WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

ERROR CONTROL FOR CONTINUOUS MEDIA AND LARGE SCALE MULTICAST APPLICATIONS

by

Christos Papadopoulos

Prepared under the direction of Professor Gurudatta M. Parulkar

---

A dissertation presented to the Sever Institute of

Washington University in partial fulfillment

of the requirements for the degree of

DOCTOR OF SCIENCE

August, 1999

Saint Louis, Missouri

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

ABSTRACT

ERROR CONTROL FOR CONTINUOUS MEDIA AND LARGE SCALE MULTICAST APPLICATIONS

by Christos Papadopoulos

ADVISOR: Professor Gurudatta M. Parulkar

August, 1999

Saint Louis, Missouri

One of the key elements that has contributed to the runaway success of the Internet today is its ability to support a wide variety of applications with often conflicting requirements. The Internet, however, offers only a best-effort service model which means that packets can be lost. Applications requiring reliable transmission usually employ a transport protocol to provide reliability. The most common reliable transport protocol today is TCP which provides total reliability at the expense of latency, a trade-off that works well for applications like file transfer.

File transfer, however, is just one class of applications. Other classes such as continuous media, transaction-oriented and multicast applications have different, sometimes opposing requirements than file transfer, and thus need different error control mechanisms. In this thesis, we present error control mechanisms for two important classes, namely (a) *interactive continuous media* (CM), and (b) *large-scale multicast* applications. The former include applications like teleconferencing, video on demand and visualization; the latter include Internet radio and movies, web and software updates and distributed interactive simulation.

Retransmission-based error recovery has been, in general, considered inappropriate for continuous media (CM) applications because of its latency. However, retransmission is still attractive because it requires minimal network bandwidth, (less than either peak-bandwidth allocation or FEC for bursty streams) and processing cost. In this thesis we have explored retransmission as a possibility for providing error control for interactive CM applications. We have designed, implemented, and evaluated a retransmission-based error control scheme for CM applications, which aims to provide the best possible reliability at the lowest cost, without violating the application's

timing constraints. We have enhanced selective-repeat retransmission with: (1) playout buffering to increase the time available for recovery, (2) gap-based rather than timer-based loss detection to minimize loss detection latency, (3) implicit expiration of sender retransmission buffers to eliminate acknowledgments, (4) conditional retransmission requests to avoid triggering late, unnecessary retransmissions, and (5) delivery of data integrity information to the application to aid in concealment. Our experimental results show that the mechanism achieves reductions in observed loss compared to loss without error control ranging from one to several orders of magnitude, without violating the application's delay constraints. Our work was one of the first to demonstrate the feasibility of retransmission for CM applications. Our conclusions were confirmed later by other researchers who have reached similar conclusions.

Multicast is becoming very important for emerging applications and services on the Internet. Multicast is already in use for audio and video conferencing, stock quote distribution, collaborative workgroup computing, push technologies, news distribution, shared whiteboard applications, broadcasting of live concerts, and web site updates. The size of multicast groups may range in size from a handful to millions. Control mechanisms must be very scalable to handle such groups. The greatest challenges are implosion from receiver feedback, exposure from reception of superfluous packets, and latency due to lack of local recovery. These problems are hard to solve because the existing IP Multicast provides a shared access channel which makes it hard to perform localized actions, e.g., local recovery, ACK and NACK fusion, etc.

We made the observation that forwarding and error control are two clearly separable components, and great benefits can be realized by decoupling and placing each one where it is more beneficial: the forwarding component at the routers, and the error control component at the receivers. The separation is very clean, and does not violate any layering principles. The result of this separation is a set of simple router forwarding services called *Light-weight Multicast Services (LMS)*. These services do not require examining or storing packets, but greatly simplify multicast control protocols. LMS is highly suited for reliable multicast, where it achieves near-optimal performance. Existing schemes like SRM do not require router assistance, but their performance is significantly less than optimal. The advantages of router assistance have not gone unnoticed: the router giant Cisco has begun marketing PGM, a reliable multicast scheme that postdates LMS, but is less efficient.

LMS is very light because it does not require per-packet state; it is highly scalable because it shields receivers from topology, and adapts almost instantly to any dynamic group changes. LMS has been extensively studied using simulation, and has been implemented in NetBSD Unix. In this thesis we present results from simulations that compare the performance of LMS with PGM, and SRM, which is a non-assisted scheme. We show that LMS outperforms the other schemes by three to ten times in terms of latency; in terms of exposure (reception of unwanted packets), LMS lags slightly behind PGM, which completely eliminates exposure. SRM with no local recovery suffers dramatically from exposure compared to LMS, which offers orders of magnitude lower exposure than SRM. We have also implemented LMS, demonstrating its feasibility, and evaluated its processing overhead, showing that it is on par with normal multicast forwarding.

to my parents and all who worked hard to educate me

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The birth of packet switched networks in the early 1960's has spawned the dream of an all-encompassing digital network that would span the globe and serve all our communication needs; this revolutionary network would carry voice, images, video and data, and bring together people from all corners of the world. Today that dream is becoming reality in the form of the Internet (Figure 1.1).

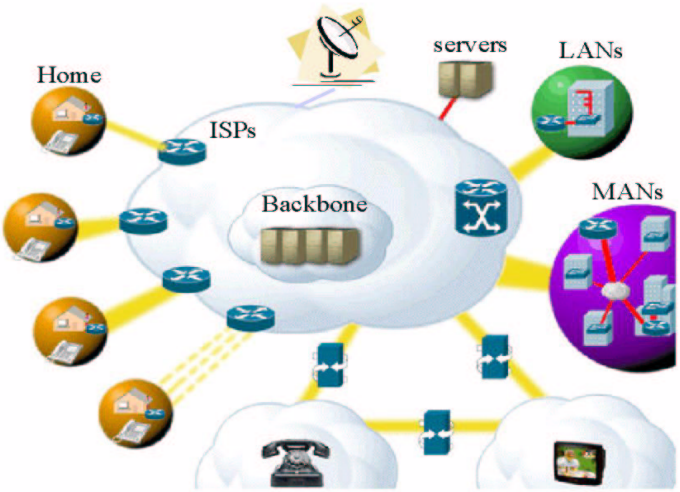

Figure 1.1: A representation of the Internet

The Internet has come a long way since its beginnings, when it was known as the ARPANET. It started as a handful of low bandwidth links used to share resources among computers, back when computers were scarce and frightfully expensive. Since its birth, the Internet has been growing exponentially; it started with the computer technology revolution, which lead to the network technology revolution, and finally to the Internet revolution. The result is that today we have a network that spans across continents, connecting several million computers and putting a staggering amount of information at our fingertips. Perhaps the biggest measure of its success is that the Internet today supports a huge number of commercial applications with revenues measured in the billions, some of which would not have been possible without the Internet.

One major component of the success of the Internet is its radical architectural shift from traditional networks like the telephone or cable television networks. Traditional networks are mostly purpose-build networks, conceived with one application in mind like voice or television. Such networks are based on a simple communication model: for example, the telephone network emulates direct wire connections by providing isolated, fixed capacity channels (called circuits). Thus, when two entities wish to communicate, they ask the network to establish a private circuit between them for their exclusive use; at the end of the conversation, the circuit is released and becomes available for use by other entities. Links are shared using *time-division multiplexing (TDM)*, which works by dividing time into fixed slots and allowing an application access to the network only during its allocated slot; thus, the bandwidth an application may use is governed by the speed of the link and duration of the slot, which is always a fraction of the total link bandwidth. Despite their limitations, existing networks like the telephone network have been highly successful, and have established a huge infrastructure around the world. They are still the main means of data transmission today, including Internet data.

Internet designers realized early on that engineering a network that emulates a wire (i.e., goes to great lengths to provide a completely reliable communication infrastructure), while well-suited for fixed bandwidth dumb terminals and telephones, is expensive and inflexible. The designers realized that by abandoning the wire-like robustness and adding intelligence at the endpoints to compensate, a network can not only be built on a much simpler and cheaper infrastructure, but can also adopt a more flexible service model that can serve a wide range of applications rather than just one. The key ingredient to such flexibility is *asynchronous packet switching*. Unlike TDM, where data

is carried in slots, packet switching allows data to be carried in variable size chunks called *packets*. In TDM the destination of the data is identified by the slot; packets, however, carry a *header* which contains the *address* (or a circuit identifier) of their destination and are therefore, *self routing*. This property allows packets to be stored in the network and forwarded immediately when the link becomes available, rather than wait for their slot; this technique, called *store and forward*, allows applications in packet-switched networks to use all of the available channel bandwidth. Packet switching has proven so successful that today about 50% of the network traffic is packet switched; that figure is expected to rise to about 80% in a few years. Therefore, it seems highly likely that network engineers will concentrate on building packet switched networks in the future.

Due to finite buffering, store-and-forward packet networks experience loss. Loss occurs due to burstiness, which is a result of computer communication. When a burst of packets arrives at a buffer which is already full, there is no choice but to drop some packets, because the network capacity has been exceeded. This results in a model called a *best-effort service model*, where the network promises to do its best to forward a packet, but delivery is not guaranteed. The best-effort model has proven highly successful in the Internet, because it makes very few assumptions from the underlying hardware. The embodiment of the best-effort model is the *Internet Protocol (IP)* which is capable of riding on many technologies and thus connecting together a wide variety of heterogeneous networks. IP clearly separates the network layer from the underlying physical layer; this decoupling of the layers allows network technologies to evolve independently, and yet still be part of the Internet. IP has been instrumental in making the Internet a “network of networks”, and thus allowing it to achieve the scale it enjoys today.

1.1. Loss is Part of the Internet

As a result of packet switching[1] and the resulting best-effort service model, network applications have to deal with loss. As mentioned earlier, in packet switched networks the finite link bandwidth, the finite buffers at the routers, the shared bandwidth model, and the bursty nature of many network applications, make it hard to absorb bursts. Thus, Internet routers often experience loss. While it is possible that advances in optical switching may eventually provide an abundance of

1. Packet switched networks can be made to have no loss, but that would increase their cost and complexity significantly.

bandwidth to terrestrial networks and loss will become rare, the heterogeneity of the Internet ensures that at least portions of it will always have limited bandwidth. Such portions include bandwidth limited segments such as ones served by modems, and wireless segments where loss may occur for reasons other than lack of bandwidth (e.g., weather conditions).

Congestion and the resulting loss is not just a function of limited bandwidth, but is actually a consequence of the Internet design. In a best-effort model, applications are not given resource bounds, but must discover available resources like bandwidth dynamically. This allows high utilization of the network. For example, today's leading Internet transport protocol, namely TCP[37], consumes bandwidth in a greedy fashion by deliberately and continuously driving the network to loss before backing off, in an effort to probe for available bandwidth.

The issue of loss is complicated even further by the recent emergence of Multicast. In multicast applications, which are envisioned to scale to thousands, or hundreds of thousands of receivers, data sent by a single sender often spans a substantial portion of the network; at such scale, the probability that a flow will experience loss is much greater than unicast.

Finally, even if we managed to eliminate loss by gross over-engineering of the entire network, loss can still occur due to mismatch in sender/receiver speed and subsequent buffer overrun. This is especially true of mobile receivers which typically have limited memory and CPU resources.

Given that Internet routers will continue to drop packets, applications must counteract loss by using *error control*. Error control is the part of a communication protocol responsible for detecting and recovering loss during data transmission. It typically contains three phases: (a) *error detection*, where the protocol detects that a packet was lost; (b) *error notification*, where the sender is notified of the loss; and (c) *retransmission*, where the sender retransmits the lost packet(s) to the receiver. Another option for error control is Forward Error Correction (FEC), where redundant data is sent along with the original data which allows the reconstruction of lost data at the receiver. Error control has been widely discussed in literature and good references include [1, 2]. We discuss it further in Chapter 3.

In summary, loss in the Internet is generally not a design flaw, but a feature that allows the Internet to be built on a very simple and economic infrastructure, provide a flexible service model that

allows several classes of applications, and achieve high network utilization. As a consequence, however, network applications must be enhanced with error control.

1.2. The need for multiple classes of Error Control

There are many classes of applications on the Internet today, and each class has different requirements for error control. Here are a few examples:

- *Web Access* and *File Transfer* are the most popular applications in the Internet today. These applications are relatively tolerant to delay; their main requirement is that data is transferred with no packet loss. There are no strict bounds on when the transfer should complete, except perhaps user frustration. Thus, even in the presence of relatively large delays (in the order of seconds), these applications' utility remains high. Examples of such applications include web surfing, FTP and email.

- *Continuous Media (CM)* applications, which include audio and video, require that data be delivered within certain time bounds, otherwise it is considered lost. Typically, these applications have a certain degree of redundancy and thus sacrificing some packets to maintain timely delivery for the remaining packets is usually an acceptable trade-off. Notice how this is exactly opposite of the previous class. Examples include teleconferencing, video-on-demand, and visualization.

- *Transaction Oriented (TO)* applications have different requirements than either file transfer or CM applications. *TO* applications require fast and reliable delivery of typically short messages. They may also have other requirements like in-order delivery or at-most-once semantics, which error control must take into account. Examples include request-response applications like database queries.

- *Multicast Applications* are a class of applications that emerged recently with the creation of the MBONE. These applications may have large numbers (hundreds or even thousands) of participants distributed worldwide. These applications' primary concern is scalability; thus they require error control schemes that are capable of coordinating recovery among a

large number of receivers. Examples include: whiteboard applications, distance learning, distributed interactive simulation and software updates.

The above list of classes of applications requiring different error control mechanisms is by no means complete; however, it serves as an indication of the many varieties of error control mechanisms that will be required in the Internet.

Defining error control mechanisms for every application is an enormous task and well beyond the scope of this work. Thus, in this thesis we address **Error Control for Interactive Continuous Media and Large Scale Multicast Applications**. We chose these classes of applications because they are very important to Internet users and pose a set of unique challenges to error control; these challenges are very different from the traditional and well-understood issues in file transfer, and have no adequate solutions yet. We describe these classes of applications next.

1.3. Error Control for Interactive, Continuous Media Applications

Continuous media applications are a class of applications employing relatively long-lived, continuous data streams, typically lasting minutes, hours, or more. Examples of CM applications include video, audio, image animation, and visualization among others. The salient characteristics of CM streams are their periodicity and strict timing requirements. For example, to produce a smooth video, *MPEG (*which is a high-quality video compression format used by most digital television programs today) requires that a frame is displayed precisely every 33 ms.

We can divide CM applications into two general classes: *interactive* and *stored media*. As the name implies, interactive applications typically involve interaction between people, and are, therefore, bidirectional; examples include teleconferencing and telephone conversations. In addition to intra-stream timing requirements, interactive applications require that data is transmitted almost instantly after it is generated, to avoid pauses in the conversation.

Like interactive applications, stored media applications demand strict timing within the stream; however, they are more flexible about when playback is initiated. The reason is that such applications are typically unidirectional and preserving interaction is not critical, except perhaps to ensure that video control operations like pause, fast-forward, and rewind remain on par with existing video

players (e.g., 0.5 - 1 sec). Examples include Internet applications like RealPlayer, Video-on-Demand, and viewing of video clips from a news server.

Error control for stored media applications is relatively easier than interactive applications because the former's unidirectional nature allows a relatively large *playout buffer* at the receiver. A playout buffer is used to store new data for a short time before playback; this is advantageous because it helps absorb jitter which helps maintain the correct playback timing at the receiver, and potentially allows additional time for error recovery. Playout buffers of several seconds are common in the Internet today; the RealPlayer application for example, typically uses a playout buffer of 20 seconds.

Most existing CM applications avoid the use of error control because it makes meeting application latency constraints difficult. For example, it is often argued that retransmission cannot be used for coast-to-coast error recovery for interactive applications because there is simply not enough time to recover. Indeed, with MPEG streams generating a new frame every 33ms and with a coast-to-coast *round-trip-time (RTT)* of 40-60ms, recovering loss with retransmission appears impossible. What makes things worse is that compressed streams like MPEG are far less tolerant to loss than uncompressed streams. Thus, due to bandwidth and loss limitations, most current Internet CM applications have resorted to custom video encodings [3, 4], which use lower bandwidth and frame rate than MPEG, and are more tolerant to loss. However, these applications typically offer much worse video quality, often resulting in jerky, postage stamp size frames. Fortunately, as bandwidth availability is increasing, MPEG-style applications will become feasible. However, this is only half of the solution; due to compression (which amplifies loss) and the presence of loss, such applications still require error control.

In this thesis, we have chosen to study error control for interactive, CM applications; we consider this to be an important class of multimedia applications because they facilitate human interaction. Our target environment is MPEG video at 30 frames per second, and target propagation latency is the US coast-to-coast round-trip-time (RTT), which is about 40-60 ms. An example application is coast-to-coast high-quality video-conferencing. This is a challenging application for the following reasons: (a) its interactive nature makes error control more difficult than stored media; (b) MPEG streams are very demanding compared to other video streams, due to their burstiness and

high frame rate; and (c) the US coast-to-coast propagation latency is large compared to the MPEG inter-frame interval.

1.3.1. Our Solution

We believe that the issues that must be addressed in designing error control for interactive CM applications include the following:

- How can we maximize time available for recovery without hurting interaction?
- Given finite recovery time, how can we maximize the probability of timely delivery of data?
- In the event that a loss is unrecoverable, how can we help the application deal with the loss?

To achieve the first objective, we observe that studies of human interaction have shown that people can tolerate a certain amount of latency in their conversations. In other words, we perceive a response from another person to be almost instant if it occurs within about 200ms [10]. We gain leverage from this observation thereby increasing the interval available for recovery with our first technique: adding a limited amount of playout buffering (about 100ms) at each side to gain some additional time for retransmission.

Our second technique, which aims to maximize the probability of recovery, is to avoid initiating retransmissions if they would never make it in time to the receiver. Such retransmissions, if allowed to go through, may unnecessarily delay transmission of original data, thus wasting precious time. To suppress these retransmissions, the receiver maintains an estimate of the RTT to the sender and uses gap-detection to detect loss and send requests. Upon loss detection, and based on the current RTT estimate and the playback buffer occupancy, the receiver determines whether a retransmission has a good chance of arriving in time before sending the request, and thus can avoid late retransmissions.

