Section 5.3 we describe our network model. In Section 5.4 we describe our multicast loss recovery protocol called *Active Error Recovery*. This is followed by a description of a signaling mechanism for locating, and, invoking/revoking active repair services in Section 5.5. We identify the router support needed by our loss recovery protocol and signaling mechanism in Section 5.6. Our conclusions and directions for future work are described in Section 5.7.

5.1 Design Goals and Principles

Our main design goals are to avoid NAK implosion, to provide retransmission scoping and to efficiently distribute the loss recovery burden of the sender among other entities. An additional design goal is to avoid NAK implosion even when no active services are available. NAK implosion is a serious problem for the following reason(s). If R receivers see independent loss with probability p, the average number of receivers that do not receive a random multicast packet sent from the sender is Rp. The average number of receivers that do not receive a random packet increases when some of the loss is spatially correlated. Thus the number of NAKs sent to the sender for every lost packet increases linearly with the number of receivers. Therefore our design must avoid NAK implosion even when active services are not available.

We now describe the design principles that we use to meet our design goals.

- *Use active services for performance enhancements only:* The use of active services should be for performance enhancement only. Active service functionality should not be essential for the correct functioning of the reliable multicast transport protocols. This goal ensures that active services could be gradually deployed in the network and

114

that the reliable multicast protocols would continue to function correctly, although with inferior performance, even when some parts of the network do not have active services.

- *Require minimal router support:* Minimal router support should be required, while preserving the routing and forwarding semantics of the current Internet architecture.

- *Assume no multicast send capability at the receivers:* The receivers receiving multicast packets from the sender need not have the multicast send capability. We make this restriction for the following three reasons. First, in commercially available IP multicast a large amount of money is charged to the multicast sender [73]. Second, some networks, such as satellite networks, only support multicast in one direction. Third, setting up multicast distribution tree(s) for a large number of senders can be expensive [4, 20, 60].

- *Accommodate route asymmetry and route changes:* The multicast data path from the sender to the receivers may be different from the unicast paths from the receivers to the sender [4, 20]. In our context, this means that feedback from a receiver to the sender may take a different path than the path from the sender to that receiver. If active services that need receiver feedback for taking desired actions are to be invoked along the path from the sender to the receivers, this asymmetry must be handled. In the Internet, routes between end–hosts might change with time [58]. This requires that when there is a route change, active services must be revoked along the "older" path and invoked along the new path.

5.2   Related Work

We now present a survey of the existing reliable multicast architectures. Even though the existing architectures and protocols provide very useful design choices, we find that none meets all our design goals and principles.

115

Scalable Reliable Multicast, SRM [22], from Berkeley, uses timer–based NAK suppression but does not have any local recovery in its basic form. The local recovery enhancements proposed by SRM involves the dynamic selection of a receiver for supplying a repair for every loss in a local neighborhood as described in Chapter 3. NAKs and retransmissions are multicast within a local neighborhood using one of the following two mechanisms. One mechanism proposes the use of separate multicast groups for grouping receivers in a neighborhood for scoping NAKs and retransmissions within neighborhoods. In addition to carefully grouping receivers, such an approach requires the receivers to have the capability to send multicast (NAKs or retransmissions). The second mechanism uses an IP TTL–based scoping. There are two problems with TTL–based scoping. TTL–based scoping limits packets within a radius and is not suitable for tree structures. Also, it is hard to approximate a good TTL value.

Reliable Multicast Transport Protocol, RMTP [47], from Bell Labs, builds a logical hierarchy of receivers, designated receivers and the sender, on top of the physical multicast routing tree, with the sender as root and receivers as leaves of the tree. The designated receivers which are non–leaf nodes in the hierarchy help in aggregation of NAKs from the receivers and also provide local recovery. Apart from the complications due to building and maintaining logical trees, the problem with RMTP is that it does not clearly specify how to scope retransmissions between levels of the logical hierarchy. RMTP does achieves limited retransmission scoping by using router support to multicast retransmissions to subtrees of the multicast tree. In this approach a multicast retransmission, from the designated receiver with source address set to that of the multicast sender, is encapsulated in a unicast packet and sent to the nearest router in the multicast tree. The router then multicasts the retransmission downstream on the multicast subtree, also called *subcast*, below it as if the retransmission came from the sender. Such a subcast is not always useful because it goes to every node in the subtree. RMTP does not specify any means for obtaining the identity of the nearest router. Another problem with RMTP is that it will suffer from NAK implosion

if there are no designated receivers. The log–based reliable multicast (LBRM) approach [31] from Stanford is very similar to RMTP.

Active reliable multicast, ARM [43], from MIT, uses processing and buffering at routers for local recovery, NAK aggregation and retransmission scoping. ARM requires a very fine grained router support and control for selective forwarding of retransmissions only along those links that lead to receivers requiring the retransmission. One problem with ARM is that it will suffer from NAK implosion if routers do not have ARM functionality.

Pragmatic Generic Multicast, PGM [68], form CISCO, also uses fine grained router support for NAK aggregation and selective forwarding of retransmissions. The main problem with PGM is that it does not have local recovery. It does not distribute sender's burden of loss recovery. All lost packets must be recovered from the sender. The PGM architecture will also suffer from NAK implosion if there is sparse support for PGM inside the network. Even though PGM faces the above problems, it provides a very simple and elegant signaling mechanism for handling route asymmetry and route changes. This signaling mechanism also forms the basis of our signaling protocol for locating and invoking/revoking active repair services. The control–on–demand architecture from AT&T Labs [25] is very similar to the ARM architecture with one important difference. It supports probabilistic "filtering" of retransmissions as explained in Chapter 2.

Large Scale Multicast, LSM [55], from WUSTL, uses router support for scoping NAKs and retransmissions and allows hierarchical loss recovery. One potential problem with this approach is that in LSM, NAKs are multicast from the receivers on the multicast group to all the receivers and in the absence of routers with LSM support there will be a NAK implosion at not just the multicast sender but every receiver belonging to the multicast group. The problem of NAK implosion at the receivers could be handled by tunneling NAKs through LSM capable routers by using a PGM–like signaling mechanism.

Table 5.1 summarizes how some of the well–known multicast loss recovery approaches described above handle NAK implosion, loss recovery burden and retransmission scoping.

117

We see that none of these approaches meets all our design goals and principles which leads us to design a new reliable multicast approach.

	PGM	ARM	RMTP	SRM
NAK Implosion	aggregation, limited suppression	aggregation	aggregation	suppression
Loss Recovery Burden	sender	sender, routers	sender, designated receivers	sender, receivers
Retransmission Scoping	router controlled	router controlled	router subcast	TTL, multiple multicast groups

Table 5.1. Features of Existing Multicast Loss Recovery Approaches

5.3 Network Model

We now describe our network model. A sender multicasts data to a group of receivers. In addition, there are servers that are co–located at every router. These servers are assumed to offer a variety of active services including the active repair services in which we are interested. Not all servers offer all kinds of services. Hence an application interested in a certain kind of service must locate the service and invoke/revoke it based on its needs.

The above model differs from the model presented in Chapter 3. In Chapter 3, special purpose repair servers are co–located with routers at strategic locations inside the network. Here we have servers co–located with routers everywhere but the services on these servers must be invoked at strategic locations. For simplicity of presentation, we refer to the servers offering active repair services as *repair servers*. The repair servers have processing and buffering resources that are used to cache data for supplying repairs or retransmissions and to aggregate receiver feedback. Repair services are invoked only on those repair servers that lie along the multicast distribution tree of a multicast session. Figure 5.1 shows the
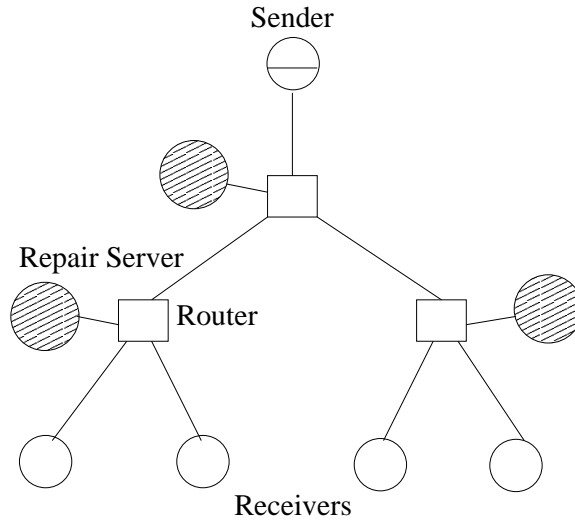
Figure 5.1. Repair Hierarchy

hierarchy of the sender, repair servers and receivers associated with a multicast distribution tree.