Our final technique helps the application conceal unrecoverable loss by supplying control information about the frame in addition to the frame itself. Thus, whenever a frame is delivered to the application, the error control mechanism passes a bitmap indicating which portions of the frame are

missing. The application may use this information in any way it pleases; for example, it could fill the missing part with parts from the previous frame.

In order to demonstrate the feasibility and viability of our solution, we implemented a transport protocol which employs the three techniques described above. Given our constraints, in the current implementation our protocol makes one retransmission attempt whenever loss is detected; however, we have shown how to extend the protocol to multiple retransmissions for applications with more relaxed constraints. The protocol's performance was tested both subjectively with animation streams, and quantitatively with random drop and bursty MPEG traces. We have found that even with a single retransmission, the protocol reduces the observed loss rate, sometimes by several orders of magnitude, without compromising the interactive nature of the stream. We have thus shown that the bias against using retransmission-based error control in CM streams is unfounded.

1.3.2. Contributions

Our work was done at a time when it was commonly believed that retransmission-based error control was unsuitable for interactive CM applications. Our main contribution is that with our research we have shown, against popular belief, that retransmission is not only possible, but can offer significant gains in lowering the observed loss rate in a CM stream while preserving interactivity. We have shown how to design such a protocol, and what error control mechanisms are appropriate for CM applications. Other researchers have subsequently reached similar conclusions [5, 6].

We have also contributed the implementation of a self-contained transport protocol with retransmission-based error control for interactive, continuous media applications. We have shown that such a protocol is not only feasible, but helps significantly in improving the perceived quality of continuous media and thus improves the ability of the Internet to carry interactive CM traffic. Our error control mechanism, while it works in a challenging environment like coast-to-coast MPEG video-conferencing, is tunable and can provide improved performance for other, less challenging classes of applications. Our transport protocol is integrated in the NetBSD Unix kernel and is available as an alternative to existing Internet protocols like UDP that lack error control.

1.4. Error Control for Large-Scale Multicast Applications

The second problem we have chosen to address is error control for large scale multicast applications. Multicast applications require simultaneous transmission of data to multiple receivers whose number can vary widely from a few, to hundreds of thousands. Multicast is a powerful communication model because it not only allows a single transmitter to reach potentially everyone on the network, but it does so in a very efficient and scalable manner. Multicast applications include distance learning, Internet radio and television, distributed interactive simulation, software updates and much more.

With the current IP Multicast service model [29], it is difficult to perform reliable data transfer in a scalable and efficient manner. Traditional error control mechanisms designed for point-to-point applications (e.g., TCP) either break down completely or give poor performance. The difficulties arise because the IP Multicast model requires that all messages sent to a multicast address reach all receivers subscribing to that address. While this model is simple, elegant and works well for data transfer, it is not suitable for data recovery. The reason is that losses in a multicast environment are local, i.e., they affect only part of the multicast tree. Attempting to recover data lost locally by global means leads to several problems, which we summarize below.

Implosion: Implosion is a problem that occurs when the loss of a packet triggers simultaneous messages (requests and/or retransmissions) from a large number of receivers. In large multicast groups, these messages will swamp the sender if unicast, or even the entire group if multicast.

Exposure: Exposure is a problem that occurs when recovery-related messages reach receivers which have not experienced loss. This is mainly due to the lack of “fine-grain” multicast in the existing model.

Recovery latency: The latency experienced by a member from the instant a loss is detected until a reply is received. Latency has great implications in application utility and the amount of buffering required for retransmission.

Adaptability to dynamic membership changes: This is a measure of how the efficiency (in terms of loss of service, duplicate messages and added processing and/or latency) of error recovery is affected by changes in the group topology and membership. In large dynamic multicast groups

receivers may join or leave at random intervals. Thus error control mechanisms that make assumptions about receiver population and/or location are not suitable for such groups.

Current solutions solve some, but not all of the above problems. For example, some solve implosion at the expense of latency and exposure, while others solve implosion, exposure and latency, but do not adapt well to membership changes. Nearly all existing solutions are significantly more complicated than unicast solutions, not only in terms of complexity but also in terms of state, because they require receivers to maintain topology-related information about other receivers.

1.4.1. Our Solution: Light-weight Multicast Services (LMS)

LMS [34] is motivated by the observation that the current multicast model offered by IP Multicast is not well-suited for scalable reliable multicast. The challenge, however, is to enhance the current model by adding minimal complexity, while maximizing the gain. We believe we have met this challenge. The key feature of our solution is the separation and isolation of the forwarding and error control components; the forwarding component is assigned to the routers (where it can be implemented most efficiently), while the error control component stays at the receivers. The result is that the multicast model need only be enhanced with new forwarding functionality. Freeing the receivers from the burden of topology-related operations, reduces the complexity of multicast error control to a level comparable to unicast. Like other solutions, our solution assumes some degree of collaboration among receivers, to maximize efficiency; however, it still performs well with minimal collaboration. While our solution requires modifications to the current IP multicast model, these are, however, far less complex than modifications proposed by others, including a major router vendor.

Briefly, our scheme works as follows: first, the routers create an implicit receiver hierarchy by selecting one of their outgoing interfaces as the parent for the remaining outgoing interfaces (the incoming interface is the one leading to the sender). Second, when detecting loss, all receivers immediately multicast requests; these are steered by routers to the parent interface, except for the request from the parent interface, which is forwarded upstream. This allows only one request to escape to the next level router. Third, before steering requests to the parent interface, a router marks its location by inserting its address in passing requests; we call this location the *turning point* and it identifies the root of the subtree that sent the request. Since the turning point is carried in the

request, routers need not remember any state. Finally, the first parent that receives the request and has the lost data, provides the router at the turning point with a retransmission, which the router multicasts to the appropriate subtree.

Note how our scheme addresses all problems listed earlier: implosion and exposure are reduced because the tags help routers form a virtual recovery tree, thus localizing recovery between parents and children. Maintaining the recovery tree at the routers is easy, because routers already have the necessary topology information. In addition, the tree adapts instantly to both membership and routing changes, since routers ensure that the recovery tree always tracks the multicast routing tree. Recovery latency is minimized because messages (requests, retransmissions) are sent immediately. Finally, the separation of forwarding and error control eliminates all topology state from the receivers (e.g., round-trip-time, back-off timers, parent/child assignment), along with the overhead associated with such state.

We have evaluated our solution in two ways: with simulation over topologies generated by an Internet topology generator, and with implementation in the kernel of NetBSD Unix. Our results have shown that our solution performs as well or better than existing solutions (including other solutions that change the multicast model), while being highly scalable and adaptive. We have shown that not only is receiver complexity significantly reduced, but performance has actually improved significantly: implosion is essentially eliminated, exposure is kept at very low levels (below 1% in most cases), recovery latency is better than unicast (30 - 60% of unicast latency), and adaptation to membership changes occurs instantly. We have added LMS into the kernel of NetBSD and have shown that very little new code is required. The additional code is simple and its forwarding overhead is actually less than regular multicast forwarding.

1.4.2. Contributions

It is important to understand the broader impact of our research on multicast. In our work we have taken a problem, namely multicast error control, and have shown how to decompose it into two sets of components: a set of forwarding services that the routers support, and a set of error control operations which are executed by the receivers. We have shown that this not only results in minimal changes to the multicast model, but also helps eliminate much of the complexity from the receivers.

This approach has provided us with a vantage point from where we can address other difficult multicast problems, such as congestion control and more.

The specific contributions of our work are as follows:

IP Multicast Model: We have defined enhancements to the current IP Multicast model to realize a highly scalable, efficient and light-weight error control mechanism.

IP Multicast Services: We have clearly defined a set of new forwarding services, and shown how to separate forwarding and error control functionality. We have defined the new state that must be added to forwarding entries at the routers, along with mechanisms to maintain and update the state.

Reliable multicast protocol: We have designed and implemented a scalable reliable multicast protocol that uses the above services.

Networking Code: We have written the appropriate networking code to implement the new forwarding services at the routers and demonstrated its correct operation. We have defined the new IP options required to implement these services.

Operating System Code: We have written the appropriate operating system code to allow applications access to the new router forwarding services, and integrated it with the existing IPC code in NetBSD Unix.

Comparison between RM schemes: We have done a simulation comparison between LMS, SRM and PGM. We also contributed our simulation code to the Network Simulator (*ns*) community. Our simulations for the first time evaluated all three schemes under identical conditions, and using topologies generated by an Internet Topology Generator utility, rather than arbitrary topologies. We have demonstrated that the network-assisted schemes (LMS and PGM) have much better performance than the non-assisted SRM. LMS is a much simpler scheme than PGM; however, we showed that LMS achieves comparable and often better performance than PGM, despite the difference in complexity.

1.5. Dissertation Overview

This dissertation is organized as follows. The next chapter describes our work on interactive CM applications. We present related work, followed by the description of our Retransmission-Based Continuous Media transport protocol. We present the motivation, design, implementation, testing, verification and finally the evaluation of the protocol.

Chapters 3, 4, 5 and 6, present our work on reliable multicast. In Chapter 3 we present some essential background to motivate the problem, followed by related work.

Chapter 4 presents a high-level design of Light-weight Multicast Services (LMS), and a receiver-reliable multicast protocol based on LMS. We present the specification of both LMS and the reliable multicast protocol. We discuss advantages, limitations, and extensions, and describe other potential applications of LMS besides reliable multicast.

Chapter 5 presents a simulation study of LMS, SRM and PGM. For this study we have implemented LMS and PGM in the *ns*, which contains an implementation of SRM, and used an Internet topology generator to simulate all three schemes in a variety of Internet-like topologies.

Chapter 6 presents the implementation of LMS in NetBSD Unix. We precisely describe the changes that are required in the networking code to accommodate LMS, the introduction of new IP options, and give the code that we added to implement LMS.

Finally, in Chapter 7 we conclude and discuss future work.

Chapter 2

Error Control for Interactive Continuous Media Applications

In the previous chapter we characterized continuous media (CM) applications as applications that have very strict timing requirements. We gave examples like audio and video applications, which require periodic transmission of data (e.g., 30 frames per second for MPEG video), and pointed out that the utility of such applications depends on the regular and timely delivery of data in order to ensure a smooth presentation, free of annoying artifacts like image distortion and jumping. While the strict timing requirements of CM applications make error control difficult, we also pointed out that compression schemes like MPEG make error recovery even more important because loss of a compressed data packet typically results in more information loss than loss of an uncompressed packet. We made a distinction between interactive and stored media applications, pointing out that error control in interactive CM applications is harder to implement than stored media applications, because in the former data needs to be transmitted almost immediately after generated to preserve the feeling of interaction. Long round-trip-times (RTT), like those experienced in coast-to-coast communication, complicate error control even further because they limit the time available for recovery.

At the time our work was published[85], it was commonly believed that retransmission-based error recovery was inappropriate for interactive continuous media (CM) applications, because of latency. However, we felt that retransmission was still an attractive option, because it requires minimal network bandwidth and processing cost, and argued that despite its latency, retransmission can be adapted for use even in interactive CM applications.

In this chapter we present the design and implementation of a retransmission-based error control scheme for CM applications, which aims to provide the best possible reliability at a minimal cost, without violating the application's timing constraints. To achieve this goal, we have enhanced *selective-repeat retransmission* with the following mechanisms: (1) playout buffering to increase the time available for recovery; (2) gap-based rather than timer-based loss detection to minimize loss detection latency; (3) implicit expiration of sender retransmission buffers to eliminate acknowledgments; (4) conditional retransmission requests to avoid triggering late, unnecessary retransmissions; and (5) data integrity information delivery to the application to aid in concealment. Our experiments have shown that the mechanism significantly reduces observed loss without violating the application's delay constraints. For example, it has reduced observed loss by orders of magnitude for both random and bursty loss. We have implemented this protocol in the kernel of NetBSD Unix, where it resides alongside popular Internet transport protocols like TCP and UDP.

The remainder of this chapter is organized as follows: in Section 2.1. we motivate the need for error control for CM applications. In Section 2.2. we present background and related work. In Section 2.3., we describe the features of our retransmission-based error control scheme that allow it to support delay-sensitive CM applications without violating their timing constraints. Section 2.4. presents details of our implementation. In Section 2.5. we present experimental results on our local 155 Mbps ATM testbed. Section 2.6. describes our extensions to the scheme to support multiple retransmissions, which is part of future work. Finally, Section 2.7. presents our conclusions.

2.1. Requirements of Continuous Media Applications

Continuous media (CM) streams are characterized by periodic and relatively long lived (e.g., minutes, hours or more) data exchange. Examples of CM streams include video, audio, image animation, and others. The salient characteristics of CM streams are their periodicity and strict timing requirements. Some CM streams (especially those carrying visual information, like video and animation) require very high bandwidth (>100 Mbps) if transmitted in raw form. To save on bandwidth, such streams are often compressed, which leads to highly bursty, variable bit-rate (VBR) output streams. Transmitting a VBR stream over packet switched networks is difficult without packet loss due to congestion, or without wasting substantial bandwidth with a peak rate reservation. Moreover, compressed CM streams are far less tolerant to packet loss, because compression

eliminates a significant amount of redundancy present in the uncompressed data. In addition, compressed data often contains control information (e.g, frame headers) whose loss may lead to misinterpretation or discarding of a large portion of otherwise correctly received data.

2.2. Avoiding Losses

To minimize network losses while maximizing the statistical multiplexing gain, several forms of congestion control have been proposed. These methods include adapting the source bandwidth according to congestion in the network [9], renegotiating the network reservations according to the source requirements [19], creation of multiple concurrent streams with different rate/quality characteristics [7], and hierarchically encoded streams [12]. These schemes are promising, even though some require support from the entire network. Bandwidth adaptation in these methods is typically not instantaneous, and thus some losses may still occur while the sources and the network settle to a new congestion-free state. Losses may also be experienced due to other reasons not related to congestion, like route changes, interference, etc. Thus, it is desirable for applications to complement their congestion control method with some form of error control which can recover losses quickly, to help achieve a graceful degradation of quality[1].

Traditional error control schemes mostly use retransmission and provide 100% reliability at the expense of latency. This is clearly the wrong model for CM applications, where late packets are as good as lost packets. However, retransmission has been widely dismissed even as a method to do partial recovery, in favor of other error control methods, including forward error correction (FEC) and concealment. We believe that retransmission was dismissed without investigating its full potential. We briefly discuss our reasoning next.

2.2.1. Forward Error Correction (FEC)

FEC[8] has been proposed as an alternative to retransmission for CM applications. FEC is an open-loop error control scheme that allows trading bandwidth for lower error rate while maintaining latency close to the RTT. To perform FEC, appropriate redundant data is sent in addition to the application's original data. The redundant data is used at the receiver to reconstruct the original data

1. We assume that quality degrades more gracefully if lost data is recovered and the application subsequently reduces its data rate to control congestion.

in the event that a loss occurs. Thus, loss is recovered without any end-to-end exchange between the sender and receiver, which makes FEC very attractive for applications with delay sensitive data. However, finding a good bandwidth/loss compromise is not always easy: computing the amount of redundancy and thus the bandwidth allocation for the FEC stream becomes difficult for bursty network losses, because the redundancy is proportional to the longest burst loss, which is very hard to predict. It is possible that the FEC bandwidth overhead required to recover bursty losses may be as much as 25 - 30% [24]. Thus, FEC can be a costly solution for high bandwidth bursty CM applications (while admittedly this may not be true for low bandwidth CM streams like audio).

2.2.2. Concealment

Concealment [28] is not strictly an error control scheme because it does not actually recover lost data, but rather creates an approximate reconstruction of the missing data based on available information. Concealment is strictly receiver-based and does not require any end-to-end exchange. Examples of concealment include substitution of lost data with data from an earlier frame, and various methods of interpolation and approximation. However, concealment creates artifacts, which may be detectable by the user, depending on the amount of data lost, the type of stream and the effectiveness of the concealment algorithm. High-quality concealment algorithms are expensive for high bandwidth applications and may necessitate the use of specialized hardware. Finally, since the effectiveness of concealment depends on the amount of available data, concealment becomes much harder with bursty loss.

Unless 100% reliability can be guaranteed, we believe that some form of concealment will be necessary at the receiver to hide the occasional unavoidable loss. However, there is a clear trade-off between loss and the effectiveness of concealment (for a given concealment method), so even if concealment is available, there is a strong incentive to keep losses low (perhaps by employing error control) to reduce the cost of concealment and achieve graceful degradation.

2.2.3. Retransmission

Retransmission-based error recovery has been, in general, considered inappropriate for CM applications. The main reason is that retransmission requires at least one additional round-trip time (RTT) to recover lost packets, which may be unacceptable to CM applications. One commonly cited example is that of a US coast-to-coast NTSC video stream, where retransmitting a packet requires

at least 40 - 60 ms, which is larger than the frame period (33 ms), and thus losses in a frame cannot be recovered in time.

While not applicable to cases where the RTT is large, it has been shown that retransmission is feasible in many cases where the RTT is relatively small (e.g., LANs and MANs), especially if a playout buffer is used to increase the time available for recovery [15,26]. Despite its latency drawback, we believe that retransmission-based error control is still an attractive solution because of its modest bandwidth and processing costs. For example, unlike FEC, retransmission requires network bandwidth proportional to the loss rate, not the data rate. In addition, processing costs associated with retransmission are low and all processing can be easily done in software (as we demonstrate in this chapter). In fact, retransmission is still used in several high-speed protocols [16]. In contrast, while some FEC encodings are simple and can be done in software (e.g. XOR), they involve data touching which is expensive.

The playout buffer is an important component of any retransmission scheme aimed at a latency-constrained environment, and is perhaps the most important difference with traditional retransmission schemes. However, as we argue later, the costs associated with the playout buffer are small. It is important to note that the size of the playout buffer is a trade-off between the gain in recovery time and the delay imposed on CM (especially interactive) applications. The playout delay in interactive applications is limited to a few hundred milliseconds, but this is still significantly larger than the RTT in future LANs and MANs. Moreover, playout buffering can be made much larger in non-interactive (e.g, stored media) applications, allowing perhaps enough time for several retransmission attempts. An example where large playout buffering is possible, is a regional video-on-demand (VOD) entertainment server serving individual homes, where good video quality is essential.

Even if retransmission can be given enough recovery time to be feasible, other important questions remain to be answered. For example, how will retransmission react with congestion control? Will the influx of retransmissions after a network loss period cause more congestion and more loss? Can retransmission be used in a multicast environment in a scalable fashion? We will not attempt to answer these questions in this chapter. Such questions require detailed studies which are beyond the scope of this work. However, we will attempt to give qualitative justifications why we believe that these problems can be solved.

The problem of interaction between retransmission and congestion is common to all retransmission-based schemes. One way to tackle this problem is for the application to set aside some bandwidth for retransmissions (as with FEC where the application reserves additional bandwidth for the FEC encoding). During congestion the application reduces its rate enough so that the new data plus retransmissions do not exceed the reservation (we again assume that delivering retransmissions, perhaps at the expense of reducing the rate of new data, is important, so that the stream will degrade more gracefully). Unlike FEC, however, the retransmission bandwidth is not used most of the time, which increases the statistical multiplexing gain in the network. In addition, as has been observed in [26] the network loss periods may be so short that it is possible that by the time a retransmission is injected in the network the congestion may have already cleared.

Retransmission typically does not scale very well in multicast environments without some form of implosion control. However, retransmission has been shown to scale very well in multicast environments where data can be recovered locally from other receivers in the multicast group [25,22,17]. FEC is better suited for multicast because it does not suffer from implosion. However, some problems still remain: the FEC encoding must contain enough redundancy to provide satisfactory service to receivers who may see different loss rates, which may incur a significant bandwidth overhead.

To summarize, we believe that retransmission-based error control, although not suited for all CM applications, is still an attractive, low-cost solution and remains a serious candidate for CM error control. Retransmission is well suited for most unicast stored media and many interactive applications where the RTT is low, and for multicast applications if the right implosion control framework is used. In special cases, retransmission can also be used in a multicast environments with large RTT: for example, in a live multicast connection if some of the receivers act as media recorders, they can still benefit from retransmissions even if they arrive too late to benefit other receivers.

From the remaining related work on retransmission for CM, the work closest to ours is Partially Reliable Streams [14]. We are not aware of any evaluation studies of PRS.

2.3. Making Retransmission Work

In order to minimize the retransmission bandwidth and latency we adopt selective repeat rather than go-back-N retransmission. The appropriateness of selective repeat for high-speed networks has been already demonstrated widely in literature [11,13,23]. To maximize the probability of recovery, we have added the following features:

2.3.1. Playout buffering

The time available for recovery may be increased with no perceptible (to the user) deterioration of quality, by introducing limited buffering at the receiver. This is called *playout buffering* and the buffering delay is called *playout or control delay.* In interactive applications the playout delay is limited by the perceptual tolerance of the user, which is around 200 ms [10]. In other words, a human user can tolerate a maximum channel round-trip delay of 200 ms in an interactive conversation[1]. In interactive applications the delay must be allocated to both endpoints, meaning that each may use up to 100 ms of playout delay. An example of using playout buffering is shown in Figure

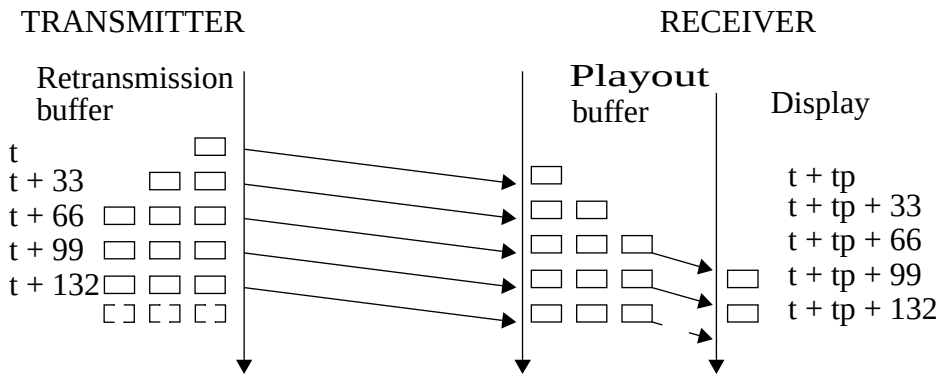

Figure 2.1: Playout buffering

2.1. We assume that frames are generated every 33 ms, so a playout buffer of up to three frames may be used, which increases the time available for retransmission is by 99 ms. This is sufficient for several retransmission attempts in LANs and MANs. Again, we note that the playout delay can be

1. This number is still debated. Some claim that this is actually a lot higher, perhaps as much as 400 ms.

much larger in case of stored media retrieval (e.g., video-on-demand) without any adverse effect on the perceived quality. It hardly matters to a viewer if the playback of a movie starts a few seconds after the movie request, if a continuous, error-free playback can be guaranteed from that point on. In practice the playout delay may be limited by the desired delay for control operations like fast-forward, pause, etc., which is approximately 0.5 - 1 second.

The size of the playout buffer is fairly small in most cases. For example, buffering 3 frames of NTSC compressed video (5 - 10 Mbps), requires a playout buffer of 62.5 - 125 kilobytes. For compressed HDTV (about 20 Mbps), the size of the playout buffer for three frames is 250 kilobytes. These numbers are well within the capabilities of modern workstations. Note that the digital frame buffer itself is much larger than this. We believe that playout buffers of this size pose no problems to television receivers and set-top boxes: some of today's high-end television sets already have enough memory to store frames for special effects (freeze-frame, slow motion, etc.).

2.3.2. Gap-based loss detection

Most transport protocols rely on timers to detect losses. In timer-based loss detection, the sender associates each packet (or group of packets) with a timer. If the timer expires before an acknowledgment is received, the packet is retransmitted. The time-out values are typically large (several times the RTT), which adds significant delay to loss detection. Therefore, unless the timer values can be determined very accurately, timer-based loss detection is not appropriate for continuous media applications.

We believe that gap-based loss detection [18] at the receiver combined with NACKs is well-suited to CM applications. In gap-based loss detection, each packet carries a sequence number. A gap is detected when a packet arrives with a sequence number higher than expected. In gap-based loss detection a gap is detected only after another packet is received. Therefore losses are detected quickly if data is sent continuously (as in high-bandwidth CM applications), provided that burst losses are not too large. Another advantage is that loss detection does not require per-packet timers.

It is important to note, however, that gap-based loss detection is applicable only if the underlying network preserves packet sequencing. For networks that do not support FIFO delivery of data, gap detection becomes more difficult, but may still be used if a decision is made about how many out-of-sequence packets can be received before a packet is declared lost. Gap-based loss detection

may take longer than time-out loss detection during prolonged network loss periods. However, sending a retransmission request before the network loss period is over may cause the request or the retransmitted data to be lost again. With gap-based loss detection, loss is detected after a new packet makes it through, so there is a good possibility that the network loss period is over (the use of RED gateways[86] invalidates this remark).

2.3.3. Implicit expiration of sender retransmission buffers

We assume that the period of a CM stream is constant and the sender knows its value. We also assume that the sender discovers the size of the receiver's playout buffer during connection setup. Using this information, the sender can estimate how long to keep data in its retransmission buffers before the data expires and can be safely discarded. Therefore, an explicit ACK from the receiver is not required to discard retransmission buffers. A simple way to implement this is for the sender to maintain a retransmission buffer with the same number of slots as the receiver's playout buffer. Thus, whenever new data is sent the oldest data in the retransmission buffer can be discarded because it has expired. Implicit data expiration eliminates delays associated with waiting for ACKs. It also eliminates retransmission of packets that would arrive late: if a retransmission request is delayed and arrives after the data has been discarded, no packets are retransmitted. Therefore the sender need not check if packets will arrive at the receiver in time before it retransmits; the sender simply retransmits if the requested data is still in the retransmission buffer.

2.3.4. Conditional retransmission requests

Since data in continuous media has a limited lifetime, there is no point requesting retransmission if the retransmitted packets will not arrive in time. In other words, retransmission should be aborted if the time left before presentation is less than the RTT. This may happen after multiple retransmission attempts have failed, or if the RTT increases (perhaps due to a change in route). Late retransmissions are undesirable because they waste network bandwidth and CPU cycles, contribute to congestion and may delay new data. To avoid late retransmissions, the receiver keeps an estimate of the round-trip time (RTT) and the presentation time for each frame, and ensures that retransmission requests are generated only if the time-to-presentation interval is greater than the current RTT estimate.

2.3.5. Data integrity information delivery to the application

Since the time available for recovery is limited, unless enough resources are reserved, there is no guarantee that data delivered to the application will be error-free. Our scheme maintains explicit information about the integrity of the received data, which is delivered to the application at presentation time. This information includes the location of missing data, if any. Such information can be valuable to the application in deciding how to deal with incomplete data (e.g., in applying concealment).

2.4. Design and Implementation

The operation of the protocol is depicted in Figure 2.2. On the sending side, the application

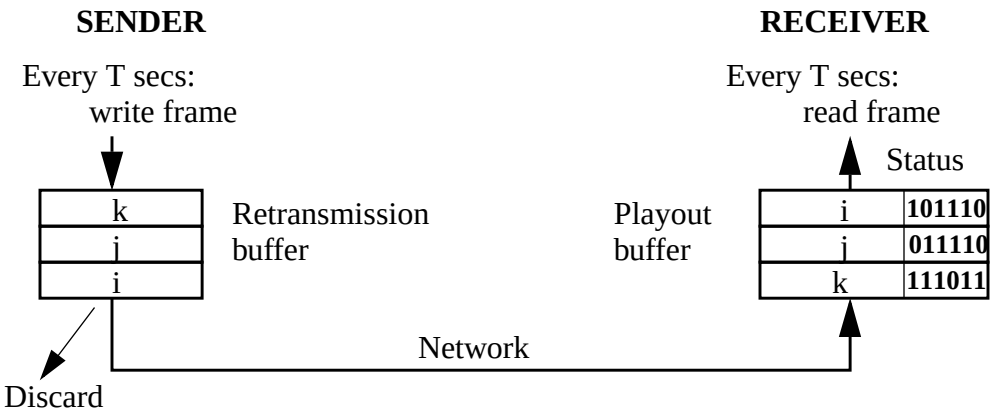

Figure 2.2: Implementation of error control scheme

writes frames to the protocol with period *T*. The protocol breaks the frame into packets and immediately begins transmitting the packets to the receiver. When packet transmission is completed, a copy of the frame is inserted into the retransmission buffer, after discarding the oldest frame in the buffer. At the receiving host, incoming packets are reassembled into frames and appended to the playout buffer. If a gap is detected during reassembly and if there is sufficient time for recovery, a retransmission request is sent immediately. The application issues read requests for frames with the same period *T* as the sender.

2.4.1. Protocol Operation

In this section we give the protocol operation at the sender and the receiver. The protocol employs a standard connection setup and tear down phases, which we omit here. All parameter exchange, like transmission and frame rate, playout buffer size, packet size, etc. takes place during the connection setup.

Sender:

- Every period T obtain a new frame from the application; break frame into packets and send it at the specified sending rate (e.g., using a token bucket); After frame transmission is complete, discard oldest frame in the retransmission buffer and insert the new frame.
- When a retransmission request is received, check if the requested data exists in the retransmission buffer; if yes, retransmit the data; if no, discard the request.
- If a RTT estimation packet is received, immediately bounce the packet back to the receiver.

Receiver:

- Every period *T* deliver the oldest frame and its error bitmap in the playout buffer to the application. This step is omitted while the playout buffer is filling up.
- Packet reception: receive packet, check for gap; if no gap, insert packet into assembled frame. If gap, update the error bitmap, and calculate time left before presentation of the frame as $T_{left} \bullet n \times T$, where *n* is the number of frames in the playout buffer before the current frame, and *T* is the rate the application consumes frames. If the time left is greater than the current RTT estimate, send a retransmission request.
- Reception of a retransmission: if the frame is still in the playout buffer, insert received retransmissions into frame and update its error bitmap; otherwise discard retransmission.
- Every RTT_ESTIMATE_IVL interval, create a RTT probe packet, insert a timestamp, increment the sequence number and transmit it to the sender.

- When a RTT_ESTIMATE_IVL packet is received, match its sequence number to an outstanding packet and estimate RTT by subtracting its timestamp from current time and update the current RTT estimate.

2.4.2. Performance

In order to test performance, we have implemented the error control scheme described in the previous section in a custom datagram transport protocol very similar to UDP. The protocol has been implemented in the NetBSD and SunOS Unix kernels[1] and sits next to UDP and TCP in the protocol stack, as shown in Figure 2.3. Currently the protocol runs on Pentium and SPARC class

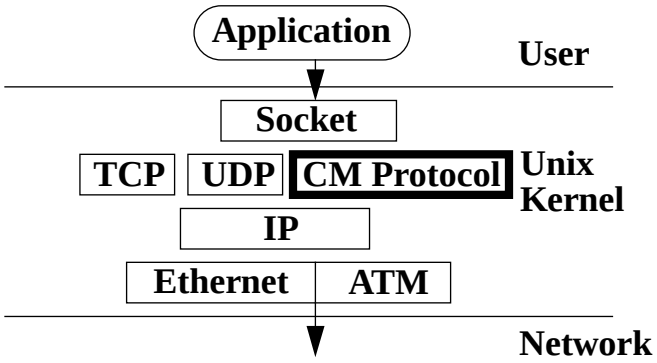

Figure 2.3: The position of our CM protocol in the protocol stack

machines. The packet delivery mechanism is IP over both Ethernet and a 155 Mbps ATM network. The protocol currently supports a single retransmission, but work is under way to extend it to support multiple retransmissions (see Section 2.6.). The sending application hands data to the protocol in units of frames. Frames can be arbitrarily large, but the protocol breaks frames into packets before transmission. Retransmission is done in units of packets. The receive end of the protocol estimates the RTT by periodically bouncing time-stamped packets off the sending host, and maintaining a smoothed average. The application sets the size (number of frames) of the retransmission and playout buffers and the preferred packet size using the `setsockopt` system call. The application receives frame status information using the `getsockopt` system call.

1. Kernel implementation allows faster loss detection and recovery, but a user-level implementation is certainly possible.

2.4.3. Processing Overhead

In the common case, (i.e., when there are no losses) the protocol processing is on par with UDP. When there is loss, the protocol processing is comparable to that of other selective repeat protocols [11,13,23]. However, in addition to the processing usually associated with selective repeat protocols, our scheme incurs the following overhead:

Conditional retransmission decision: the receiver requires information about the period of the CM stream and an estimate of the RTT to make retransmission decisions. The former is a parameter passed to the protocol by the application during connection setup. In our implementation RTT estimation is done using time-stamped packets and a smoothing function similar to [20].

Retransmission and playout buffer management: The buffer management overhead in our scheme is comparable to that in other selective repeat schemes. The main difference is that our scheme uses a small FIFO queue of buffers rather than a single buffer. However, the extra overhead of managing such a FIFO queue is small.

Playout buffer status update and delivery to application: The playout buffer status consists of a bitmap indicating the presence or absence of packets. Note that such a bitmap is part of the selective repeat implementation. Unlike other selective repeat protocols, in our scheme the bitmap is made available to the application. This involves preparation of the a *frame status*, which is a data structure containing the loss bitmap and the packet size (assuming all packets are of constant size). Both the data and the frame status can be read with a single system call[1] (`recvmsg`); the frame status is delivered to the application as *ancillary* data (see detailed discussion on `recvmsg` and ancillary data in Chapter 4).

2.5. Experiments

In this section we describe the experiments we performed to evaluate our protocol. The main objective is to examine the effectiveness of our protocol in reducing the observed loss, without violating the application timing constraints. To achieve this objective, we evaluated the protocol both quantitatively by measuring the reduction in packet loss, and qualitatively by transmitting raw video

1. Our current implementation requires 2 system calls, *read* and *getsockopt*, but it is a simple matter to change this interface to use *recvmsg*.

and experiencing visually the improvement. In addition, we conducted experiments to verify the correct operation of the protocol.

In order to measure packet loss, we needed a tool that can insert arbitrary loss and delay between the sender and the receiver. Such a tool was not available to us so we created our own. We start by describing this tool. Then, we present three experiments that used this tool to measure the loss improvement under different loss models, namely random, simple bursty loss and trace-driven, bursty MPEG loss. Finally, we briefly describe our experiments using raw video, and the users' reactions.

In all our experiments, the sender and receiver write and read frames using their own local clock, i.e., there is no feedback to keep them synchronized. To begin the session, the sender starts sending frames and the receiver's clock starts when the first frame is received. For long-lived streams (i.e., those lasting for several hours), clock drift between the sender and receiver may lead to frame overflow or underflow; in case of overflow, the receiver will periodically experience a frame loss; in case of underflow, the receiver will have to pause playback periodically, to allow the playout buffer to fill up. We have not attempted to account for clock drift between the sender and the receiver; rather we assume that the drift is small enough to be negligible.For example, we expect that given good quality clocks, clock drift will cause frame overflow or underflow perhaps not more than once an hour. Note that using a clock synchronization utility such as NTP can keep the clocks synchronized within 10 ms.

2.5.1. A Network Loss and Delay Emulation Tool

To aid in our experimentation, we have implemented a tool to emulate network delay and loss. The tool consists of a middle machine, whose kernel was modified to receive packets from the sender, delay and/or drop packets according to a user-specified loss model, and finally forward packets to the receiver. The tool is used in the configuration shown in Figure 2.4 to emulate a variety network delays and loss models. The tool allows us to apply the same loss model in both the forward and the reverse direction, which is a pessimistic assumption since it assumes both directions are equally congested. This has a negative impact on performance by dropping too many retransmission requests. The tool is quite flexible; it is driven by application level system calls that set and modify its behavior at any time during the experiment.