Once repair services are invoked at a repair server, the repair server joins the multicast group on which packets are sent to the receivers and cache a "recent" set of packets. These packets are retransmitted downstream on request as discussed in Chapters 3 and 4. The repair servers themselves recover lost packets from an upper level repair server or the sender. The repair servers also aggregate NAKs from downstream.

In the next two sections we describe a loss recovery protocol and a signaling mechanism for locating, invoking/revoking active repair services for the above network model. The loss recovery protocol and the signaling mechanism meet all our design goals and are based on our design principles.

5.4   Active Error Recovery Protocol

In this section we describe our multicast loss recovery protocol called *Active Error Recovery* (or AER in short). We present only a generic version of the AER protocol. The detailed specifications can be found in the Appendix B. Both AER and the signaling mech-
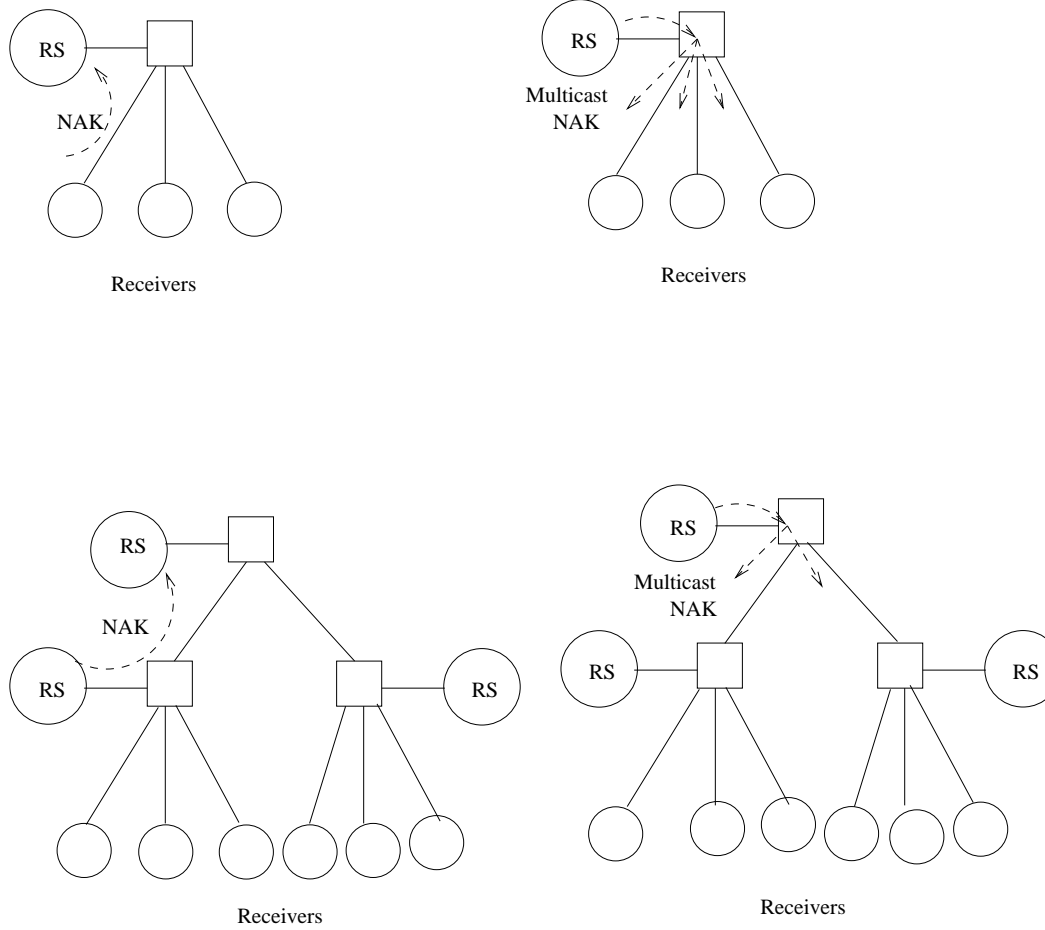
Figure 5.2. NAK Suppression Scenarios

anism presented in Section 5.5 use *subcast* which is defined to be multicast transmission from a repair server to the multicast subtree below it.

The AER protocol exhibits the following behavior:

- The sender multicasts packets to a multicast address that is subscribed to by all receivers and participating repair servers.

- On detecting a loss, a receiver (or repair server), after waiting for a random amount of time, unicasts a NAK to its closest upstream repair server (the manner in which the identity of the nearest repair server is determined is given in Section 5.5) and starts a NAK retransmission timer. If the receiver (or repair server) receives a NAK

120

for the same packet from its upstream repair server prior to sending its own NAK, it suppresses its own NAK transmission.

- If a repair server has the packet for which it received a NAK from a receiver or a lower level repair server, it subcasts the packet downstream. If the repair server does not have the packet, it immediately subcasts (multicasts on the subtree below the router connected to the repair server) the NAK downstream and begins the process of obtaining the packet from the sender (or its upstream repair server) if it has not already done so (as described in the previous item). AER only subcasts a NAK to the next downstream repair servers (or receivers, if there are no downstream repair servers). A subcast repair packet is also intercepted by the next downstream repair servers but forwarded further down the tree only if there is a pending NAK for that repair.

- The expiration of the NAK retransmission timer at a repair server, (or a receiver) without prior reception of the corresponding repair, serves as the detection of a lost packet for the repair server (or the receiver) and a NAK is retransmitted.

- On receiving a NAK from a downstream repair server, the sender remulticasts the requested packet to all the receivers and repair servers. As mentioned above, these repairs are intercepted by intermediate repair servers and forwarded further down the tree only if there is pending NAK for that repair.

Figure 5.2 exhibits two NAK suppression scenarios. Note that NAKs are unicast upstream but multicast downstream.

In summary, the AER protocol addresses the challenge of NAK implosion by using random timer–based NAK suppression and by using the repair server hierarchy for NAK aggregation. Even though NAKs can be aggregated by repair servers, NAK suppression is still used to ensure that NAK implosion does not happen when some part or whole of the distribution tree is without repair servers. AER distributes the burden of loss recovery

121

among the repair servers. It scopes retransmissions with the help of subcasting and interception of retransmissions.

- **Performance of AER**

The performance of AER is almost the same as L1. For the loss model presented in Chapter 3 the only difference in performance of AER and L1 is due to the fact that in AER, NAKs are first unicast upstream to a repair server and then "reflected" downstream for NAK suppression. This causes one additional traversal of the link(s) on which the NAK is unicast. The receiver or repair server that sends the NAK must also process one extra NAK, the one that gets reflected downstream. This will increase the network bandwidth usage and processing costs only by a negligible amount. As L1 has been shown to perform better than SRM–like and RMTP–like protocols, we argue that AER enjoys a similar superiority.

The performance of ARM [43] is likely to be better than AER because ARM uses very fine granular router support to allow retransmissions to be sent only on those router interfaces from which NAKs are received. Requiring router support that requires change in data forwarding runs contrary to our design principles, as noted in Section 5.1.

5.5   Signaling Mechanism

In this section we present a simple signaling mechanism for the following three purposes; locating repair services, invoking/revoking repair services, and handling route asymmetry and route changes. The signaling mechanism can be described as follows.