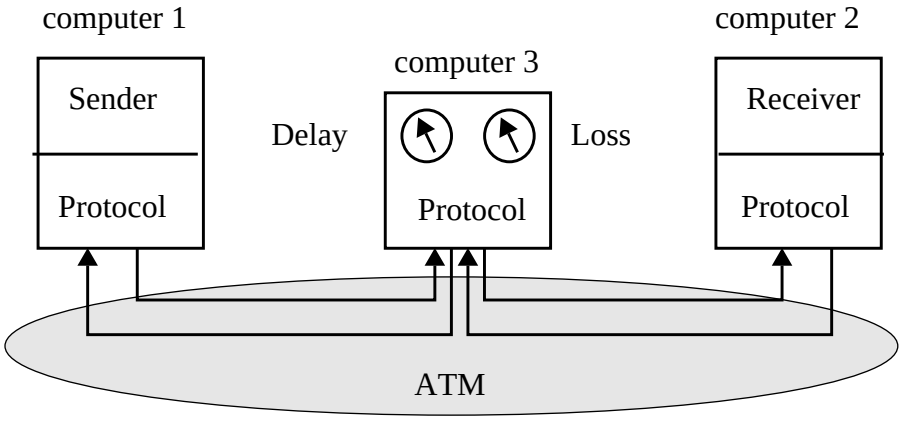

Figure 2.4: Protocol evaluation environment

In addition to experimentation, we used this tool during testing and debugging of our protocol. With the aid of this tool and standard debugging techniques (kernel debugger, print statements) we verified that our protocol worked as expected. For example, we verified that requests and retransmissions were sent and received correctly, and that no requests were generated when the RTT was set too high.

2.5.2. Experiment 1: Random Bidirectional Loss

In this experiment we measured the improvement attained by our protocol for random, bidirectional loss. We first note that the observed loss rate with one retransmission attempt and random loss can be derived analytically, and is given by: $L_{observed} = l^2_{link} \cdot (2 - l_{link})$ where l_{link} is the network loss probability[1]. Thus, for example, if the network loss probability is 10^{-4}, retransmission will reduce the observed loss to 10^{-8}, an improvement of four orders of magnitude. We use the analytic expression of loss to plot a graph that shows the reduction in link loss with one retransmission, in Figure 2.5.

In the random loss experiment we have compared experimental results to those predicted by the analytic expression. We also investigated the effect of varying the RTT on our protocol. We show here the results from one sample experiment, where the packet forwarding machine was configured

1. Assuming bidirectional random loss; the expression is easily derived, see end of the chapter.

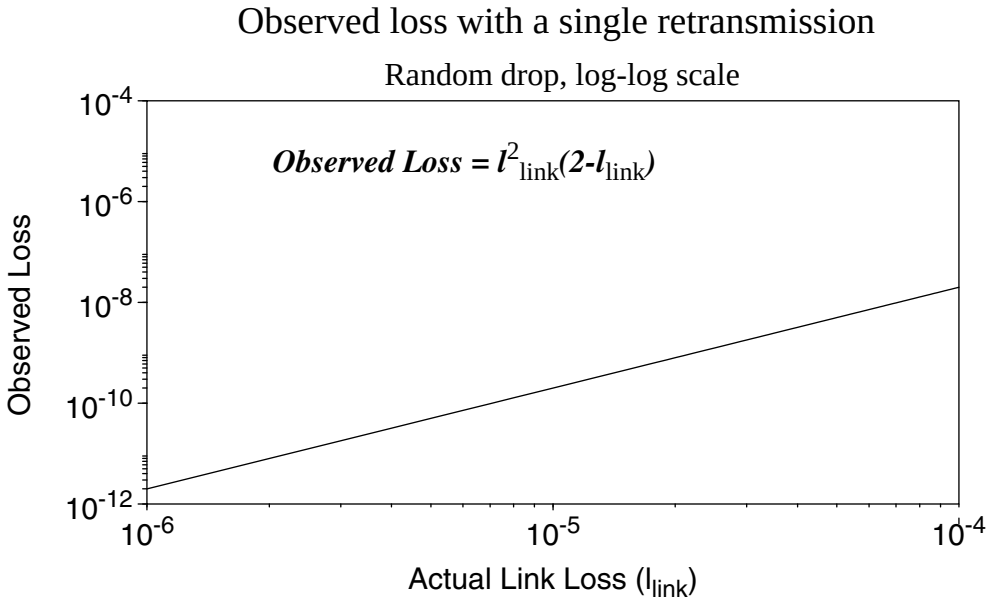

Figure 2.5: Improvement in observed loss with one retransmission

to randomly drop packets with probability 0.1. This probability was set artificially high for testing purposes. Then 10,000 packets were sent and the observed loss was recorded. The results are shown in Figure 2.6. The graph shows that our protocol maintained an error rate of about 0.019, which is what is predicted by the analytical model, for RTTs less than the playout delay. As the RTT increases the time to recover is reduced, and thus the observed loss quickly climbs towards the link loss.

2.5.3. Experiment 2: Bidirectional Bursty Loss

The previous experiment tested our protocol with simple random loss which resulted mostly in single-packet loss. In this experiment tested our protocol with bursty loss, where a larger sequence of packets may be lost. We configured the packet forwarder to alternate between two states: a loss state and a no-loss state. A simple model is used to emulate bursty network losses, where loss and no-loss periods are uniformly distributed in the intervals <mean loss period plus or minus loss deviation> and <mean no-loss period plus or minus no-loss deviation>. The same loss model was used in both forward and reverse directions. The results, shown in Table 2.1, show that retransmission significantly decreases the observed error rate; the decrease depends on the duration of the loss and

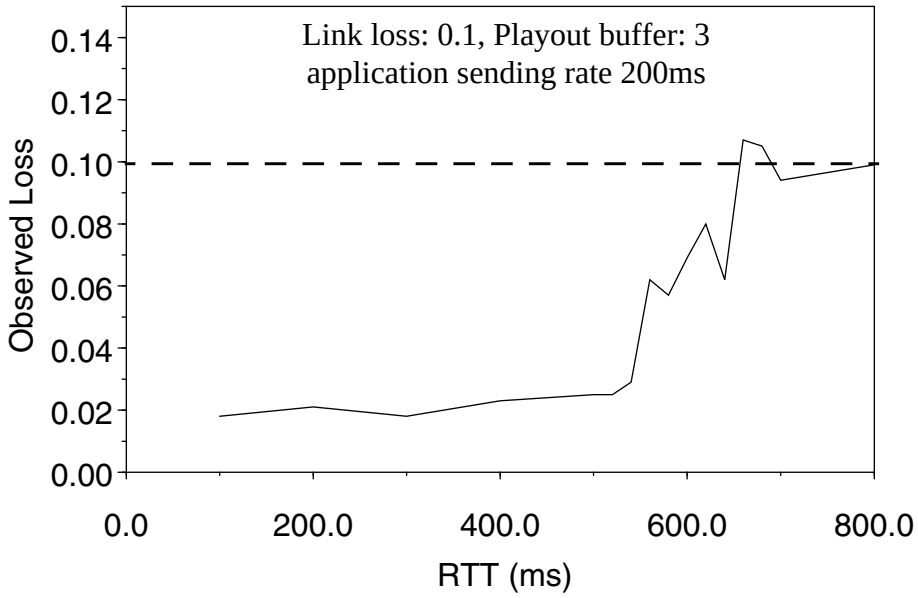

Figure 2.6: Experiment 1: Random, bidirectional loss

Table 2.1: Experiment 2: Bursty bidirectional loss

No-Loss Period (ms)	No-Loss Dev (ms)	Loss Period (ms)	Loss Dev (ms)	Packet Loss with no RTX	Packet Loss with RTX
50	5	5	2	0.082	0.017
100	20	10	5	0.1	0.0061
100	20	5	2	0.054	0.003
200	20	5	2	0.025	0

no loss periods, but is generally about an order of magnitude or more. Thus, this experiment shows that even with bursty loss the gain with retransmission can be substantial.

2.5.4. Experiment 3: Bidirectional Bursty Loss with MPEG Load

In this experiment we used the loss model described by Ramamurthy and Raychaudhuri [26], and is depicted in Figure 2.7. This model was derived by simulating several MPEG streams across an ATM video multiplexer. The mean loss period of this model was about 4 ms, and the mean no-loss period about 115 ms. With this loss model running on the packet forwarder, we run an application transmitting constant-size frames at 20 frames per second. The frame size was 48 KB and packet size was 1KB. The playout buffer was 3 frames. A total of 1000 frames were transmitted for

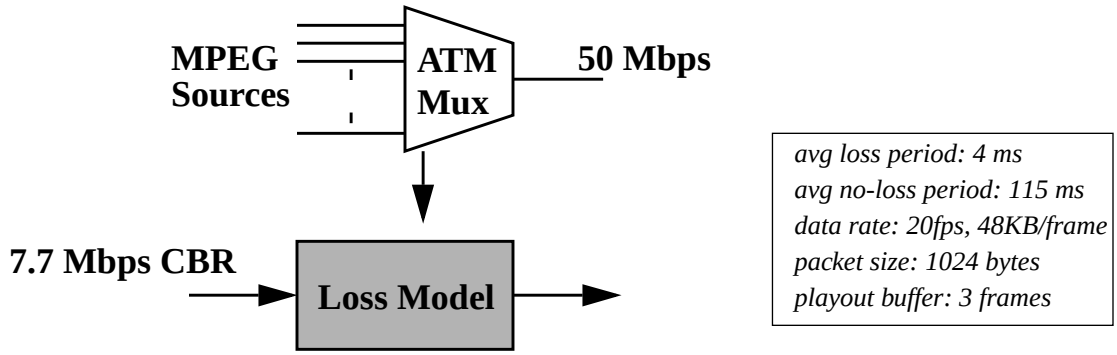

Figure 2.7: MPEG experiment setup

each measurement. The experiment has two parts: one with retransmission turned off, and the other with retransmission turned on. Table 2.2 shows two runs of the experiment without retransmission.

Table 2.2: Experiment 3: Bursty loss without retransmission

Packet Loss (%)	Number of bursts Lost	Average burst loss size (packets)
0.0253	72	17
0.030	95	15

The first column shows the percentage loss due to packets being transmitted during the loss period. The second column shows how many bursts were lost during each run and the third column shows the average size of a lost burst in terms of packets. Table 2.3 shows runs from the same experiment,

Table 2.3: Experiment 3: Bursty loss with retransmission

Average RTT (ms)	Observed Packet Loss (%)	Number of bursts Lost	Average burst loss size (packets)
8	0.0019	8	11
15	0.0013	4	15
21	0.0024	8	14
42	0.0034	11	15
61	0.0075	25	14
82	0.0047	17	13

but this time with retransmissions turned on. Each run was performed with different RTT, which is

shown on the first column. The remaining columns are the same as the previous runs. The results show that with retransmission the observed loss is reduced by about an order of magnitude compared to the loss without retransmission. The average burst loss size does not seem to change with retransmission, even though the number of lost bursts is significantly reduced. This leads us to conclude that most of the lost bursts with retransmission result from lost requests. This is an indication that multiple retransmission attempts will help in recovering more losses.

2.5.5. Experiment 4: Qualitative evaluation with raw video

In our last experiment, we used our protocol to transmit a sequence of raw video which was displayed at the receiver to allow qualitative evaluation of the resulting stream. We set up playback so that any missing data would appear as black pixels. The advantage of using raw video is that we did not have to be concerned with problems with compression, like losing synchronization. We viewed the stream with retransmission turned on and off. The result was a clear improvement with retransmission, while there were no adverse effects in frame timing. We believe that this is a strong hint that the protocol will improve the subjective quality of CM streams.

2.6. Future Work: Extending Protocol to Multiple Retransmissions

In the previous sections we have argued that gap-based loss detection works well with continuous media streams because the continuous flow of data allows fast gap detection, without resorting to timers. However, while the protocol we described works well for detecting lost data packets, it does not allow us to detect lost requests and retransmissions, and can only be used to send a single request per gap. Since our experiments have indicated that multiple retransmission attempts may be beneficial, it is desirable to extend the protocol to multiple retransmission attempts, but without losing the fast detection capabilities provided by gap detection.

To extend our protocol to multiple retransmissions, we add another field to the packet header which we use to perform *retransmission gap detection*. This field carries the sequence ID number of the last retransmission request serviced by the sender. We emphasize that the retransmission ID sequence number is totally independent of the packet sequence number. The use of the new field is depicted in Figure 2.8. The steps performed by the receiver and the sender are as follows:

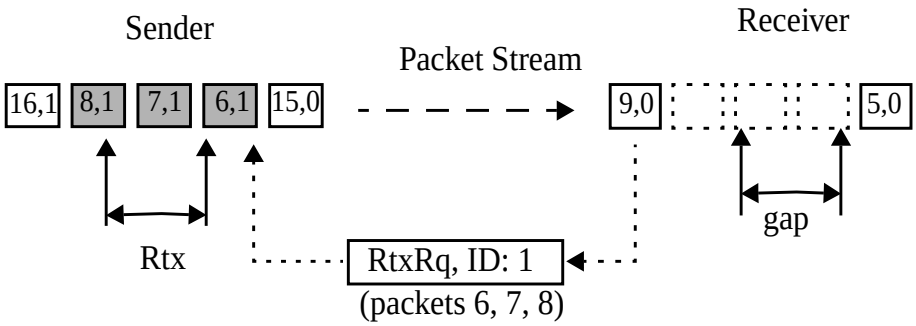

Figure 2.8: The use of RtxRq ID fields

Receiver:

- Upon detection of a gap in the CM stream, the receiver performs a retransmission test (i.e., it checks if enough time is available to recover lost data), creates a retransmission request and assigns it a new request ID from a monotonically increasing sequence number space. A copy of the request is saved in a retransmission request stack.

- If the stack contains other outstanding requests, a retransmission test is performed for each one. As a safeguard, all requests for which the retransmission test succeeds may be bundled with the new request and immediately sent to the sender. This is not necessary, but since requests are short, the overhead of bundling requests is low, and such redundancy may guard against any previously lost requests. Since requests are stored in LIFO order, if the retransmission test fails for a request all other requests below in the stack can be safely discarded.

- When there are outstanding requests, the receiver examines the retransmission ID field of all incoming data packets. If the retransmission ID field is equal or higher than the next expected retransmission ID but only some or none of the retransmitted packets were received, the receiver performs a retransmission test and sends a request with a new ID for the missing packets. The old request is removed from the stack.

- Periodically, the receiver traverses the request stack performing a retransmission test as described before. Requests that pass the test are sent again. All others are discarded.

- When a retransmission is received, the receiver removes the corresponding request from the stack.

Sender

- The sender keeps a local variable containing the maximum request ID serviced. Upon reception of a request, the sender checks the request ID to ensure that this is a new request. If not, the request is discarded.

- If the request has not been serviced, the sender sends the requested retransmissions and updates its local state, as well as the request ID field in all subsequent data packets.

The most significant cost of this scheme is the additional field in the packet header. Other costs include possible redundant requests that may be sent to the sender. The scheme, however, totally eliminates duplicate and late retransmissions while allowing fast loss detection and recovery.

2.7. Conclusions

In this chapter we have presented a retransmission-based error control protocol aimed at interactive continuous media applications. The protocol differs from traditional protocols in that it employs gap-detection, conditional retransmission, and playout buffering in order to serve the requirements of CM applications. We have implemented the protocol in the kernel of NetBSD Unix, alongside with protocols like TCP and UDP. We have evaluated the protocol with a variety of experiments, which have shown that the protocol significantly reduces observed loss, often by orders of magnitude, without violating the timing constraints of continuous media applications. We have also described how to extend the protocol to continuously sense available time and attempt multiple retransmissions only while there is a reasonable chance for recovery.

We believe that with our protocol we have presented strong evidence that retransmission can be effective in reducing loss in a wide range of CM applications. This result is important because, at the time our work was done, it was commonly believed that retransmission was unsuitable for such applications. Our most significant contribution, the error control scheme, is not tied to a particular protocol implementation, and thus can be incorporated in other protocols for CM applications (e.g., those currently using RTP [27]).

Derivation of Analytic Expression for Random Loss

We derive the analytic expression that shows what the improvement is to link loss with a single retransmission. This expression is $L_{observed} = l^2_{link} \cdot (2 - l_{link})$

Our assumptions are as follows:

- Total packets sent: *P*
- Independent bidirectional link loss with loss probability: *l*
- Single-packet drop

The derivation follows:

Packets dropped = Retransmission requests = Pl

Lost retransmission requests = (Pl)l = Pl^2

Retransmissions initiated: Pl - Pl^2 = Pl(1-l)

Retransmissions lost: Pl(1-l)l = Pl^2(1-l)

Total packets lost: Pl^2 + Pl^2(1-l) = Pl^2(2-l)

<u>Effective Link Loss</u> = l^2(2-l)

Chapter 3

Background and Related Work

Reliable multicast has been a topic of great interest for many years, and therefore a large amount of work has been done in the area. Recently, however, researchers turned their attention to the problems of large scale multicast, where groups can have receiver populations ranging from a few to hundreds of thousands. Such applications require highly scalable multicast error control.

In this chapter, we begin by presenting a brief but essential background on error control. We then proceed to describe the problems encountered when we apply traditional error control techniques to large-scale multicast. Next, we describe related work, which we divide in two parts: the first is a general overview of the literature, where we categorize proposed solutions into two categories, one for solutions that do not require network assistance and another for solutions that do. In the second part of our overview we describe in more detail two prominent error control solutions, both of which we used in the simulation comparison with our work.

3.1. Background: Positive vs. Negative Acknowledgments

Traditionally, data recovery is done using positive acknowledgments. In general, recovery works as follows: imagine a sender and a receiver, wishing to share data reliably. The simplest method to achieve reliability is by using the *stop and wait protocol*. With this protocol, the sender transmits a packet, and sets a retransmission timer. The receiver, upon receiving the packet, returns a *positive acknowledgment* notifying the sender of the successful reception of a the packet. Upon reception of the acknowledgment, the sender cancels the retransmission timer and sends the next packet. If either the data packet or the acknowledgment is lost, the sender's timer expires, the

packet is retransmitted and a new timer is set. The process repeats until the sender transmits (and receives acknowledgments) for all its data. In practice, in order to improve efficiency, protocols like TCP use a window - which allows multiple packets to be in transit before receiving an acknowledgment - appropriately called a *window-based error control* mechanism.

This is a very simplified description of window error control (more details can be found in any of the numerous networking books, including [1, 2]). What is important, however, is that with this type of error control, the sender is responsible for detecting loss and sending retransmissions. The receiver's job is relatively much simpler: upon receiving of new packet, it acknowledges the sequence number of the highest consecutive packet it has received. Because the sender does most of the important work, this approach is known as a *sender-reliable* approach[36].