- The sender periodically multicasts "source path messages" or SPMs in short (similar to source path messages of PGM [68] and path messages of RSVP [81]) on the same multicast group on which the data is sent to the receivers. Each SPM contains a field for storing the originator's IP address (which we will see shortly is not typically the sender's IP address).
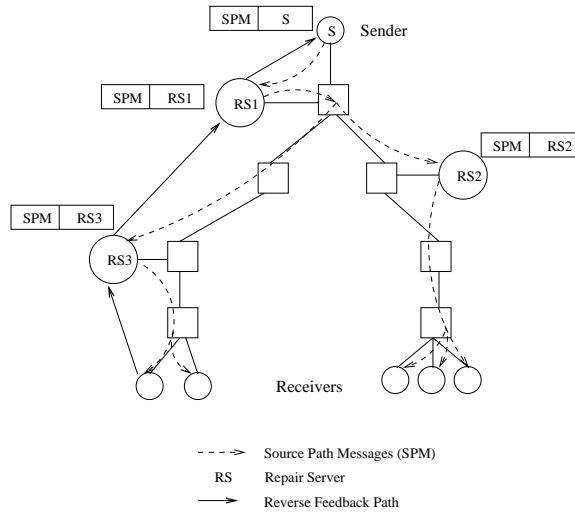
Figure 5.3. Source Path Message

- SPMs are intercepted by routers, with attached repair servers, along the path from the sender to the receivers and sent to the attached repair server. The router support required for interception of SPMs for them forwarding to the attached repair server is described in Section 5.6.

- On receiving an SPM, a repair server notes down the IP address of the previous repair server or the sender. This value is carried in the SPM. A repair server then replaces that address by its own address and subcasts the SPM downstream.

- On receiving an SPM, a receiver records the IP address of the repair server in the SPM.

The above signaling mechanism establishes reverse paths from the receivers to the sender through the repair servers. Establishing reverse paths is essential because the multicast data path from the sender to a receiver could be different from the unicast data path from the receiver to the sender on which NAKs are sent. In order to to be able to send NAKs to recover lost packets, a receiver or a repair server needs to know the identity of the nearest upstream (*w.r.t.* itself) repair server that is on the multicast data path. Once the

reverse path is established, a receiver or a repair server can send NAKs for lost packets to its nearest upstream repair server (which could be the sender). Figure 5.3 demonstrates the establishment of the reverse path from the receivers to the sender using SPMs.

Repair servers advertise themselves through the SPMs by including their IP address in the SPMs before subcasting the SPMs down the multicast tree. If for some reason a repair server does not wish to advertise itself, or wishes to unadvertise itself after having advertised itself earlier, it simply subcasts the SPM it receives without including its own IP address in the SPM.

SPMs are sent periodically from the sender to deal with route changes. Subsequent to a route change, the SPMs are sent on a different multicast tree (from the tree before the change). The new multicast tree involves new repair servers. Once again these new repair servers advertise their presence in the SPMs and the repair servers and the receivers update the address of their nearest repair servers. This allows establishment of new reverse paths from the receivers to the sender through the repair servers. Periodic SPMs also allow repair servers to unadvertise themselves as stated above.

SPMs are also used for invoking repair services at repair servers along the path from the sender to the receivers. Those repair servers which advertise themselves in the SPMs, also start repair services. When there is a route change, no SPMs are received by the repair servers along the old path. These repair servers, after waiting for a certain specified amount of time, revoke the repair services and free up all the resources that are used by the repair services.

In summary, the signaling mechanism presented above provides a very simple way of locating, invoking/revoking repair services while flexibly handling route asymmetry and route changes.

5.6   Router Support

We now look at the new functionality required at routers to support AER and the above signaling mechanism. New functionality is needed at the routers to support subcast and interception of SPMs, repairs, and NAKs.

One possible approach for achieving subcast and the interception of SPMs, repairs, and NAKs is to use a separate multicast group, for each repair server, which all the next level downstream repair servers (or receivers) must join. Subcast packets (SPMs, repairs, and NAKs) are sent by repair servers on these multicast groups and received by the next level downstream repair servers or receiver. Even though this approach does not require any additional router support, it requires a large number of multicast groups and their management and hence is not practical.

We propose to add the following simple functionality at routers for subcast and interception of packets without requiring any changes in the forwarding mechanisms of the router.

5.6.1   Subcast support

Subcast can be achieved using IP encapsulation [47] (also shown in Figure 5.4) as follows. A subcast packet is sent unicast to the router co–located with the repair server. This unicast packet encapsulates an IP packet header that contains the IP address of the original multicast sender in the source address field and the multicast group address in the destination address field. On receiving the unicast from the repair server, the co–located router multicasts the encapsulate packet on the multicast subtree below it as if it was from the original sender. The router needs to provide the functionality to multicast the encapsulated packet downstream.

5.6.2   Interception of SPMs, NAKs, and Repairs

A router must distinguish between packets that need attention of the co–located repair servers such as the SPMs, NAKs and repairs, and the packets that need to be simply for-
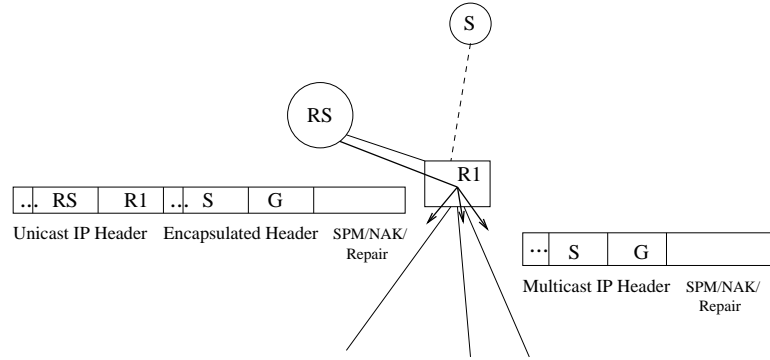
Figure 5.4. Subcast Using IP Encapsulation

warded. The IP protocol header allows certain optional fields that can be used in an IP packet to get per hop attention as it is forwarded from the packet source to destination. These optional fields are called IP options. A *router alert* IP option (defined in IPv6 recommendations [9]) can be used in a packet to "get the attention" of a router. The routers that recognize this option will send the packet containing this option to the co–located repair server. Routers that do not recognize this option or do not have any co–located repair servers simply forward this packet. Packets without the IP option are forwarded by all routers as usual.

SPMs sent on a multicast group need to be intercepted at routers with attached repair servers even before repair services are invoked and before the repair server has joined the multicast group. We define a new IP router alert option called an SPM option. When a router co–located with a repair server receives an IP packet with the SPM option, it forwards the packet to the repair server. The only state that the router needs to maintain is the address of the interface that leads to the repair server.

The AER protocol requires subcast NAKs and repairs to be intercepted at the next downstream routers that have co–located repair servers and sent to the repair servers. We define another IP router alert option called Repair–NAK that is included in the repairs and the NAKs and is used for identifying a repair or NAK as a packet requiring special attention

126

at a router with co–located repair server. The router with a co–located repair server gives special attention to a packet with the Repair–NAK option but unlike in the case of SPMs it intercepts and sends the packet to the repair server only if the repair server is a member of the multicast group on which the NAKs and repairs are sent. If the repair server is not a group member, then the router simply forwards the packet downstream. A router can easily check if the co–located repair server is a group member as follows. By maintaining the address (say A) of the interface that leads to the repair server, the router can check the repair server's membership by looking for the presence of A inside the routing table entry for the multicast group address. Again, as in the case of SPMs, the only additional state an active router needs to maintain is the address of interface leading to the repair server.

5.7   Conclusions

In this chapter we have presented the design of a reliable multicast architecture and protocol that uses active repair service. We showed how the challenges of NAK implosion, retransmission scoping and distribution of loss recovery burden can be flexibly and efficiently addressed requiring only minimal router support. We also showed how active repair services can be located, invoked/revoked through a simple signaling mechanism.