We saw that the sender-reliable approach, uses positive acknowledgments. Another approach, which uses *negative acknowledgments* and puts the burden of recovery on the receiver, is appropriately called a *receiver-reliable* approach[36]. In a receiver reliable approach, the receiver is responsible for detecting loss by keeping track of the sequence number of arriving packets. A gap in the sequence number indicates a packet loss; for example, the reception of packet *n* followed by packet *n+2* signifies that packet *n+1* may have been lost[1]. Upon detection of a gap, the receiver sends back to the sender a negative acknowledgment, i.e., a message requesting the retransmission of packet *n+1*. After sending the negative acknowledgment, the receiver sets a timer, waiting for the retransmission to arrive. If the retransmission fails to arrive before the timer expires, another negative acknowledgment is sent and a new timer is set; the process repeats until the packet is received successfully. In a receiver-reliable approach, it is the sender's job that is now much simpler: it responds to each negative acknowledgment by sending a retransmission.

Although it appears that both sender-reliable and receiver-reliable approaches have equivalent functionality (namely the reliable transmission of data), they have some rather significant differences. In addition to placing the burden of recovery at different ends, the two mechanisms differ in a more fundamental manner: with the sender-reliable approach, both the sender and the receiver are notified that a packet was successfully delivered (a process that happens continuously, as data

1. While we ignore re-ordering in our discussion, note that gap-detection can still be used in the presence of re-ordering if the receiver defines how many out-of-sequence packets it can receive before it declares loss.

is transmitted). With the receiver-reliable approach, however, the absence of negative acknowledgments is ambiguous: it either means that the receiver has received all packets successfully, or that negative acknowledgments are getting lost. Therefore, with a receiver-reliable protocol, the sender can never be certain that the receiver has received all packets. Among other things, this has significant implications on buffer allocation: the sender must either provide infinite buffers, which is not practical in most cases, or make an independent decision (and risk being wrong) about when to purge and reclaim its retransmission buffers. If buffers are purged early, some data may be lost forever. One way to overcome this limitation is to have the receiver periodically send positive acknowledgments.

To summarize, receiver-reliable approaches (i.e., those using only negative ACKS) cannot provide 100% reliability; only sender-reliable approaches (i.e., those employing positive ACKS) can make such guarantees. However, receiver-reliable approaches, have merit: because they place the burden of recovery at the receivers, are better suited for reliable multicast, as we will see next.

3.2. Problems with Reliable Multicast

One could argue that the sender-reliable approach can easily be applied to small multicast groups of perhaps a dozen receivers or so. Modern hosts and networks have adequate capacity to accommodate the extra processing, state, and positive acknowledgments that would be required. However, for multicast groups that can scale to hundreds or thousands of receivers, it is clear that such solutions are hopelessly non-scalable. A sender serving thousands of receivers would be forced to maintain state, receive and process an acknowledgment from each receiver and for every packet (or window) it sends out. This not only would create a huge load at the sender, but may also create congestion in the network as the acknowledgments funnel back towards the sender. The problem became apparent very early, and thus the vast majority of reliable multicast protocols today have adopted receiver-reliability with negative acknowledgments, which reduces receiver feedback from one message per packet sent per receiver, to one message per packet lost per affected receiver. While initially this appears to significantly reduce feedback from receivers, it is still far from adequate to ensure scalability. The reasons are discussed next.

3.2.1. The Error Model

First, let us define the error model we will be using in our solution. We assume that if a packet gets lost, all receivers downstream of the loss miss the packet. The loss of one or more consecutive packets constitutes a loss event. Receivers detect these events when they see a gap. Typically, one request and one retransmission is required to recover from a loss event (assuming no further loss). If the same packet (or burst) is lost at different places independently, then we regard this as separate loss events and an independent process must be initiated to recover from such losses.

3.2.2. Implosion

We already discussed why reliable multicast protocols have adopted negative acknowledgments in an effort to achieve scalability. Here we show why this is not sufficient to make error control scalable. We begin with a problem known as *implosion*, caused by receiver feedback.

Recall that a negative acknowledgment is sent by each receiver that detects a gap in the sequence number of received packets. Also recall that with multicast, the sender sends a single packet addressed to the group, which is replicated only at the branching points in the multicast delivery tree. Figure 3.1 shows such a multicast tree with a large number of receivers. Now suppose that the “X” marks a link where a packet is dropped. Note that this link serves most all the receivers in the group. The result of this single drop is that every receiver downstream the link misses the packet. When the next packet gets through, the resulting gap causes each receiver to send a negative acknowledgment back to the sender. As one can imagine, the outcome is disastrous: a single packet loss has caused a negative acknowledgment from almost every receiver in the group, resulting in *NACK* implosion at the sender.

The problem is complicated because its magnitude depends on the location where the packet is dropped. In the best case, if the packet is dropped on a link serving a single receiver, only one NACK would be generated. However, at the worst case, if a packet is dropped near the source, all receivers will lose it. Since it is impossible to predict where a packet is dropped, an error control scheme must be capable of dealing with all possible loss scenarios, from loss at a single receiver to loss by the entire group.

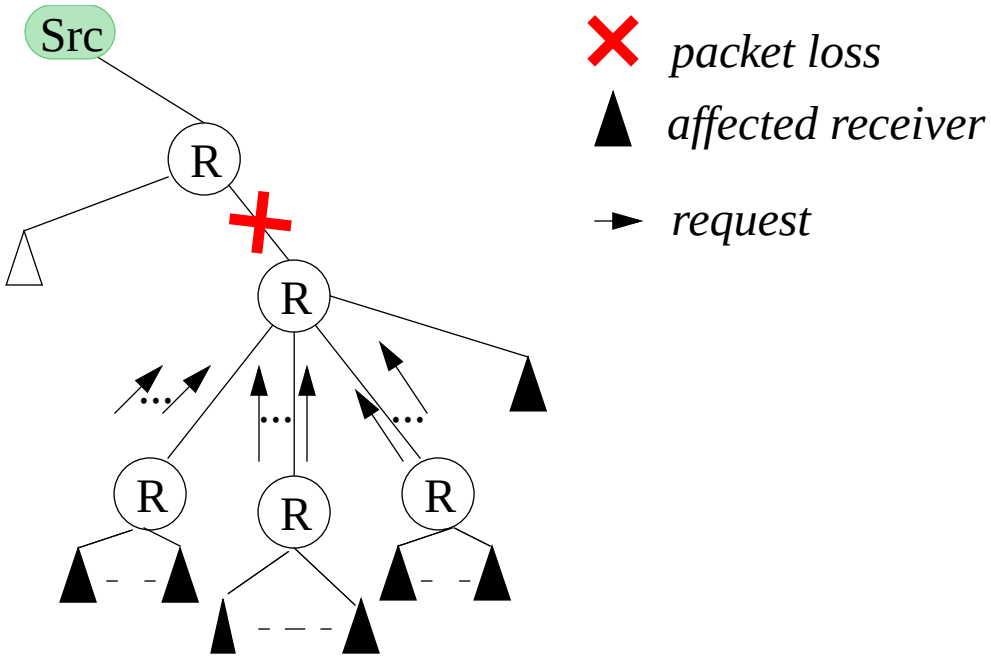

Figure 3.1: A single packet drop creates NACK Implosion.

3.2.3. Exposure

The problem of implosion is caused by receiver feedback to the sender. Implosion, however, is only one side of the problem. Even if we had the means to notify the sender of a packet loss in a scalable manner (i.e., without implosion), another problem remains, which pertains to the manner retransmissions are delivered to the receivers. This problem is *exposure*.

Let us examine the scenario in Figure 3.2. The topology is the same as in the previous figure, except that loss now affects only one receiver. The affected receiver sends a NACK back to the sender, the sender prepares a retransmission and is about to send it. The sender has two choices at this point:

- *Unicast*: send the retransmission via unicast to the affected receiver
- *Multicast*: multicast the retransmission to the entire group.

Although at first glance the first option seems to be the best, recall that in general a packet loss may affect any number of receivers, from a single receiver to the entire group. Therefore, in cases

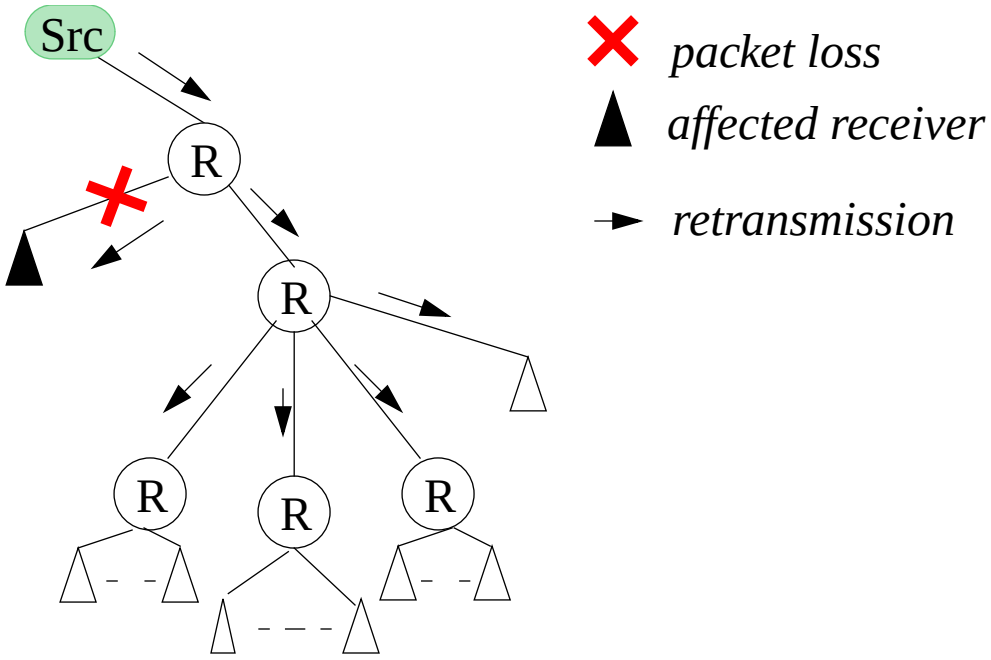

Figure 3.2: Loss at a single receiver causes exposure

where the entire group misses a packet, unicasting the retransmission to each receiver is impractical; the second approach (multicast to entire group) is definitely better. However, adopting the second approach means that when only one receiver loses a packet all receivers will be forced to receive a retransmission. This may lead to the "crying baby problem", where one receiver behind a lossy link causes excessive retransmissions to the entire group. Therefore, we conclude that neither approach is acceptable.

A hybrid approach has been suggested, that employs a threshold to select between unicast and multicast. With this approach, the sender waits for some amount of time (typically the maximum RTT in the group) while collecting NACKs. If the number of NACKs is below the threshold, then retransmissions are unicast to each receiver; if the number exceeds the threshold, the retransmission is multicast. Clearly, while this approach is an improvement, is still not scalable. As the number of receivers grows very large, selecting the threshold becomes increasingly difficult. A reasonable trade-off between implosion and exposure may be to select a threshold value as a function of the number of receivers. In practice, however, determining the number of receivers is difficult (recall that IP Multicast does not keep track of group membership). In addition, the threshold

may be limited to far less than half the receiver population because of sender's NACK processing capability may be low.

To summarize, exposure is the problem that occurs when recovery messages are delivered to receivers that do not need them. As the group gets very large, the probability that a packet is lost by one or more receivers (and hence retransmission is required) increases and a large number of recovery messages are generated. If these messages are not contained and exposure is high, scalability will suffer because unwanted packets will not only incur necessary processing at the receivers, but will also waste network bandwidth and possibly lead to congestion.

3.3. Overview of Related Work

There has been a significant amount of research on reliable multicast protocols. The early work has focused on distributed systems, providing primitives for constructing distributed applications, such as the ISIS system[67] and the V-kernel[68]. Other early work has focused on local area networks or broadcast links [69, 70, 71, 72, 73]. We will not cover the early work here; a good survey can be found in [64]. We will focus on recent work on reliable multicast that aims to provide scalability to very large groups.

As we described earlier, the vast majority of reliable multicast protocols use receiver-reliable recovery which was shown by Pingali, Towsley, and Kurose to be superior to sender-reliable recovery [36]. We begin our overview with reliable multicast schemes that require no assistance from routers. We we call these *non-assisted schemes*. Then, we proceed to list schemes that require network assistance, which we call *assisted schemes*. We will continue our overview of related work with a more detailed look at SRM and PGM, the two schemes that we use in our comparison with our scheme.

3.3.1. Non-Assisted Schemes

Many of the non-assisted schemes are hierarchical schemes, which organize receivers in a tree. Each receiver is assigned a parent and zero or more children. Request implosion is controlled by allowing requests from children to their parents only. Duplicate replies and exposure are reduced by either unicasting retransmissions from parents to children or multicasting after some threshold

of requests is exceeded. Parent discovery is a crucial step in hierarchical schemes. Some schemes are static, i.e., the parent/children allocation is fixed. Others are dynamic, and allow members to reorganize the tree as the group topology changes. Dynamic schemes are more flexible but require more complex parent discovery mechanisms.

The Reliable Multicast Transport Protocol (RMTP) [35] is an example of a static hierarchical scheme. The source multicasts data to all receivers, but only the *Designated Receivers* (DRs) return acknowledgments. Losses in RMTP are recovered from DRs. Retransmissions are either unicast or multicast depending on how many requests were received. This, however, is a crude solution because it performs well only at the extremes (if there are too many or very few losses). Otherwise, it incurs significant overhead, either in terms of network traffic or exposure. Although not implemented, RMTP was the first protocol to propose the use of *subcast*[1], a router service that allows a router to multicast a packet to all downstream links.

The Log-Based Receiver-reliable Multicast (LBRM) [32] is another example of a static hierarchical scheme, aimed at distributed interactive simulation (DIS) applications. LBRM uses a primary logging server and a static hierarchy of secondary logging servers which log all transmitted data. Data is multicast from the source to all logging servers and all receivers; however, only the primary logging server returns acknowledgments to the source. The receivers request lost data from the secondary logging servers; in turn, the secondary logging servers request any lost data from the primary logging server. Similar to RMTP, retransmissions in LBRM are either unicast or multicast, or multicast based on a threshold. Both RMTP and LBRM are based on a static hierarchy and thus require explicit set-up of DRs or logging servers before new regions can be added to the group.

The Tree-based Multicast Transport Protocol (TMTP) [40] is an example of a scheme that uses a dynamic hierarchy. In TMTP, every region has a Domain Manager (DM). When a DM joins a group, it searches for a parent using an expanding ring search. During the search, the new DM repeatedly broadcasts a “SEARCH_FOR_PARENT” request by increasing the time-to-live (TTL) value. When one or more DMs respond, the new DM selects the closest DM as its parent. Thus, the DMs form a dynamic hierarchical control tree. Each endpoint maintains the hop distance to its

1. Term coined by Adam Costello.

DM, and each DM maintains the hop distance to its farthest child. These values are used to set the TTL field on requests and replies to limit their scope. To further limit request implosion at the DMs, TMTP uses randomized backoff for requests, which, however, increases latency.

LGMP [74] is a hierarchical, subgroup-based protocol, where receivers take the responsibility of dynamically organize themselves into subgroups. Subgroups select a Group Controller to coordinate local retransmissions and process feedback messages. LGMP subgroups are self-organizing and self-adaptive according to the current network load and group membership. In LGMP subgroups may not always achieve congruency. LGMP has been implemented and some of its testing was carried out on the MBONE.

TRAM [76] is another dynamic tree-based protocol designed to support bulk data transfer. TRAM uses TTL to form the receiver tree. The tree formation and maintenance algorithms borrow from other schemes like TMTP, but TRAM has a richer tree management framework, supporting member repair and monitoring, pruning of unsuitable members, and aggregation and propagation of protocol related information.

MFTP [75] is designed for reliable distribution of files to a large number of receivers. Data is transmitted in passes. After each pass, receivers unicast NACKs back to the sender using random back-off delay to avoid implosion. The sender collects all NACKs and transmits all missing packets in the next pass. The process repeats until all receivers receive the data and no NACKs are sent. It is clear that MFTP trades latency for reliability, a trade-off which is acceptable for file transfer, but may not be acceptable for other applications.

In summary, static hierarchical schemes like RMTP and LBRM do not adapt to rapid membership changes or changes in topology. Dynamic hierarchical schemes like TMTP, LGMP, and TRAM rely on approximate methods (e.g., expanding ring search) to discover parents and send replies. The use of expanding ring search for parent selection can lead to other forms of suboptimality, due to lack of congruency between the recovery tree and the underlying topology. Other schemes like MFTP are mostly suited for bulk data transfer.

3.3.2. Schemes Using FEC

We briefly touched on Forward Error Correction [8] in Chapter 2. FEC is attractive in a multicast environments with a high degree of uncorrelated loss, because such losses can be repaired efficiently. FEC typically increases the bandwidth required to transmit data, depending on the encoding method used. Recently, techniques have been proposed that reduce this overhead and increase the effectiveness of FEC [78, 79, 77]. We chose to investigate retransmission in our work because it offers very low cost in terms of bandwidth and does not require encoding/decoding of data at the ends. Some of the techniques we have devised in our work can be used with FEC solutions, for example in sending scoped parity packets.

3.3.3. Assisted Schemes

In the last few years, there have been several proposed schemes that use network assistance for reliable multicast, which we describe below. Most of these postdate our work.

In Addressable Internet Multicast (AIM)[33], the authors propose to extend Internet routing by defining a rich set of services. These services require routers to assign per-multicast group labels to all routers participating in that group. There are three types of labels: positional, distance, and stream labels. Positional labels are used to route messages to individual members of the group. Distance labels are used to locate near-by members. Stream labels are used to subscribe to traffic generated by a subset of sources. AIM defines new routing mechanisms based on the presence of these labels. These mechanisms are:

- Positional routing: route a message to a particular destination router
- Reachcast: route to the closest router that has a member belonging to the group
- Positional reachcast: route to the closest router that has a member belonging to the group but towards some destination
- Reverse reachcast: route to the routers that can reach current router with a reachcast
- Reverse positional reachcast: route to the routers that can reach current router with a positional reachcast.

One application of the above mechanisms is the Reliable Multicast Architecture (RMA). In RMA, members requiring a retransmission ask their local router to send a request using a positional reachcast towards the source. A reachcast eventually reaches members that have the requested data, which respond by sending a retransmission via positional routing. The proposed labeling scheme has less overhead when used in shared trees. If used in source-based trees, each source tree requires its own labels. The overhead of distributing the labels after a membership change can be high if groups are highly dynamic: whenever a new branch is added to the multicast tree, all the routers below the new branch may have to change their labels.

Search Party [30] builds on our work by aiming to enhance robustness. In Search Party, requests are not routed deterministically, as in LMS, but randomly using a new mechanism called “randomcast”. This mechanism is used by routers to randomly route requests to either the parent or to one of the children. Search Party trades efficiency (in terms of increased latency and duplicates) for better robustness.

OTERS [80], uses a modified version of the mtrace[66] utility to construct a recovery tree that is congruent with the underlying multicast tree. OTERS builds the tree by incrementally identifying subroots in the multicast routing tree using back-tracing. For each subroot, OTERS selects a Designated Receiver (DR) which acts as the parent. OTERS solves the problem of maintaining congruency (in other words, ensuring that the recovery tree mirrors the underlying multicast tree), but receivers are still exposed to topology and have to keep track of changes in the structure of the underlying multicast group. In addition, the overhead of using mtrace probes may be high in highly dynamic groups.

Tracer [65] is similar to OTERS in that it also uses the mtrace utility to allow each receiver to discover its path to the source. Once the path is discovered, receivers advertise their paths to nearby receivers using expanding ring search. Once receivers discover nearby receivers, they use the data from tracing and their loss rate to select parents. Tracer can be used as a facility to create congruent trees for other tree-based protocols, such as RMTP. As with OTERS, Tracer exposes receivers to the underlying topology of the group and incurs overhead due to mtrace probes.

3.4. SRM

We now proceed to describe in detail the two schemes that as discussed in the following chapters, we have simulated and compared with our scheme. We start with Scalable Reliable Multicast (SRM) [17].

SRM employs two clever global mechanisms to limit the number of messages generated, namely duplicate suppression and back-off timers. In SRM, recovery messages (requests and replies) are multicast to the entire group; receivers listen for recovery messages from other receivers before sending their own, and suppress their recovery messages if they would duplicate one already seen. The intended goal is to allow the multicast of only one recovery message. In order to increase the effectiveness of the suppression mechanism, especially in densely packed groups, the round-trip-time between receivers is artificially enlarged (for recovery messages only) with the addition of back-off delay. To improve performance, the added delay consists of a fixed and a random component, calculated separately at each receiver. The fixed component is based on the distance of the receiver to each sender, and the random component is based on the density of the receivers in the neighborhood. However, these components have to be re-calculated when group membership, topology, or network conditions change, meaning that SRM needs time to adapt to improve performance.

SRM performs well in suppressing requests but slightly worse in suppressing replies. However, SRM has the following disadvantages:

- The backoff delay for requests is set to some multiple of the unicast delay to the sender. Thus, on average, recovery delay will be higher than unicast.
- The randomization only ensures a unique requestor or replier with a certain probability. In topologies where the distance based tiebreaker is ineffective (e.g., a star), an unfortunate trade-off must be made. Using large random numbers can make the probability of a unique requestor or replier high but increase the recovery latency; using small random numbers can make latency small but increase the probability of duplicates.

- The “multicast to everyone” approach provides excellent fault tolerance, but also exposes recovery to *all* members of the multicast group. This situation is compounded if multiple requestors and repliers respond.
- A new receiver joining the group must measure the propagation delay to every existing receiver in the group in case the new receiver is elected as a replier. Also, if adaptive timers are used, several request-reply rounds are needed before timers stabilize.

Simulation results on random topologies with fixed timer values [17] show that SRM typically requires about 2-3 times the unicast round-trip delay to recover a lost packet and produces around 2 - 10 duplicates in the process. The SRM designers have proposed an algorithm to adapt timers to improve performance. Using adaptive timers reduces the number of duplicates after the timers are tuned. However, timer adaptation can be a slow process, and there may be transients when loss shifts from one location to another.

To avoid multicasting all messages to all members, SRM proposes the use of the TTL field in the IP header to limit the scope of recovery messages. However, this approach limits the scope of messages within a radius, while losses affect a subtree. Thus, it still allows duplicates to reach other regions, as shown in Figure 3.3.

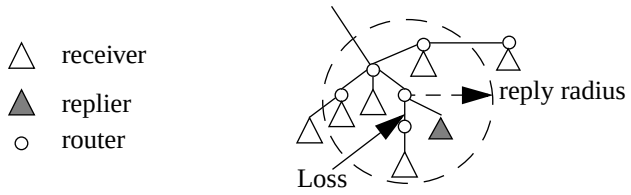

Figure 3.3: Scoping with TTL is not always effective

In [81] two approaches to enhance SRM with local recovery have been studied: one requires receivers to maintain the hop-count between themselves and all other receivers; the other suggests the use of multiple multicast groups. However, maintaining the hop-count to each receiver is a costly approach and prone to errors. Creating multicast groups for the sole purpose of recovery is a slow process, and multiplies the overhead associated with each multicast session, since creating and pruning a group is currently relatively expensive.

3.5. PGM

PGM [47] is a reliable multicast protocol marketed by the router company Cisco. PGM is a network-assisted scheme, that unlike the schemes we described earlier, peeks into the transport layer and requires per-lost-packet state at the routers.

The basic operation of PGM is depicted in Figure 3.4, and is as follows: Upon detection of

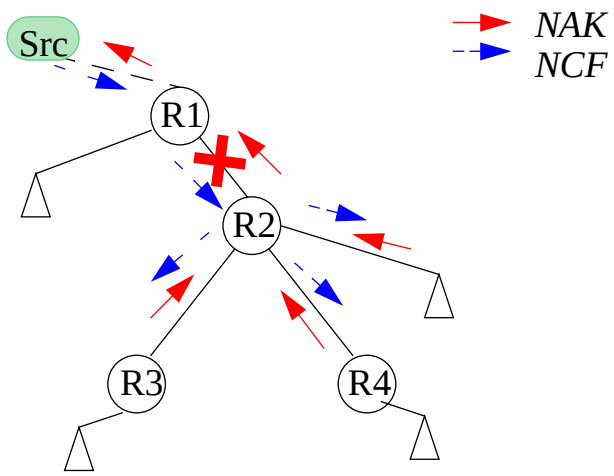

Figure 3.4: PGM operation

loss, every PGM receiver selects a short random back-off interval and then unicasts a negative acknowledgment (NAK in PGM terminology) to its upstream PGM router. Upon reception of a NAK, the upstream router performs the following steps:

- multicasts a NAK confirmation (NCF) on the link the NAK was received on to suppress other NAKs.
- creates retransmit state noting the sequence number of the requested data and the link the NAK was received on.
- repeatedly unicasts the NAK to its upstream router until a corresponding NCF is received.

Both NAKs and NCFs are examined hop-by-hop by the PGM routers. If similar NAKs arrive from other downstream links, these links are also added to the list. However, no additional NAKs

are propagated upstream. This effectively controls implosion by allowing only one NAK per lost packet to reach the source.

When the source receives a NAK, it responds with a retransmission (RDATA). As with NAKs and NCFs, RDATA is examined hop-by-hop by the PGM routers. Each router forwards RDATA on links for which is has previously established state (i.e., links where a NAK was received for that retransmission). Thus, RDATA packets follow the path laid out by previous NAKs, and consequently reach only those receivers that have sent a NAK. This completely eliminates exposure. After a router forwards RDATA, it discards the corresponding NAK state.

In PGM, all retransmissions originate from the source. Provision is made for suitable receivers to act as Designated Local Retransmitters (DLRs) but in order for a receiver to become a DLR it must lie directly on the path towards the source. This severely limits the deployment of DLRs. When a DLR is deployed, its operation is similar to the source.

PGM in its current specification, faces the following problems:

The dangling NAK state: if a NAK or RDATA is lost, previous NAK state is not discarded at the routers until the NAK state expires. Thus, when receivers that fail to receive RDATA timeout and send a NAK again, these NAKs are blocked by the routers. The reason is that PGM routers block duplicate NAKs from being propagated upstream while NAK state is present. Since NAK state is normally erased by passing RDATA which did not arrive, the state is still present until it times out. The NAK state expiration interval, however, is just a soft-state safeguard to eliminate stale state, and thus is typically large (several seconds). The solution proposed by the PGM designers is to make the NAK state permeable to one NAK after 1 second, thus limiting the amount of time a receiver's NAKs can be blocked.

Repeated retransmissions: While PGM in its current specification guarantees that a retransmission will not reach a receiver that has not requested it, it does not guarantee that a single retransmission will cover all receivers that have requested a retransmission. In some topologies PGM may have to send the same retransmission multiple times to cover all receivers. The reason is that a receiver close to the loss may send a NAK and trigger a retransmission before NAKs from distant receivers have a chance to establish NAK state in downstream routers. Since NAK state is wiped out by RDATA, a NAK arriving at a router after RDATA went through will re-establish

NAK state back to the source. This problem is depicted in Figure 3.5, which shows the same

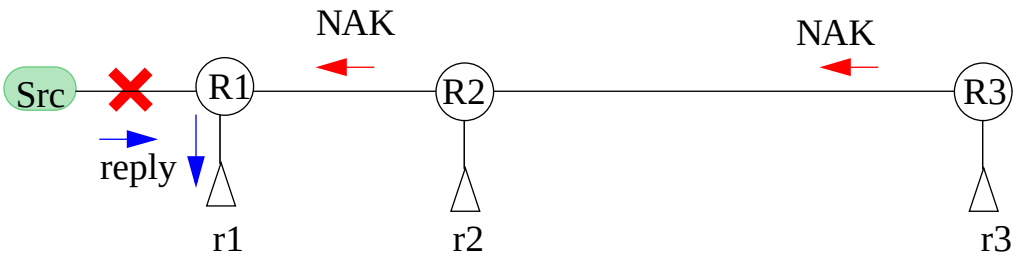

Figure 3.5: The repeated retransmissions problem in PGM

packet being retransmitted three times in response to a single loss.The RTT between receivers is such that the retransmission initiated by r1 passes through R1 to r1 before the NAK from r2 arrives at R1. Thus the NAK from r2 goes to the source and triggers another retransmission. If r3 is sufficiently distant, its NAK arrives at R2 after the second retransmission to r2, triggering a third retransmission of the same packet.

Retransmitting the same packet multiple times may create congestion near the source, which may lead to more loss and, in turn, more retransmissions, eventually leading to congestion collapse. A solution to this problem was sketched by the PGM designers and involves adding a back-off delay at the source before RDATA is sent. This allows NAKs to establish the appropriate state at the routers. This reduces multiple retransmissions at the expense of increasing recovery latency.

Chapter 4

LMS: Light-weight Multicast Services

In this chapter we present the major contribution of this thesis, namely *Light-weight Multicast Services*. LMS is a set of services provided by routers to greatly simplify the solutions to basic multicast transport problems. Even though LMS was motivated by the problem of scalable reliable multicast, we stress that LMS itself is not tied to a specific problem. It is rather a set of services that *facilitate* efficient solutions to several problems. In this chapter we concentrate on how LMS can be applied to reliable multicast, which is just one application of LMS; we list other possible applications of LMS at the end of the chapter.

LMS changes the existing IP multicast model by enhancing it with new forwarding functionality. The decision to change the IP model did not (and should not) come lightly. The existing model has arguably [82, 83] served the Internet very well to date, and understandably any attempt to change it will be met with skepticism. The model, however, which has proven successful in a unicast Internet, has come under some criticism in multicast applications. Much effort has been expended to date attempting to devise solutions to basic multicast transport problems like error and congestion control within the confines of the existing model (see some examples in Chapter 3: Related Work). These solutions, however, have met with generally limited success, and appear to have lost a significant amount of the elegance and efficiency present in their unicast counterparts.

Why have solutions to basic multicast transport problems proven so elusive? We touched upon these issues in the previous chapter, where we described problems like implosion and exposure.

These problems make multicast error recovery far more difficult than unicast, and consequently much harder to solve within the existing model. Thus, it is perhaps time to explore enhancements of the traditional IP multicast model.

In this chapter, we begin by defining an imaginary but highly efficient scheme, which we believe to be close to optimal in terms of efficiency and scalability. We use this scheme as a guiding light in our effort to realize a practical solution. After failing to apply traditional techniques to satisfactorily implement our solution, we present Light-weight Multicast Services (LMS). We describe how with LMS we can finally realize our elusive solution. We then proceed to discuss in detail the issues raised by LMS, including enhancements, limitations, and applications to other problems. Finally, we conclude the chapter by summarizing the features and strengths of LMS.

4.1. Defining an Efficient Scheme

In the previous chapter we described two important scalability problems faced by reliable multicast, namely implosion and exposure. Recall that the magnitude of these problems varies, depending on the location of a packet drop, which is hard to predict, thus making these problems difficult to solve.

Before trying to address these problems individually, we first try to determine if we can devise an integrated solution, i.e., one that eliminates both implosion and exposure. For this exercise we set aside any concerns about implementation - we will return to these later. We adopt a top-down approach for a couple of reasons: first, defining such a solution helps us understand the problems better; second, by defining a "good" solution, we arm ourselves with a standard against which other solutions can be compared with. In this section, we define a solution, analyze its operation and justify why we believe it to be near-optimal and therefore a good standard.

4.1.1. A Near-Optimal Solution

Consider the example in Figure 4.1, which shows a multicast group experiencing loss. Assume that a packet is dropped on link *L* marked with "X". Also assume that the source is relatively far away from the point of loss and there is a closer receiver that has received the data correctly. In our example, two receivers become important: the one marked as *requestor* and the one marked as *replier*. What makes these receivers important is that the requestor is the closest of the receivers

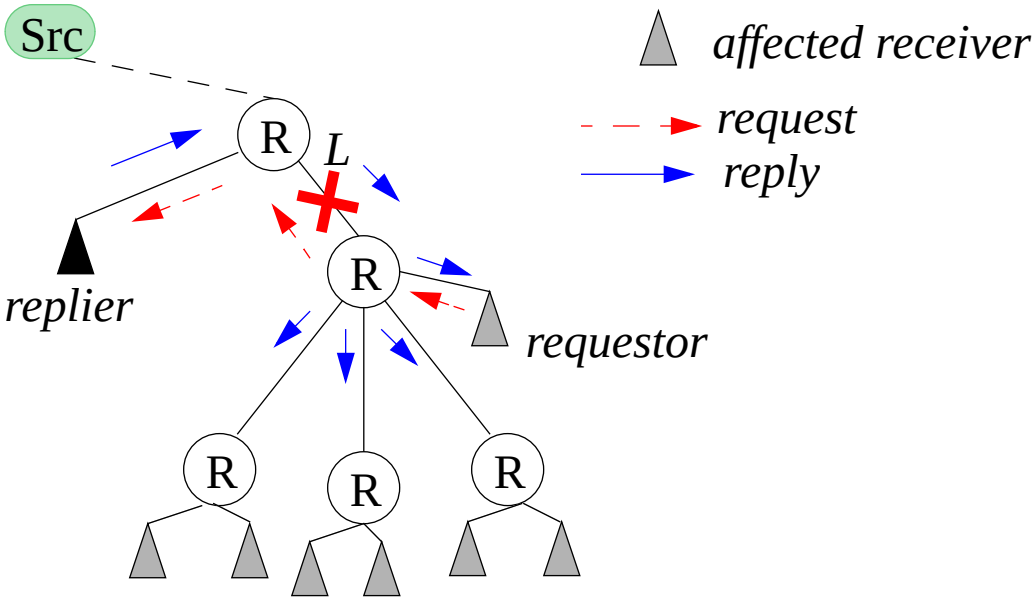

Figure 4.1: An imaginary, efficient recovery scenario

downstream of the loss that has missed the packet, and the replier is the closest receiver *upstream* of the loss that has the packet. As you might have guessed, we would like these receivers to initiate recovery, because their proximity to the loss will minimize *recovery latency*, another important performance metric which we will discuss in a moment.

So how can these two receivers collaborate to facilitate recovery? Assume that by some imaginary mechanism, the requestor knows that it is the closest receiver to the loss and sends a NACK upstream, while all other receivers refrain from sending NACKs; since only one request is generated, we have no NACK implosion. Then, using another imaginary mechanism, the NACK is forwarded to the replier only, without bothering any other receivers upstream. Finally, the replier using yet another imaginary mechanism initiates the multicast of a retransmission on link *L*; as a result, the retransmission rides on the existing multicast tree and is thus delivered to all receivers that missed the original packet.

We claim that the scheme we just presented is very efficient. Let us justify this claim:

- Only one NACK is generated by the receiver closest to the loss. A single NACK per lost packet is both necessary and sufficient to initiate recovery (assuming no NACK loss).

- Only one retransmission is generated by the receiver immediately above the loss. Again, assuming no further loss of retransmissions, a single retransmission is both necessary and sufficient to repair the loss.

- Initiating the retransmission at the point of loss ensures that only affected receivers receive the retransmission, which eliminates exposure. In addition, utilizing the underlying multicast tree results in good efficiency (in terms of latency and bandwidth) in delivering the retransmission.

- The NACK and the retransmission are initiated by the receivers closest to the point of loss. In addition, both messages are sent immediately upon error detection and reception of a request. These actions together ensure that requests and retransmissions travel the minimum distance and are sent as soon as possible, minimizing latency.

Let us study the general characteristics of our imaginary scheme a little further. We note that the scheme uses *local recovery.* By local recovery we mean that a loss may not necessarily be recovered from the original sender, as done in unicast, but may be recovered from a nearby receiver who has the lost packet. Local recovery is crucial to scalability because it does not require the participation of the source to recover each loss[1] (which may occur frequently in very large groups). Thus, by using receiver-reliable recovery *and* allowing receivers to send retransmissions, we significantly improve scalability by taking a substantial burden away from the sender. The use of local recovery implicitly assumes a *collaborative environment,* i.e., that receivers trust each other not only to retransmit lost data, but also to retransmit the *correct* data. This touches upon another thorny issue, that of *multicast security,* which is still under research. We discuss some security issues towards the end of this chapter. In addition to trust, local recovery assumes that receivers are willing and able to allocate some resources for recovery, namely processing cycles and buffer space.