We have presented a very simple approach for invoking and revoking active services. In our approach repair services are invoked by the source path messages and get revoked via a soft state timeout if no source path messages are received for sufficient period of time. There is no provision of revoking repair services based on changes in the network loss behavior. Another limitation of our architecture is that the repair services are embedded in the AER protocol. An important future work would be to design an architecture that makes the active services available as composable services which can be invoked/revoked by different protocols and applications as warranted by their requirements and network conditions.

CHAPTER 6

SUMMARY AND FUTURE WORK

6.1   Summary of the Dissertation

In this dissertation, we have designed and evaluated multicast loss recovery architectures and protocols that make efficient use of both the network and end–resources, and scale to applications that can potentially have several thousand receivers spanning wide area networks.

One of the important problems in large scale multicast loss recovery is that of a receiver receiving unwanted retransmissions of packets lost at other receivers. In Chapter 2, we have examined an approach for providing retransmission scoping for reliable, scalable multicast communication by using multiple multicast channels for recovery of lost packets. In this approach, rather than requiring that all receivers receive all retransmitted packets (regardless of whether a given receiver had already received a given packet correctly), the additional multicast channels are used to allow only those receivers that actually *want* a particular packet to actually receive that packet. We considered the idealized case of an infinite number of multicast channels as well as the more realistic scenario of using a small, fixed number of multicast channels. We also considered two different models of sender behavior: the one–to–many scenario and the many–to–many scenario. Our analytic models have demonstrated that significant performance gains (in terms of reduced receiver overhead, and a reduced overall protocol overhead in the case of many–many communication) can be realized in such environments, over a range of system parameters.

We discussed two mechanisms for implementing multiple multicast channels, one using multiple IP multicast groups and the other using a control–on–demand active networking

approach. With current IP multicast support, we obtain savings in receiver processing through a local filtering scheme but cannot save on network bandwidth, whether or not we move the filtering point inside the network. However, by using some processing at routers and a small buffer space (up to 200 bytes per multicast session), we can save on both sender and receiver processing costs as well as network bandwidth.

Another problem arises when the sender alone bears the burden of handling loss–feedback and supplying retransmissions to a large group of receivers. Local recovery approaches, in which entities other than the sender aid in loss recovery, distribute the loss recovery burden and also reduce network bandwidth consumption and recovery latency. In Chapter 3, we proposed a new server–based local recovery approach in which designated hosts called repair servers are co–located with routers at strategic locations inside the network. These repair severs cache packets to provide the ability to quickly and efficiently recover from loss at the point of loss. They also exploit the physical hierarchy of multicast trees for feedback aggregation. The caching of packets and subsequent loss recovery from the designated servers helps in distributing the loss recovery burden among the sender and the designated servers. We compared the performance of our server–based approach with two traditional receiver–based approaches, in which loss recovery only involved the participating receivers and the sender. Using analytical models, we demonstrated the performance gains in using the server–approach in terms of end–system throughput, network bandwidth and overall processing costs. The performance gains increase as the size of the network and the loss probability increase, making the server–based approach more scalable with respect to these parameters. In Chapter 3 we also estimated the processing power required at the repair servers per multicast session for achieving the performance gains.

The server–based approach will perform efficiently only when sufficient buffering resources are available at the servers. In Chapter 4, we studied the buffer requirements and buffer replacement policies at a repair server for providing reliable multicast. We used analytical models to show how repair server buffer requirements depend upon the packet

arrival rate at the repair server, the required number of retransmissions, and the time be-
tween retransmissions of a packet. We found buffer requirements to be very sensitive to
small values of the number of retransmissions of a packet within the packet inter–arrival
time but increase very slowly with the number of receivers to which a repair server is re-
sponsible for supplying repairs. We determined the effect of the number of repair servers
and buffer size at a repair server on the end–system throughput and network bandwidth us-
age. We examined three buffer replacement policies; FIFO, FIFO–MH (that we proposed)
and LRU. We showed that a simple FIFO buffer replacement policy performs nearly as well
as FIFO–MH and even better than LRU when the number of buffers is small.

An important issue in server–based local recovery is that *static* deployment of servers
might not be useful due to changing network conditions (e.g., route changes [58], or
changes in loss behavior [79, 80]). In order to substantially benefit from the server–based
approach, it is important to make it more dynamic and flexible. The required dynamism
and flexibility could be provided using the recently proposed *active* services [2] paradigm.
In Chapter 5, we designed a multicast loss recovery architecture and protocol that uses
active repair services and showed how the challenges of NAK implosion, retransmission
scoping and distribution of loss recovery burden can be handled flexibly and efficiently
with minimal router support. We also showed how active repair services could be located
and invoked/revoked through a simple signaling mechanism.

6.2   Future Research Directions

In this section we indicate several promising areas of future work some of which follow
naturally from the work in this dissertation.

6.2.1   Modeling the Cost of Additional Network Resources

In our research, we have proposed several approaches that require additional support,
other than packet forwarding, from the network. The mechanism for implementing multi-

ple multicast channels using active networking requires processing resources for processing NAKs and "filtering" retransmissions, and memory to keep NAK state. The server–based and active–services–based local recovery approaches require processing and buffering resources at the repair server for caching packets, recovering from loss, receiving retransmission requests and supplying repairs. Some router resources are also required for subcast and interception of NAKs, retransmissions and SPMs. A very interesting and important avenue of future work is to model the cost of these additional resources used inside the network so as to study the performance benefits due to these resources as a function of their cost.

6.2.2   Buffer Requirements

We have studied the buffer requirements, at a repair server, of one multicast session in isolation. An extremely important future work will be to study the buffer requirements when several multicast sessions share the buffer resources at the repair server. It would also be interesting to extend our buffer study to other applications such as web caching to determine the memory requirements at web caches. The cache–access patterns in a web cache application are likely to be different from the retransmission requests received at the repair server.

In our work we have shown how the buffer requirements depend on the various system parameters such as upstream retransmissions, loss probability, packet arrival rate, retransmission interval and number of receivers. We would like to use the knowledge of these dependencies for studying the placement of repair servers or invocation of repair services in the multicast tree.

6.2.3   Composable Multicast Services

During our research on reliable multicast, we have identified a set of active services that can be used to enhance the performance of end–to–end multicast transport protocols. These services include repair service [37], feedback aggregation service for loss feedback

aggregation/congestion feedback aggregation [39], log service to allow recovery of *old* data [31], data rate conversion service [49], parity encoding service [40, 52, 66]. There services are not required for correctness of the protocols, but when invoked, result in higher performance. An interesting and important topic for future work will be the design of an architecture, that instead of embedding a specific set of services in a specific protocol, makes them available as *composable active services*. By composable active services, we mean individual active services, placed at various locations inside the network, one or more of which can be dynamically and flexibly combined with different end–to–end transport protocols to enhance application and network performance. These services are used based upon application requirements and network conditions. A signaling mechanism to dynamically invoke/revoke services and a measurement–based infrastructure to react to changing network conditions will be two important components of such a composable services architecture.

6.2.4 Other Multicast Research

Even though IP multicast has been around for almost a decade now, it is yet to be deployed widely. Some of the major barriers are the complex nature of multicast routing protocols, the lack of effective congestion control mechanisms for multicast transport and the lack of an appropriate pricing structure for multicast. There is an immediate need to revisit the IP multicast service model to possibly simplify it and develop simple routing protocols (or simplify existing ones) that can be easily understood and deployed. The recent work reported in [32, 60] is an important step in this direction. With regard to multicast congestion control, the current proposal of adjusting the sender rate [6, 26, 27, 76] based on the worst receiver rate can only be considered as a good starting point. Such a scheme is unfair to receivers who can receive data at higher rates. An approach that *dynamically* groups receivers based on their capabilities (and capabilities of network links leading to them) must be studied more carefully. Billing for multicast is a tricky problem

[28]. Between the sender and the receivers, who should be charged how much remains an open question.