In terms of performance, we noted that our scheme not only eliminates implosion and exposure, but also minimizes recovery latency. We discussed in the previous chapter the importance of limiting implosion and exposure in achieving scalability, but it may not be clear why minimizing latency is equally important. An obvious reason is that reduced latency may increase application utility: for

1. By loss here we mean the entire effect of dropping a single packet and the resulting loss observed by all receivers that missed it.

example, in multimedia applications data is only useful if delivered within certain time window, otherwise it is considered lost. However, another important reason is that latency affects the amount of buffering needed to serve retransmissions. Recall from Chapter 3, that with NACK-based error control an inexact decision must be made as to when to purge retransmission buffers. Thus, if recovery latency increases with the group size, buffers must also increase to limit the probability that a request may arrive after the buffers have been released.

Having sketched what we believe to be a near-optimal solution, we now turn to issues of implementation. Our multicast model is IP Multicast, which provides a simple, and therefore quite flexible model. IP Multicast uses a multicast-to-everyone model, i.e., a packet addressed to the group is received by all members, and provides receiver anonymity, i.e., the exact constituency of the group is not known to anyone.

When trying to implement our solution within this model it immediately becomes apparent that this will be a difficult task. Specifically, we encounter the following problems:

- Due to anonymity, it is impossible to identify the requestor/replier pair, and thus we are not able to forward requests to an appropriate replier.
- There is no mechanism to identify either the router or the link at the root of the loss subtree.
- There is no mechanism to deliver a message to a router for retransmission.
- There is no mechanism to allow routers to perform fine-grain multicast to restrict a retransmission within a particular subtree.

The topic of how to overcome these difficulties is the subject of the following sections. To summarize, in this section we have sketched the operation of a near-optimal but imaginary recovery mechanism, assuming a collaborative environment, which we claim is very scalable because it eliminates implosion and exposure, and minimizes latency. We also saw that implementing such a scheme within the current IP multicast model poses many challenges. In the next section we describe an existing approach that attempts to addresses the problems we have identified. We argue, however, that this approach fails to provide a general solution.

4.2. Implementing Reliable Multicast Using a Hierarchy

In the previous section we presented a scheme that although efficient, is hard to implement due to the existing multicast model's anonymity and lack of fine grain multicast. In this section we present two variations of an approach that attempts to solve these problems by organizing receivers into a hierarchy.

4.2.1. A Logical Receiver Hierarchy

With a hierarchical approach, receivers are arranged in a logical structure, where each receiver is assigned a *parent*. The receivers assigned to a parent are its *children*, and the parent/children assignment forms a *hierarchy*. Parents and children know of each other by maintaining state. The hierarchy is typically created when the multicast group is formed and may remain *static* for the lifetime of the group, or may be *dynamic*, i.e., allow receivers to re-structure themselves as the group membership changes. Static hierarchies are simple to maintain, while dynamic hierarchies pose several challenging problems; we will examine these shortly. With a hierarchy, multicast error recovery is typically done as follows:

- when a receiver detects a loss, the receiver unicasts a request to its parent;
- if the parent has the requested data, it unicasts a reply back to the child; if not, it must also have sent a request to its parent, and thus it simply remembers that the child needs a retransmission;
- after the retransmission arrives, each parent forwards it to the children that requested it.

Note the differences of the hierarchical scheme compared to the efficient recovery scheme presented earlier. In the hierarchal scheme, every affected receiver generates a retransmission request; however, there is no implosion because the hierarchy limits the scope of requests between children and parents. Even though only one request actually triggers the retransmission, the remaining requests do not go to waste: they help eliminate exposure by establishing a return path for retransmissions. However, there are inefficiencies in this model compared to the earlier one. These are as follows:

- Retransmissions are not sent utilizing the existing multicast tree, but propagated hop-by-hop, which requires more work at the receivers and increases recovery latency.

- To achieve efficiency, we must be careful to maintain a hierarchy whose logical structure is *congruent* with the underlying multicast tree. By congruent we mean that the logical and physical structure should coincide. If congruency is not achieved a receiver may be assigned a parent that is downstream in the multicast tree. Such a parent will not be helpful in recovery because it will have lost the same packet as its child (assuming correlated loss). Maintaining congruency is hard to achieve because it requires knowledge of topology.

- If the group membership changes dynamically, the hierarchy needs be updated frequently: New receivers have to find suitable parents, and existing receivers whose parents leave the group must be re-assigned to new parents. Finding parents is hard due to the multicast model's anonymity.

- A hierarchy needs to be established for each sender. Thus, in multi-sender groups using source-based or bidirectional trees, the hierarchy overhead is proportional to the number of senders.

In summary, while a receiver-organized logical hierarchy appears to solve the problems of implosion and exposure quite elegantly, such an approach is more appropriate to relatively static groups. It is not well-suited for large, dynamic groups because it is hard to maintain a congruent logical and physical structure. For such groups we need a solution that can efficiently maintain congruency.

4.2.2. A Router Hierarchy

In the previous section we examined a fairly good approximation of our efficient solution using a logical receiver hierarchy. We argued that while such a solution is reasonably efficient, is not well suited for large, dynamic groups because, among other problems, it is hard to maintain congruency between the logical and physical structure. In this section, we address the problem of congruency by using help from the network, i.e., from routers.

The problem of maintaining congruency can be solved if we enlist help from the network. Assuming that routers are allowed to buffer packets, detect losses, send requests and retransmit lost packets, we could make each router a receiver in the multicast group. Then, by using only routers to form our hierarchy we can solve the congruency problem, as depicted in Figure 4.2. Here is how

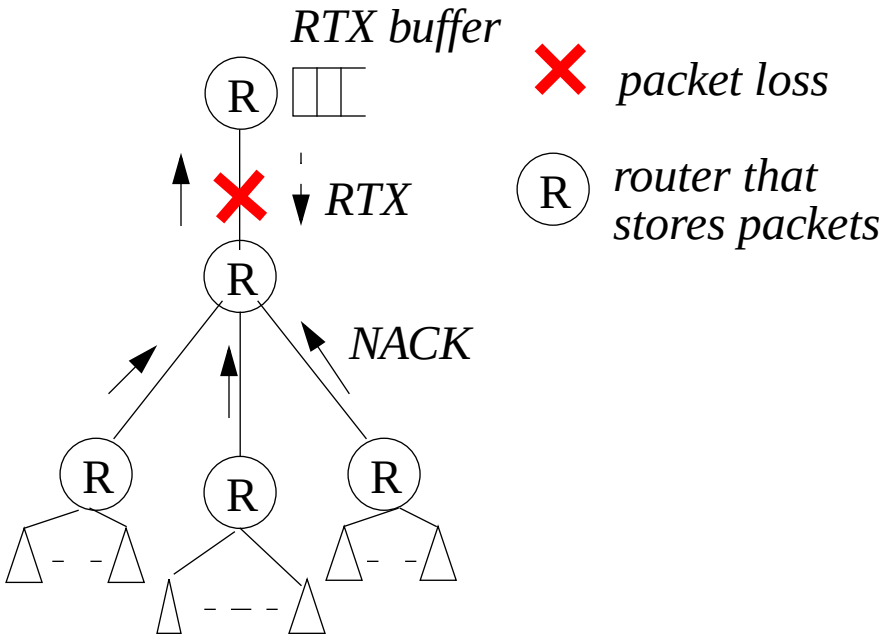

Figure 4.2: Reliable multicast using a router hierarchy

it would work:

- as soon as a packet is dropped on a link, the link's downstream router detects the loss and sends a request to the upstream router asking for the packet to be re-multicast on that link;
- routers downstream the loss send only one NACK upstream and suppress similar NACKs arriving on their downstream links; this eliminates implosion;
- the first router upstream of the loss re-multicasts the packet on the link where it received the NACK (which is the link on which the packet was dropped); this ensures that the retransmission follows the same path the original transmission would have followed, had it not been dropped, thus eliminating exposure.

This solution is quite simple and elegant; it not only retains the advantages of the previous solution (i.e., suffers from neither implosion nor exposure), but it actually improves upon them: it reduces recovery latency, because requests travel a shorter distance, and it eliminates the hop-by-hop delivery of retransmissions, replacing it with a single, efficient multicast. However, despite its simplicity and elegance, the scheme has several problems that make it impractical. These problems include:

- It is extremely heavy-weight. Such a solution requires that routers buffer all multicast packets that require reliability. This may potentially consume an enormous amount of resources at the router - resources that might be better spent on reducing loss, not recovering from it.

- Processing at routers would increase significantly. Routers are already struggling to keep up with increasing link speeds and high demand. Such a scheme would slow them down considerably because they would have to manage buffers and state, and make transport-level protocol decisions (e.g., when to purge buffers and how to handle late requests). Such protocol processing would most likely be done in software, bypassing the fast path.

- Such a solution not only violates layering, but goes against one of the fundamental Internet design principles that states that intelligence is best placed at the endpoints rather than inside the network.

In summary, our attempt to create an acceptable implementation of our efficient solution using traditional ways to create a hierarchy has failed. We considered a receiver hierarchy first. While this proved to be quite efficient, it has difficulties maintaining congruency between the logical and physical structure, and is thus unsuitable for applications involving dynamic groups. Then, we considered a router hierarchy, which not only solved the congruency problem, but actually improved the performance of our efficient recovery scheme. Unfortunately, we observed that in practice a router hierarchy is very hard to implement, because it adds too much weight and complexity to the routers and violates proven Internet design principles.

4.3. The LMS Concept

In this section we come to the main contribution of this thesis. Armed with the lessons learned in the previous sections, we take a fresh look at the problem, in an attempt to reconcile efficiency and practicality. We begin by taking a closer look at the router hierarchy solution, which despite its problems, is appealing due to its elegance, efficiency and simplicity.

We have already seen that the scheme's main strength is that because it lives at the routers, it is able to take advantage of its location to automatically build a congruent hierarchy; in other words, the scheme can instantly select points to (a) perform request suppression and (b) initiate fine-grain multicast of retransmissions. However, hidden in the above statement is the following key observation: it is not the router's processing power that the scheme exploits, but the router's *location*. In other words, the key advantage of enlisting routers to assist in error recovery, is not to utilize their processing cycles, but their knowledge of topology. This is important because our major objections with the router hierarchy scheme emanated from the fact that we had routers do processing; but as we just observed, processing is not what is important in this scheme. Would it then be possible to move the processing away from the routers, but not lose the location advantage? Or to summarize the question:

> *In the heavy-weight router hierarchy model, efficiency is not achieved due to harnessing the routers' processing power, but due to their location. Would it then be possible to move the processing away from the routers while retaining the location advantage and thus preserving the elegance and efficiency of the model?*

The above question lies at the heart of LMS; it also defines what LMS is: a set of services whose purpose is to allow the migration of processing away from the routers while retaining the location information. LMS achieves this by moving the processing from each[1] router to another entity, called a *surrogate*. The surrogate thus becomes responsible for performing recovery tasks on behalf of the router. The conceptual transformation from the router hierarchy to the surrogate model is depicted in Figure 4.3.

---

1. Here we are referring to routers that would have participated in recovery in the router hierarchy model.

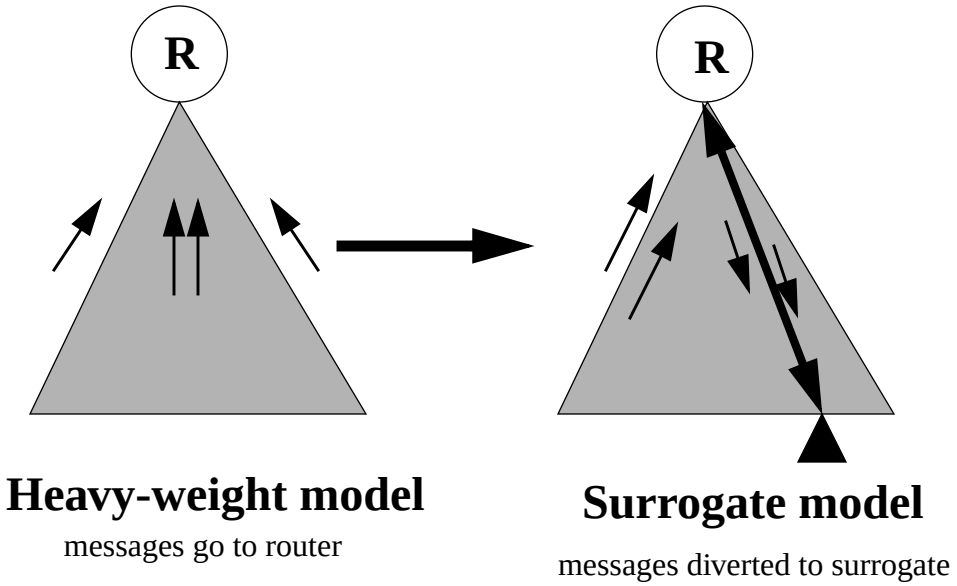

Figure 4.3: The LMS Concept

The next question is “who should the surrogate be?” While it is certainly possible to off-load router processing to a recovery server (e.g., a separate box attached to the router), this only sidesteps the problem. With such a server we introduce a new resource which needs to be managed and shared, just like the router before. Such a resource also constitutes a central point of failure. But more importantly, we have not removed the weight from the network - we simply introduced another network entity to carry it. Thus, we propose that the best candidates for surrogates are the *receivers*. While expecting a receiver to do some work towards recovery is not unreasonable (such a collaborative environment is at the heart of the Internet philosophy [21]), selecting receivers as surrogates has significant advantages: (a) it pushes the load of recovery at the ends rather than concentrating it at the routers, (b) has great flexibility because any future changes can be done at the ends without affecting the network, and (c) all recovery operations remain within the transport layer, which avoids the layer violation of the earlier model.

4.4. LMS Core Ideas

Having decided that we will use receivers as surrogates, we need to address the following questions:

- how does a router select a surrogate?
- how does a router redirect messages to its surrogate?
- how does the surrogate send messages on behalf of the router?

Before addressing these questions, let first us explain our terminology. Since the main problem here is multicast error recovery, we will shift from the term “surrogate” to the terms “requestor” and “replier”, as used earlier in the context of our efficient recovery solution. In most cases, we will be using the term “replier” to denote the router surrogate; however, both the requestor and the replier are surrogates. What distinguishes them in the context of error recovery is whether they are sending a request or a retransmission. In summary:

Router surrogate = Replier = selected receiver

Requestor = any receiver (plain or a surrogate) that sends a request

4.4.1. Selecting a Replier (surrogate)

Since we chose receivers as repliers, a router needs to select a separate replier for each multicast group. The process of selecting a replier is very simple:

- if the router has two or more downstream links, select one as the replier link;
- if the router has only one downstream link, select the downstream link;
- if the source is directly attached to the router, select the source.

Figure 4.4, shows a possible router-replier allocation. The links that lead to a replier are in bold. Note that a router only needs to know the replier *link*, not the actual receiver. For example, router R2 selects the right-most link as the replier link; it does not need to know that E5 is the receiver acting as a replier, which was selected by router R4. Therefore, a router only needs to identify the next hop of a path leading to a replier, which has some important advantages:

- If the replier changes, the change remains mostly local. For example, if R4 decides to switch to E4 as its replier (because E5 either left the group or crashed) R2 does not need to change its replier state.

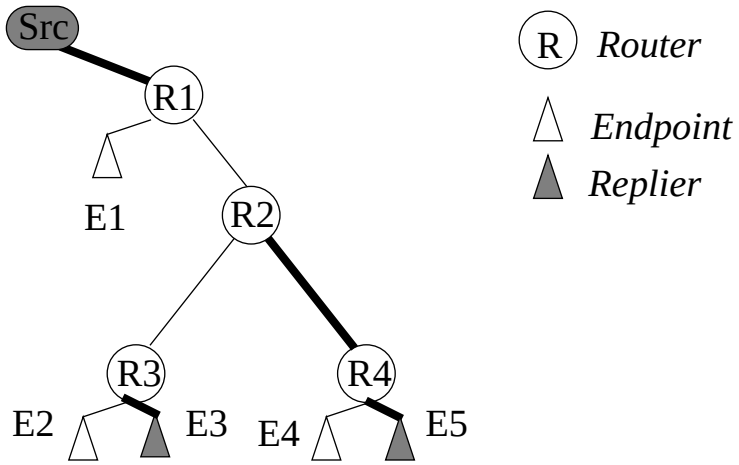

Figure 4.4: A possible replier allocation in LMS

- Receivers do not have to be notified that they have been selected as repliers. A receiver knows it has been selected if it receives a request. There is, however, no guarantee that the receiver will remain a replier for the next request.

- The replier state at the router is small. It is simply an identifier for the replier link.

In our discussion thus far, we did not specify which link the router selects as a replier link when there are two or more candidates. While the simplest solution would be to choose one at random, there are many reasons why we would prefer the receivers to influence the replier selection. For example, some receivers may be better suited to act as surrogates because they are located on fast machines, or machines with larger memory. Therefore, we introduce a replier *metric* to the receiver join procedure by which receivers control the replier selection at the routers, as described next.

Replier Selection Mechanism

To establish replier state, joining receivers express their desire to become repliers with the join request. We expect that all receivers will agree to become repliers (data recovery can be achieved even if this is not the case, but at reduced efficiency). Along with the join request, receivers communicate a metric of their appropriateness to be repliers. Routers pick repliers according to this metric. Receivers repeat their willingness to be repliers and their metric on every membership refresh.

Thus, through the combination of the join/refresh messages and the metric receivers can control the replier allocation at routers.

Replier Selection Metric

The metric used by repliers to communicate their appropriateness to the routers is a scalar value, whose meaning is determined by the receivers. The routers do not need to interpret the metric, but simply compare it with the metric provided by other receivers. Thus, receivers wishing to minimize latency for example, may choose a metric that represents the round-trip-time from repliers to the routers, so that routers select the nearest replier. Other receivers may use processing and memory capacity as a metric, so that hosts with more resources are preferred over those with less resources.