APPENDIX A

DESCRIPTION OF PROTOCOLS N1 AND N2

A.1   Protocol N1

The protocol N1 [71] exhibits the following behavior

- the sender multicasts all packets,

- whenever a receiver detects a lost packet, it transmits a NAK to the sender over a
  point-to-point channel and starts a timer,

- the expiration of the timer without prior reception of the corresponding packet serves
  as the detection of a lost packet.

A.2   Protocol N2

The protocol N2 [71] exhibits the following behavior

- the sender multicasts all packets and state information

- whenever a receiver detects a lost packet, it waits for a random period of time and
  then multicasts a NAK to the sender and all other receivers, and starts a timer,

- upon receipt of a NAK for a packet which a receiver has not received, but for which
  it initiated the random delay prior to sending a NAK, the receiver starts the timer and
  behaves as if it had sent that NAK,

- the expiration of the timer without prior reception of the corresponding packet serves
  as the detection of a lost packet

APPENDIX B

AER PROTOCOL SPECIFICATIONS

We now describe the AER protocol specifications in terms of *use cases*. Use cases [34] are a narrative dynamic operational description of system or protocol operations.

B.1 Sender Protocol

State Information:

$nextSeq$ - The sequence number to use for the next original data packet sent.

B.1.1 Instantiation of the Protocol

This use case begins when the sender protocol is instantiated (by upper layers). The protocol data structures are initialized and endpoints (e.g., sockets) are opened for transmission of data packets and reception of NAK packets. $nextSeq$ is initialized to 1. A number of source path message (SPM) packets (e.g, 10) are formed and sent to the multicast group address (obtained from the entity that instantiates the protocol). The SPM packet format is described in Appendix A. Each SPM packet contains the IP address of the sender, the multicast group address and port number, and the current highest data packet sequence number of the transmitted data packets, which is equal to nextSeq minus 1. A highest data packet sequence number of 0 indicates that no data packets have been transmitted by the sender. The SPM timer is started after sending out the SPM packets. The duration of the SPM timer is set to some constant value (e.g., 4 seconds). This use case ends when the protocol is taken to a wait state (waiting for the events described in Use CasesB.1.2 – B.1.5).

B.1.2 SPM Timer Service Routine

This use case begins when the SPM timer expires. An SPM packet is transmitted to the destination multicast address. The SPM packet also contains the current highest data packet sequence number sent, which is equal to $nextSeq - 1$. This use case ends when the SPM timer is reset.

B.1.3 Sending Data

This use case begins when data (a variable length data block) is received from the upper layer. This data is placed in one or more data packets. Each data packet is assigned a sequence number of $nextSeq$, then $nextSeq$ is incremented for the next data packet. All data packets are then buffered in a retransmission buffer. The current sender NAK count (maximum value of NAK count seen in NAK packets so far) for each data packet is initialized to 0. These data packets are then sent out to the multicast group. The data packet format is described in Appendix A. This use case ends when the data packet transmission(s) are completed.

B.1.4 Processing a Received NAK Packet

This use case begins when a NAK packet is received from the lower layer. The NAK packet contains the sequence number of the data packet whose retransmission has been requested. It also contains a NAK count for that data packet. The NAK packet format is shown in Appendix A. If the NAK packet NAK count is greater than the current sender NAK count of that data packet in the retransmission buffer, then the requested data packet is fetched from the retransmission buffer, the current sender NAK count for the data packet is set to the NAK packet NAK count and the data packet is sent out to the multicast address. Otherwise, the NAK packet is ignored and no retransmission is sent out. This use case ends when the data packet retransmission has been initiated or the NAK packet is ignored.

B.1.5 Processing a Stop

This use case begins when a stop protocol request is received from the upper layer. All of the resources allocated to the protocol layer are freed. This use case ends when the resources are freed.

B.2 Receiver Protocol

State Information:

$repairServer$ - The address of the repair server for the receiver.

$readNextSeq$ - The sequence number of the last data packet delivered to the application.

B.2.1 Instantiation of the Protocol

This use case begins when the receiver protocol is instantiated (by upper layers). The protocol data structures are initialized and endpoints (e.g., sockets) are opened for receiving transmissions and retransmissions of data and NAK packets, and also for sending NAK packets. This is followed by a multicast join operation for joining the multicast group on which data and NAK packets are expected. The multicast address and port numbers are received from the entity that instantiates the protocol. $repairServer$ and $readNextSeq$ are both initialized to 0. This use case ends when the protocol is taken to a wait state where it waits to receive data, NAK or SPM packets and also waits for expiration of timers as described below.

B.2.2 Processing a Received SPM Packet

This use case begins when an SPM packet is received. The IP address of the nearest repair server is contained in the SPM packet. This address is stored in repairServer. The SPM packet also contains the current highest data packet sequence number. If this sequence number is equal to zero, then it is ignored, Otherwise, this this sequence number is used to detect any gaps of missing packets by calling Use Case B.2.4. This use case ends when Use Case B.2.4 is called.

B.2.3   Processing a Received Data Packet

This use case begins when a data packet is received. Any NAK retransmission timer or NAK suppression timer that might have been started for this data packet is canceled. If this data packet has not already been received and contains data that has not already been passed up to the application (seen by comparing the data packet sequence number with readNextSeq), then the data packet is buffered at the receiver and Use Case B.2.8 is called. Otherwise the received data packet is discarded. This use case ends when the received data packet is discarded or when it is buffered and Use Case B.2.8 is called.

B.2.4   Detecting Gaps in Data Packet Sequence Numbers

This use case begins when gaps in received data packet sequence numbers must be detected and handled. The data packet buffer is searched backward, starting with either the sequence number from a received data packet or the sequence number plus one from an SPM packet, for data packets that have not been received and have not had the data packet recovery process started. If a data packet is missing and has not had the recovery process started (the receiver NAK count for the data packet is zero), then the receiver NAK count for the data packet is initialized to 1 and Use Case B.2.9 is called for the missing data packet sequence number. The search continues backwards through the buffer, one data packet at a time, until a data packet is reached that has either been received and buffered, has already had the data packet recovery process started, or has been delivered to the application (seen by comparing the data packet sequence number with readNextSeq). This use case ends when all data packets in the last gap have had the recovery process started.

B.2.5   NAK Suppression Timer Service Routine

This use case begins when the NAK suppression timer for a missing data packet expires. If either the data packet in question has arrived or the receiver NAK count is greater than some fixed value (e.g., 48), then the timer is ignored. Otherwise, a NAK packet for the data packet containing the receiver NAK count is unicast to the nearest repair server using the

repairServer address and a NAK retransmission timer for the data packet is started. The duration of the NAK retransmission timer is some fixed value (e.g., 1.75) times the round trip time between the receiver and the source. This use case ends when either the timer is ignored or a NAK retransmission timer for the data packet is started.

B.2.6 NAK Retransmission Timer Service Routine

This use case begins when the NAK retransmission timer for a missing data packet expires. If the data packet in question has arrived, then the timer is ignored. Otherwise, the receiver NAK count for the data packet is incremented by 1 and Use Case B.2.9 is called. This use case ends when either the timer is ignored or Use Case B.2.9 is called.

B.2.7 Processing a Received NAK Packet

This use case begins when a NAK packet is received. If the NAK packet is for a data packet that has been received and either buffered or delivered to the application (seen by comparing the data packet sequence number with $readNextSeq$), then the NAK packet is ignored. Otherwise, the NAK packet NAK count is compared with the receiver NAK count for the data packet in question. If the NAK packet NAK count is greater than or equal to the receiver NAK count for the data packet, then the receiver NAK count for the data packet is set to the NAK packet NAK count, any NAK suppression timer or NAK retransmission timer for the data packet is canceled, and a NAK retransmission timer for the data packet is started as described in use case B.2.5. This use case ends when either the NAK packet is ignored or the NAK retransmission timer is started.

B.2.8 Processing a Buffered Data Packet

This use case begins when a received data packet is buffered. First, use case B.2.4 is called to start the recovery process for any missing data packets. If the application is using an asynchronous receive mode, then the data packet buffer is checked for any data that can be delivered to the application. This "Segmentation and Reassembly" is done using

readNextSeq along with the "First Segment" and "Segmentation" flags in the data packet headers. If data is delivered to the application, $readNextSeq$ is updated. This use case ends when either use case B.2.4 returns or when data is delivered to the application.

B.2.9   Start Suppression Interval

This use case begins when a NAK suppression interval is to be started for a missing data packet sequence number. A NAK suppression timer is started for the data packet with a duration that is a random variate, uniformly distributed between 0 and an upper limit (e.g., 1.5), times the maximum round trip time between the next upstream repair server (which may be the sender) and the other receivers or repair servers serviced by that entity. This use case ends when the NAK suppression timer is started.

B.2.10   Processing a Stop

This use begins when a stop protocol request is received from the upper layer. All of the resources allocated to the protocol layer are freed. This use case ends when the resources are freed.

B.3   Repair Services Protocol

State Information:

$repairServer$ - The address of the repair server for this repair server.

$maxSeqRcvd$ - The maximum data packet sequence number received.

B.3.1   Instantiation of the Protocol

This use case begins when the active node receives the first SPM packet. The active node searches for the repair server object for the source address and multicast group and, not finding any, creates a new repair server object for the stream. The source address and multicast group are extracted from the SPM packet and stored by the repair server. The repair server object is added to the active node cache using the source address and

140

multicast group. $maxSeqRcvd$ is initialized to 0. Use case B.3.3 is then called for further SPM packet processing. This use case ends when use case B.3.3 returns.

B.3.2 SPM Wait Timer Service Routine

This use case begins when the SPM wait timer expires. This event indicates that either no packets are being sent by the sender anymore on this multicast group or the routes have changed. All timers for the repair server are canceled, all buffer resources for the repair server are freed and the repair server object is removed from the acitve node cache. This use case ends when the protocol state is destroyed.

B.3.3 Processing a Received SPM Packet in the Wait State

This use case begins when an SPM packet is received in the wait state. The repair server cancels any current SPM wait timer and starts a new one using a fixed duration (e.g., 60 seconds). The maximum data packet sequence number transmitted by the sender is contained in the SPM packet. Use case B.3.5 is called using this sequence number to detect missing data packets. $repairServer$ is set to the address of the nearest repair server contained in the SPM packet. The repair server then places its own address in the SPM packet and multicasts it downstream. This use case ends when the the SPM packet is multicast downstream.

B.3.4 Processing a Received Data Packet

This use case begins when a data packet is received by the repair server. The data packet is stored in the repair server buffer, if it is not already present there, based on a certain buffer management policy (e.g., limited size FIFO). Any NAK suppression or NAK retransmission timer for the data packet are canceled. If the data packet is a retransmission and there is a pending NAK for this data packet, then this data packet is also retransmitted (multicast) downstream. Use case B.3.5 is called using this data packet's sequence number

to detect missing data packets. $maxSeqRcvd$ is updated if this data packet's sequence number is larger. This use case ends when $maxSeqRcvd$ is updated if required.

B.3.5 Detecting Gaps in Data Packet Sequence Numbers

This use case begins when either a data packet (say sequence number p_i) or SPM packet (sequence number p_i is equal to the maximum sequence number sent plus one) is received and lost data packet must be detected. If $maxSeqRcvd$ is equal to zero, then this use case ends. If p_i is greater than the maximum data packet sequence number received ($maxSeqRcvd$) by two or more, then use case B.3.6 is called for each data packet sequence number between $maxSeqRcvd$ and p_i that either does not have any repair server NAK state (the NAK state containing a NAK pending flag, a NAK count and a Downstream NAK count) or has NAK state with the NAK pending flag clear. This use case ends when either Use Case B.3.6 is called for each missing data packet or no action is taken.

B.3.6 Starting Data Packet Recovery

This use case begins when a data packet (say sequence number p_i) is detected as missing and the data packet recovery process must be started. Any NAK suppression or NAK retransmission timer for data packet p_i is canceled. If no NAK state exists for data packet p_i, then NAK state is created. The NAK state NAK pending flag is set, and the NAK state NAK count and Downstream NAK count are set to 1. A NAK packet for data packet p_i is created and multicast downstream. A NAK suppression timer is created for data packet p_i with a duration that is a random variate, uniformly distributed between 0 and an upper limit (e.g. 1.5), times the maximum round trip time between the next upstream repair server (which may be the sender) and the other receivers and repair servers serviced by that entity. This use case ends when a NAK suppression timer for data packet p_i is started.

B.3.7 Processing a Received Multicast NAK Packet from Upstream

This use case begins when a multicast NAK packet is received for a data packet (say sequence number p_i) from upstream. If a NAK suppression or NAK retransmission timer for data packet p_i has been started, it is canceled. If no NAK state exists for data packet p_i, then a new NAK state is created with the NAK pending flag cleared, the repair server NAK count set to the NAK packet NAK count, and the Downstream NAK count set to zero. If NAK state already exists and the NAK packet NAK count is greater than the repair server NAK count for data packet p_i, the repair server NAK count for data packet p_i is changed to the NAK packet NAK count. A NAK retransmission timer is started for the data packet with a duration of some fixed value (e.g., 1.75) times the round trip time between this repair server and the sender. This use case ends when the NAK retransmission timer is started.

B.3.8 Processing a Received Unicast NAK Packet from Downstream

This use case begins when a unicast NAK packet is received for a data packet (say sequence number p_i) from downstream. If no NAK state exists for data packet p_i, then a new NAK state is created with the repair server NAK count and Downstream NAK count (described below) both set to 0 and the pending NAK flag set. If the NAK packet NAK count is greater than the repair server NAK count for data packet p_i, then use case C.9 is called. Otherwise, the NAK packet NAK count is compared to the repair server Downstream NAK count for data packet p_i. The repair server Downstream NAK count is the highest NAK count seen by the repair server from unicast NAK packets for data packet p_i sent from downstream. It is used only for the purpose of preventing unnecessary downstream multicasts of NAK packets for NAK suppression. If the NAK packet NAK count is greater than the repair server Downstream NAK count for data packet p_i, then the repair server Downstream NAK count for data packet p_i is set to the NAK packet NAK count and the NAK packet is multicast downstream for the purpose of NAK suppression. Otherwise

no action is taken. This use case ends when either use case B.3.9 is called, a NAK packet is multicast downstream or no action is taken.

B.3.9   Processing a New NAK Packet

This use case is called when the NAK count of a received unicast NAK packet for a data packet (say sequence number p_i) is greater than the repair server NAK count for data packet p_i. The repair server NAK count and Downstream NAK count for data packet p_i are both set to the NAK packet NAK count. If data packet p_i is available in the repair server buffer, then the pending NAK flag for data packet p_i is cleared, data packet p_i is retrieved from the buffer and it is multicast downstream. If data packet p_i is not available in the repair server buffer, then Use Case B.3.10 is called. This use case ends when either data packet p_i is multicast downstream or Use Case B.3.10 is called.

B.3.10   Start Suppression Interval

This use case begins when a NAK suppression interval is to be started for a missing data packet sequence number (say sequence number p_i) due to the arrival of a unicast NAK from downstream. Any NAK retransmission timer for data packet p_i is canceled and a NAK suppression timer is started for data packet p_i if it has not already been started. The duration of the NAK suppression timer is a random variate, uniformly distributed between 0 and an upper limit (e.g., 1.5), times the maximum round trip time between the next upstream repair server (which may be the sender) and the other receivers or repair servers serviced by that entity. The NAK packet is multicast downstream in an attempt to suppress any further NAK packets for data packet p_i. This use case ends when the NAK packet has been multicast downstream.

B.3.11   NAK Suppression Timer Service Routine

This use case begins when the NAK suppression timer for a data packet expires. A NAK packet for the data packet is then sent to the nearest upstream repair server (which

may be the sender) with the NAK packet NAK count set to the repair server NAK count for the data packet being NAKed. A NAK retransmission timer is started for the data packet with a duration of some fixed value (e.g., 1.75) times the round trip time between this repair server and the sender. This use case ends when the NAK retransmission timer is started.

B.3.12   NAK Retransmission Timer Service Routine

This use case begins when the NAK retransmission timer for a data packet expires. The repair server NAK count for the data packet is incremented by 1. A NAK packet for the data packet is multicast downstream with the NAK packet NAK count set to the repair server NAK count for the data packet. A NAK suppression timer is started for the data packet with a duration of a random variate, uniformly distributed between 0 and an upper limit (e.g., 1.5), times the maximum round trip time between the next upstream repair server (which may be the sender) and the other receivers and repair servers serviced by that entity. This use case ends when the NAK suppression timer is started.

B.4   Packet Formats

1. *SPM:* Version Number, Packet Type, Multicast Group Address, Port number, Repair Server Address, Maximum Transmitted Data Packet Sequence Number. The IP header of an SPM contains the SPM option.

2. *Data Packet:* Version Number, Packet Type, Sequence Number, Timestamp, First Segment Flag, Segmentation Flag, Retransmission Flag, Payload.

3. *Unicast NAK:* Version Number, Packet Type, Data Packet Sequence Number, NAK Count, Fast Repair Flag, Source Address

4. *Repair Packet:* Same as Data Packet. In addition, the IP header contains the repair/NAK option.

5. *Multicast NAK:* Same as Unicast NAK Packet. In addition, the IP header contains the repair/NAK option.

BIBLIOGRAPHY

[1] Ammar, M., and Wu, L. Improving the Throughput of Point–to–Multipoint ARQ Protocols Through Destination Set Splitting. In *Proceedings of IEEE Infocom*, pages 262–269, June 1992.

[2] Amir, E., Mccanne, S., and Katz, R. An Active Services Framework and it Application to Real Time Multimedia Transcoding. In *Proceedings of ACM SIGCOMM Conference*, September 1998.

[3] Alexender, D., Braden, R., Gunter, C., Jackson, A., Keromytis, A., Minden, G., and Wetherall, D. Active Network Encapsulation Protocol (ANEP). *RFC Draft*, July 1997.

[4] Ballardie, A., Francis, P., and Crowcroft, J. Core Based Trees. In *Proceedings of ACM SIGCOMM Conference*, 1993.

[5] Bhagawat, P., Mishra, P., and Tripathi, S. Effect of Topology in Performance of Reliable Multicast Communication. In *Proceedings of IEEE Infocom*, June 1994.

[6] Bhattacharyya, S., Kurose, J., and Towsley, D. The Lost Path Multiplicity Problem in Multicast Congestion Control. In *Proceedings of IEEE Infocom*, March 1999.

[7] Billhartz, T., Cain, J., Farrey-Goudreau, E., Fieg, D., and Batsell, S. Performance and Resource Cost Comparisons for the CBT and PIM Multicast Routing Protocols in DIS Environments. In *Proceedings of IEEE Infocom*, March 1996.

[8] Birman, K., Schiper, A., and Stephenson, P. Lightweight Causal and Atomic Group Multicast. In *ACM Transactions on Computer Systems*, Vol.9, No.3, pages 272–314, August 1991.

[9] Bradner, S., and Mankin, A. Recommendation for IPng. *RFC 1752*, January 1995.

[10] Calvert, K., Doar, M., and Zugera, E. Modeling Internet Topology. In *IEEE Communications Magazine*, June 1997.

[11] Casner, S. Frequently Asked Questions (FAQ) on the Multicast Backbone (MBONE). ftp://ftp.isi.edu:mbone/faq.txt.

[12] Chang, J., and Maxemchuk, N. Reliable Broadcast Protocols. In *ACM Transactions on Computer Systems*, Vol.2, No.3, pages 251–273, August 1984.

[13] Cheriton, D. *End–to–End mailing list*, September 1994.

[14] Cheung, S., Ammar, M., and Li, X. On the Use of Destination Set Grouping to Improve Fairness in Multicast Video Distribution. Tech Report: GIT-CC-95-25, Georgia Institute of Technology, Atlanta, July 1995.

[15] Costello, A., and Mccanne, S. Search Party: Using Randomcast for Reliable Multicast with Local Recovery. In *Proceedings of IEEE Infocom*, March 1999.

[16] Crowcroft, J. *End–to–End mailing list*, September 1994.

[17] Decitre, P., Estublier, J., Khider, A., Rousset De Pina, X., and Vatton, I. An Efficient Error Detection Mechanism for a Multicast Transport Service on the Danube Network. *Local Computer Networks*, North-Holland Publishing Company, 1982.

[18] Deering, S. Host Extensions for IP Multicasting. *RFC 1112*, August 1989.

[19] Deering, S. Multicast Routing in Datagram Internetwork. Ph.D. Dissertation, Stanford University, December 1991.

[20] Deering, S., Estrin, D., Farinacci, D., Jacobson, V., Liu, C., and Wei, L. The PIM Architecture for Wide–Area Multicast Routing. In *IEEE/ACM Transactions on Networking*, Vol 4, No. 2, April 1996.

[21] Fenner, W. Internet Group Management Protocol, Version 2. *RFC 2236*, November 1997.

[22] Floyd, S., Jacobson, V., McCanne, S., Liu, C., and Zhang, L. A Reliable Multicast Framework for Light–weight Sessions and Application Level Framing. In *IEEE/ACM Transactions on Networking*, Vol.5, No.6, pages 784–803, December 1997.

[23] Floyd, S., Jacobson, V., McCanne, S., Liu, C., and Zhang, L. A Reliable Multicast Framework for Light–weight Sessions and Application Level Framing. A later version of the ACM SIGCOMM Conference paper, November 1995.

[24] Mahdavi, J., and Floyd, S. TCP-Friendly Unicast Rate-Based Flow Control. http://www.psc.edu/networking/papers/tcp_friendly.html, January 1997.

[25] Hjálmtýsson, G., and Bhattacharjee, S. Control–on–Demand. AT&T Technical Memorandum, 1997.

[26] Golestani, J., and Sabnani, K. Fundamental Observations on Multicast Congestion Control in the Internet. In *Proceedings of IEEE Infocom*, March 1999.

[27] Handley, M., Floyd, S., and Whetten, B. Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control. http://www.east.isi.edu/RMRG/newindex.html.

[28] Herzog, S., Shenker, S., and Estrin, D. Sharing the "Cost" of Multicast Trees: An Axiomatic Analysis. In *Proceedings of ACM SIGCOMM Conference*, August 1995.

[29] Hofmann, M. Enabling Group Communication in Global Networks. In *Proceedings of Global Networking*, Calgary, Alberta, Canada, November 1996.

[30] Hofmann, M. A Generic Concept for Large Scale Multicast. In *Proceedings of International Zurich Seminar on Digital Communication (IZS'96)*, Springer Verlag, February 1996.

[31] Holbrook, H., Singhal, S., and Cheriton, D. Log-Based Receiver Reliable Multicast for Distributed Interactive Simulation. In *Proceedings of ACM SIGCOMM Conference*, pages 328–341, August 1995.

[32] Holbrook, H., and Cheriton, D. IP Multicast Channels: EXPRESS Support for Large-Scale Single-Source Applications. To appear in *ACM SIGCOMM Conference*, September 1999.

[33] Institute for Simulation and Training. Standard for Distributed Interactive Simulation - Application Protocols. Tech Report: IST-CR-94-50, University of Central Florida, Orlando, Fla, 1994

[34] Jacobson, I. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley Object Technology Series, March 1994.

[35] Kay, J., and Pasquale, J. The Importance of Non-Data Touching Processing Overheads in TCP/IP. In *Proceedings of ACM SIGCOMM Conference*, 1993.

[36] Kasera, S., Kurose, J., and Towsley, D. Scalable Reliable Multicast Using Multiple Multicast Groups. In *Proceedings of ACM SIGMETRICS Conference*, June 1997.

[37] Kasera, S., Kurose, J., and Towsley, D. A Comparison of Server–Based and Receiver–Based Local Recovery Approaches for Scalable Reliable Multicast. In *Proceedings of IEEE Infocom*, March 1998.

[38] Kasera, S., Kurose, J., and Towsley, D. A Comparison of Server–Based and Receiver–Based Local Recovery Approaches for Scalable Reliable Multicast. Tech Report: UMass CMPSCI TR 97-69, October 1998.

[39] Kasera, S., Bhattacharyya, S., Keaton, M., Kiwior, D., Kurose, J., Towsley, D., and Zabele, S. Scalable Fair Reliable Multicast Using Active Services. Submitted to *IEEE Network Magazine*, July 1999.

[40] Kermode, R. Scoped Hybrid Automatic Repeat Request with Forward Error Correction (SHARQFEC). In *Proceedings of ACM SIGCOMM Conference*, September 1998.

[41] Koifman, A., and Zabele, S. A Reliable Adaptive Multicast Protocol. In *Proceedings of IEEE Infocom*, March 1996.

[42] Kuri, J., and Kasera, S. Reliable Multicast in Multi-access Wireless LANs. In *Proceedings of IEEE Infocom*, March 1999.

[43] Lehman, L., Garland, S., and Tennenhouse, D. Active Reliable Multicast. In *Proceedings of IEEE Infocom*, March 1998.

[44] Levine, B., and Garcia-Luna-Aceves, J. A Comparison of Known Classes of Reliable Multicast Protocols. In *Proceedings of IEEE ICCC*, November 1996.

[45] Levine, B., Lavo, D., and Garcia-Luna-Aceves, J. The Case for Reliable Concurrent Multicasting Using Shared Ack Trees. In *Proceedings of ACM Multimedia*, November 1996.

[46] Levine, B., and Garcia-Luna-Aceves, J. Improving Internet Multicast with Routing Labels. In *Proceedings of IEEE ICNP*, October 1997.

[47] Lin, S., and Paul, S. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings of IEEE Infocom*, 1995.

[48] Macedonia, M., and Brutzman, D. MBone Provides Audio and Video Across the Internet. In *IEEE Computer Magazine*, April 1994.

[49] Maxemchuk, N., Padmanabhan, K., and Lo, S. A Cooperative Packet Recovery Protocol for Multicast Video. In *Proceedings of IEEE ICNP*, October 1997.

[50] McCanne, S., Jacobson, V., and Vetterli, M. Receiver-driven Layered Multicast. In *Proceedings of ACM SIGCOMM Conference*, August 1996.

[51] Nonnenmacher, J., Biersack, E., and Towsley, D. Parity–Based Loss Recovery for Reliable Multicast Transmission. In *IEEE/ACM Transactions on Networking*, Vol.6, No.4, pages 349–361, August 1998.

[52] Nonnenmacher, J., Lacher, M., Jung, M., Biersack, E., and Carle, G. How bad is Reliable Multicast without Local Recovery?. In *Proceedings of IEEE Infocom*, March 1998.

[53] Osland, P., Kasera, S., Kurose, J., and Towsley, D. Dynamic Activation and De-activation of Repair Servers in a Multicast Tree. Submitted for publication, July 1999.

[54] PANAMA (Protocols for Active Networking with Adaptive Multicast Applications) Project. Active Error Recovery Use Cases. http://www.tascnets.com/panama/AER, June 1999.

[55] Papadopoulos, C., Parulkar, G., and Varghese, G. An Error Control Scheme for Large–Scale Multicast Applications. In *Proceedings of IEEE Infocom*, March 1998.

[56] Partridge, C., and Pink, S. A Faster UDP. In *IEEE/ACM Transactions on Networking*, Vol.1, No.4, pages 429–439, August 1993.

[57] Paul, S., Sabnani, K., and Kristol, D. Multicast Transport Protocols for High Speed Networks. In *Proceedings of ICNP*, 1994.

[58] Paxson, P. End-to-End Routing Behavior in the Internet. In *IEEE/ACM Transactions on Networking*, Vol.5, No.5, pages 601–615, October 1997.

[59] Pejhan, S., Schwartz, M., and Anastassiou, D. Error Control Using Retransmission Schemes in Multicast Transport Protocols for Real–Time Media. In *IEEE/ACM Transactions on Networking*, Vol.4, No.3, pages 414–427, June 1996.

[60] Perlman, R., Lee, C., Ballardie, A., Wang, Z., Mauffer, T., Diot, C., Thoo, J., Green, M. Simple Multicast: A Design for Simple, Low-Overhead Multicast. Internet-draft, http://search.ietf.org/internet-drafts/draft-perlman-simple-multicast-02.txt, February 1999.

[61] Pingali, S., Towsley, D., and Kurose, J. A Comparison of Sender–Initiated and Receiver–Initiated Reliable Multicast Protocols. In *Proceedings of ACM SIGMET-RICS Conference*, 1994.

[62] Pusateri, T. IP Multicast Over Token-Ring Local Area Networks. In *RFC 1469*, June 1993.

[63] Ramakrishnan, S., and Jain, B. A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LANs. In *Proceedings of IEEE Infocom*, March 1987.

[64] Rizzo, L., and Vicisano, L. A Reliable Multicast Data Distribution Protocol Based on Software FEC Techniques. In *Proceedings of the Fourth IEEE HPCS'97 Workshop*, Chalkidiki, Greece, June 1997.

[65] Rizzo, L. Fast Group Management in IGMP. In *Proceedings of HIPPARC'98 Workshop*, 1998.

[66] Rubenstein, D., Kasera, S., Towsley, D., and Kurose, J. Improving Reliable Multicast Using Active Parity Encoding Services. In *Proceedings of IEEE Infocom*, March 1999.

[67] Rubenstein, D., Kurose, J., and Towsley, D. Real–Time Reliable Multicast Using Proactive Forward Error Correction. In *Proceedings of NOSSDAV*, July 1998.

[68] Speakman, T., Farinacci, D., Lin, S., and Tweedly, A. Pragmatic General Multicast. Internet Draft, August 1998.

[69] Sripanidkulchai, K., Myers, A., and Zhang, H. A Third–Party Value–Added Network Service Approach to Reliable Multicast. In *Proceedings of ACM SIGMETRICS Conference*, June 1999.

[70] Tennenhouse, D., Smith, J., Sincoskie, W., Wetherall, D., and Minden, G. A Survey of Active Network Research. In *IEEE Communications Magazine*, January 1997.

[71] Towsley, D., Kurose, J., and Pingali, S. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. In *IEEE Journal of Selected Areas in Communications*, April 1997.

[72] Towsley, D. An Analysis of a Point-to-Multipoint Channel Using a Go–Back–N Error Control Protocol. In *IEEE Transactions on Communications*, 33:282-285, March 1985.

[73] UUCast. http://www.uu.net/lang.en/products/access/uucast/.

[74] Vicisano, L., Rizzo, L., and Crowcroft, J. TCP–like Congestion Control for Layered Multicast Data Transfer. In *Proceedings of IEEE Infocom*, March 1998.

[75] Wetherall, D., Guttag, J., and Tennenhouse, D. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *Proceedings of IEEE OPENARCH*, April 1998.

[76] Whetten, B., and Conlan, J. A Rate-based Congestion Control Scheme for Reliable Multicast. White Paper, GlobalCast Communications.

[77] Xu, X., Myers, A., Zhang, H., and Yavatkar, R. Resilient Multicast Support for Continuous–Media Applications. In *Proceedings of NOSSDAV*, May 1997.

[78] Yavatkar, R., Griffioen, J., and Sudan, M. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *Proceeding of ACM Multimedia*, November 1995.

[79] Yajnik, M., Kurose, J., and Towsley, D. Packet Loss Correlation in the MBone Multicast Network. In *Proceedings of IEEE Global Internet Conference*, November 1996.

[80] Yajnik, M., Moon, S., Kurose, J., and Towsley, D. Measurement and Modeling of the Temporal Dependence in Packet Loss. In *Proceedings of IEEE Infocom*, March 1999.

[81] Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D. RSVP: A New Resource Reservation Protocol. In *IEEE Network Magazine*, September 1993.