In summary, receivers are free to agree upon which metric is important, translate it into a scalar value and communicate it to the routers. It is a simple matter for the routers then to chose repliers according to the receivers' wishes.

4.4.2. Steering Messages to Repliers

After the replier selection is in place, the routers need mechanisms to steer messages from requestors to repliers. We describe these mechanisms next.

A request sent by a requestor needs to climb up the multicast tree until it finds a router which has a replier. It it thus important that the request follows the reverse multicast path. To achieve this the request is multicast, but contains a special IP option that forces routers to remove it from the fast path and examine it. That option is the IP Router Alert option [44]. The request contains the information shown in Figure 4.5.

The first two fields are part of the standard IP multicast header. The values in the IP option are new and are used as follows: The address of the original source is needed so that the router will select the correct replier (see discussion on source spoofing in section 4.9.) The request identifier simply identifies this message as a request; the request sequence number counts the number of similar requests sent; it is used to suppress duplicate retransmissions as discussed in section 4.5. The request bitmap specifies which packets are requested. The last field, the turning point information, is initially empty. We explain its purpose next.

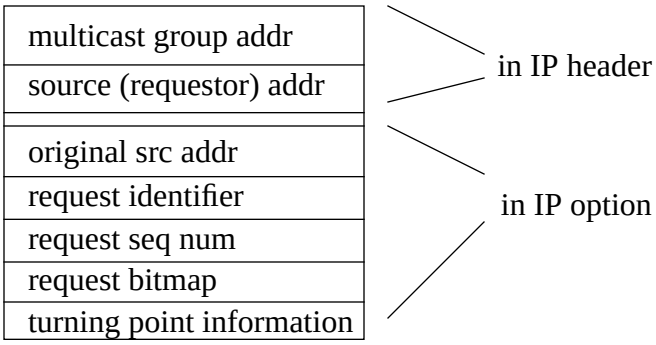

Figure 4.5: An LMS request

Request Handling at the Routers

As we saw before, even though a request is multicast to the group, it is immediately picked up by the first router because of the IP Router Alert option. A request may arrive at the router from three possible locations, as depicted in Figure 4.6:

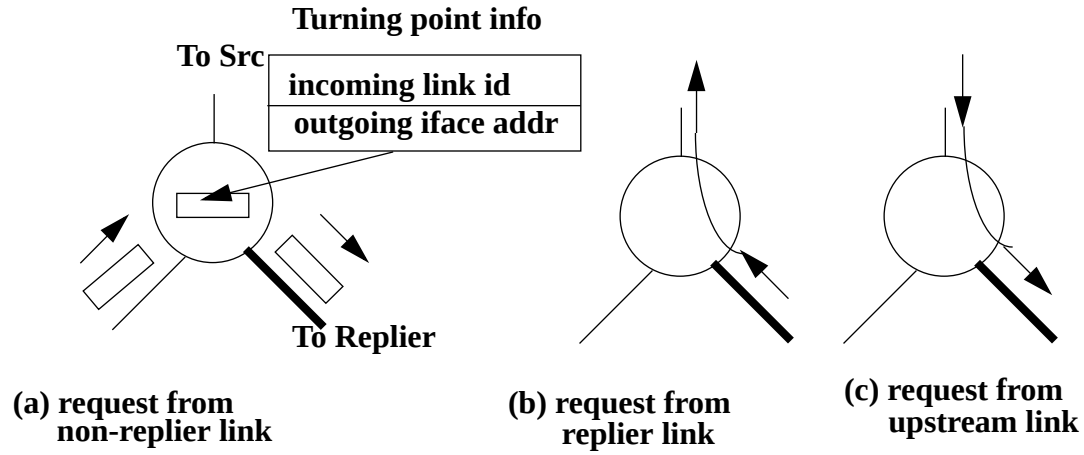

Figure 4.6: Request handling at a router

- **Request arrives on a non-replier link:** note that this case implies that the router has at least two downstream links. We call this point the *Turning Point* of the request. The reason is that the router will turn this request around (recall that the request was traveling upstream until this point) and send it out the replier link. The turning point is the only case

where some router processing is involved: before forwarding the request to the replier link, *and if the turning point field is empty*[1], the router fills in the turning point information. This information consists of an identifier for the link the request arrived on (the non-replier link) and the address of the interface the request if forwarded out on. Note that the turning point information globally identifies the point where the request was received. The reasons for carrying this identification in the request will become apparent when we describe how replies are sent.

- **Request arrives from the replier link:** when a request arrives from the replier link, then the router forwards the request to the upstream link, without making any changes to the packet.

- **Request arrives from the upstream link:** similar to the previous case, when a request arrives from the upstream link, the router forwards it to replier link unchanged.

Let us now examine the net result of request handling at the routers and see how it avoids implosion. Note that each router allows only one request to be sent upstream - the replier request. All other requests are funneled into the replier link, and follow replier links to eventually reach the replier. Therefore, if all routers behave in a similar fashion the maximum number of requests a router sends to the replier is equal to the number of downstream links at that router minus one. Assuming that routers do not have a large fan out, repliers will not experience implosion. Routers with a large fan out can be dealt with as described in section 4.13.

To summarize, we have added a simple steering mechanism at the routers which solved the implosion problem by allowing only one request to escape each router and propagate upstream; all other requests are forwarded to repliers.

The Request Turning Point

In the previous section we briefly touched upon the issue of the turning point information carried in a request. This is an important concept in LMS. The turning point is essentially the point where a request, during its search for a replier, arrives at a router through a non-replier link. Thus,

1. See later discussion on proxy directed multicast in section 4.14.1. as an example of where a router may find the turning point field non-empty.

a request has limited scope: it moves upstream until it finds the next closest path to a replier. A request arriving on a particular link signifies that all the receivers downstream that link require the data. Thus the location of the turning point specifies the root of the loss subtree that sent the request.

Note that because of the request handling, only one replier (who has the data) will receive a request; in addition, this replier will receive only one request for each loss event. This is a nice property because it eliminates ambiguity and the need to coordinate multiple potential repliers.

4.4.3. Directed Multicast (DMCAST)

If a replier receives a request but does not have the requested data, the replier ignores the request since it must have sent a similar request of its own. Otherwise, the replier retransmits the data using a *Directed Multicast* (DMCAST). This is the final service LMS adds to the routers; its purpose is to enable fine-grain multicast to eliminate exposure.

The operation of a DMCAST is shown in Figure 4.7. To perform a DMCAST, the replier creates

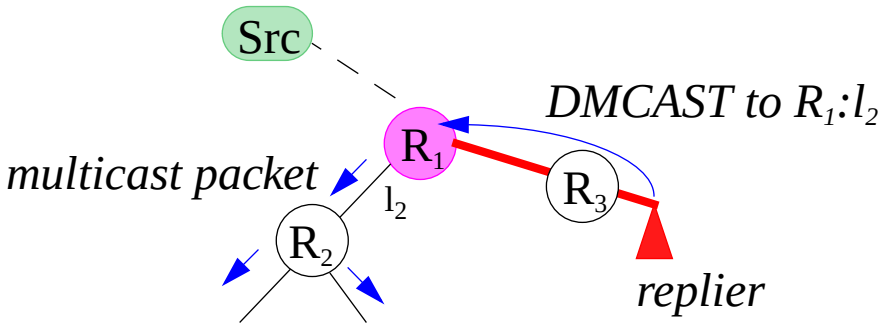

Figure 4.7: A Directed Multicast (DMCAST)

a multicast packet, addresses it to the group, and inserts the requested data. The packet contains an IP option with the turning point link identifier obtained from the request. The replier then encapsulates the multicast packet in a unicast packet and sends it to the turning point router, whose address was again obtained from the request. When the turning point router receives the unicast packet, it decapsulates the multicast packet, strips the IP option and multicasts it on the specified link. From that point on, the packet travels as a regular multicast packet originating from the source.

4.4.4. LMS Summary

In the previous sections we presented the operation of LMS from the router's point of view. Here we summarize the concepts and operation of LMS.

LMS employs three important concepts:

- Each router selects a receiver which acts as a surrogate. The surrogate performs processing on behalf of the router. We call this surrogate a replier.
- Routers are enhanced with steering mechanisms that allow the delivery of special messages to the surrogate. The router simply forwards these messages, and maintains no per-packet state.
- Finally, the routers are enhanced with the capability to receive encapsulated multicast messages and multicast them on specific links, thus delivering them to specific subtrees.

These concepts work together to enable receivers to construct an efficient recovery mechanism. These steps are depicted in Figure 4.8 and summarized below.

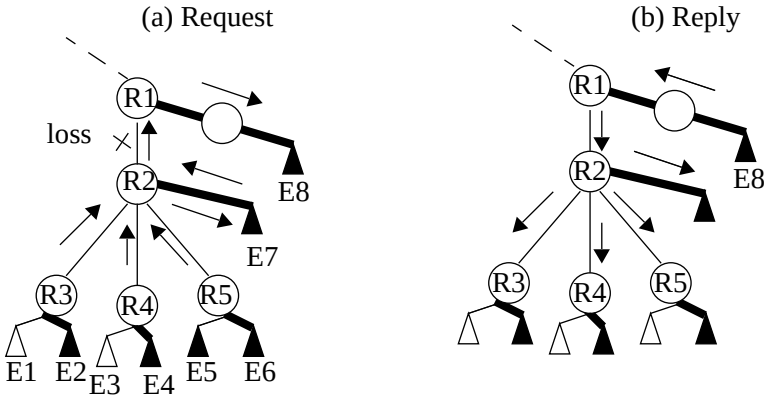

Figure 4.8: LMS Summary

Sending a request:

Replier links are in bold. Loss occurs on the link between R1 and R2. Endpoints E1 through E7 detect the loss. Then, the following events take place:

- E7 sends a request, which R2 forwards to R1 because E7 lies on R2's replier link.
- E1 sends a request which is forwarded by R3 to E2. Similarly, requests from E3 and E5 are forwarded to E4 and E6 by R4 and R5, respectively.
- The request from E2 is forwarded to R2, because E2 is on R3's replier link. Similarly, the requests from E4 and E6 are also forwarded to R2.
- R2 forwards requests from E2, E4 and E6, to E7, which ignores the requests since it does not have the data (but has requested it).
- The request from E7 reaches R1, which forwards it towards E8, which has the requested data.

Sending a Reply

Once E8 receives the request and determines that it has the requested data, it prepares a reply and sends it as follows:

- E8 creates a multicast message containing the reply. E8 encapsulates the message in a unicast message and sends it to R1 (the request's turning point).
- R1 decapsulates the multicast message and multicasts it on the link leading to R2.
- From that point on, all downstream routers and endpoints treat the reply as a regular multicast message coming from the source.

Note that LMS routers maintain no state other than the replier link identifier and cost. Routers need not remember anything about requests that passed through. Routers are not even aware that these messages are sent for data recovery; they simply forward messages according to the new forwarding rules, just as they forward regular multicast packets according to standard multicast rules. Thus, just like normal multicast, no per-packet state is needed at the routers.

4.5. Preventing Duplicate Retransmissions

Multicast error recovery protocols (including those using LMS) that allow receivers to send retransmissions, face an ambiguity problem when requests arrive at receivers asking for data just received through retransmission. We will call these requests *late requests*. The ambiguity problem is depicted in Figure 4.9: assume that the links between R3 and R2 and R4 and R2 are links with

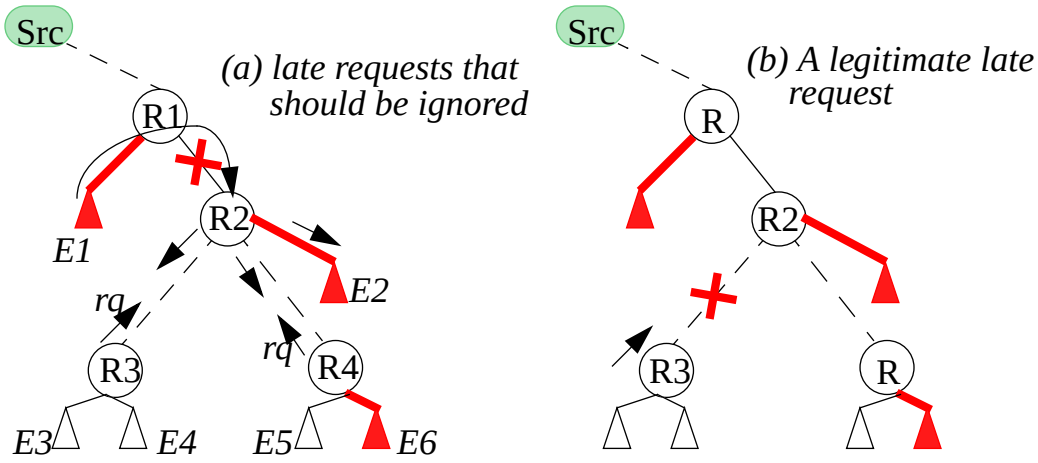

Figure 4.9: Late requests lead to ambiguity

long propagation delay, as shown on the left of the figure. Now suppose a packet is lost between R1 and R2. A request from E2 will reach E1 which will DMCAST the data to the subtree. Now suppose that the requests from R3 and R4 reach E2 after the reply; obviously E2 should ignore these requests. However, if we look at the right side of the figure, late requests may be legitimate if the retransmission was lost again. For example, if the previous retransmission was lost between R2 and R3, the request from R3 is obviously legitimate.

Clearly, the arrival of late requests leads to ambiguity: E2 does not know if the requests have crossed the retransmission, or if they are legitimate requests for a lost retransmission. To overcome this problem, we propose that repliers do not serve late requests, unless they receive a second request. An optimization is for receivers to mark their first request, which can always be safely ignored by repliers.

The above is a conservative approach, but one that ensures that eliminates ambiguity. If some duplicates are acceptable, repliers can serve requests immediately (except those marked as first). Another possible solution is to introduce an "ignore" period at repliers, where repliers ignore requests for some time after receiving a retransmission. We expect that different applications will employ their own method of dealing with late requests.

4.6. LMS Specification: Forwarding Services

In the previous sections we described how error control can be performed using the services provided by LMS. Our discussion intentionally blurred the distinction between the LMS forwarding services and the error control operations at the endpoints. In this section and the next, we clearly separate the two and give a clear description of the actions taken at routers and endpoints. In this section we describe the LMS forwarding services; in the next section we give the error control operations at the endpoints that use the LMS services.

4.6.1. Replier State

The replier state at the routers is created and maintained with *replier refresh* messages. A refresh message carries with it the following information: (a) the <*S, G*> entry this message is for; and (b) an associated *cost*. A refresh message may arrive from any of a router's downstream interfaces for a particular <S, G> tree. For the following discussion, *replier-iface* and *replier-cost* refer to the router's existing replier interface and cost; *new-iface* and *new-cost*, refer to the interface the incoming refresh message was received and the new cost carried by the message. The values replier-iface and replier-cost are initialized to values unattainable in reality (e.g., infinity). Periodically, the router checks all replier entries and expires those that have not been updated within a predefined interval. The following pseudocode describes the router actions after a refresh message is received:

```
if refresh arrived from upstream interface, then
     silently discard packet
else if the router has only 2 interfaces in <S, G>, then
     send refresh upstream
else if replier-iface = new-iface, then
     replier-cost = new-cost
     send refresh upstream
```

```
else if replier-cost > new-cost
          replier-cost = new-cost
          replier-iface = new-iface
          send refresh upstream
else silently discard packet /* replier-cost <= new-cost */
```

4.6.2. Handling Requests at the Router

As described earlier, requests are multicast but arrive at routers carrying a special IP option. The option causes the requests to be picked up by the router and examined more closely. Requests carry an <S, G> entry with them, where S is the address of the source that sent the data being requested. The following are the actions a router takes upon receiving a request. As before, the router's replier interface is stored in the variable *replier-iface*, and the interface the request arrived on is in *new-iface*.

```
if replier-iface not set, then
     send request upstream
else if replier-iface = new-iface, then
     send request upstream
else
     if turning point info is empty, then
          turning point address = replier-iface address
          turning point iface = new-iface
     forward request to replier-iface
```

4.6.3. Handling Directed Multicasts at the Router

A directed multicast arrives at a router in the form of a multicast packet encapsulated in a unicast packet. The multicast packet carries a special IP option with the identifier for the interface the packet should be multicast. Assume that the variable *dmcast-iface* contains the identifier of that interface. Upon reception of a dmcast the following actions take place:

```
decapsulate packet
dmcast-iface = iface carried in IP option
if dmcast-iface is valid, then
```

```
     multicast decapsulated packet on dmcast-iface
else silently drop packet
```

4.7. LMS Specification: Error Control

In the previous section we described the forwarding actions at a router for LMS messages. Note that no error control specific operations were done during these forwarding actions. In these section we specify the host specific actions that perform the actual error control.

4.7.1. Sending Requests

As we mentioned earlier, receivers detect losses using gap detection. When a receiver detects a gap G it performs the following actions:

```
create request(G) /* (see Figure 4.5) */
clear Turning Point information
set request seq num = 0
send request and start retransmission timer
while (reply not received)
     increment request seq num
     double wait interval
     send request
```

4.7.2. Receiving Requests

Assume that a replier receives a request *RQ* with sequence number for gap *G* and the turning point of the request is *TP*; The replier then performs the following actions:

```
if we do not have the requested data, then
     discard request
else if we received data as original transmission, then
     send dmcast (TP, G)
else /* a late request */
     deal with late requests /* application specific */
```

4.7.3. Receiving Retransmissions

As mentioned earlier, receiving a reply in LMS is very much like receiving normal data:

```
if gap exists for retransmission, then
     insert retransmission in the buffer
else discard the packet
```

A fine point in our protocol is that it is possible (as we have seen during our simulations) that a retransmission may arrive before the receiver even detected its loss. This can happen if another receiver experienced a shorter loss and initiated retransmission early. When this occurs, the receiver simply treats the retransmission as original data and initiates requests for any gaps detected by the reception of the retransmission.

4.8. Problem: Duplicate Data Packets

Since there is always one replier in our scheme, no duplicate replies are ever generated. It is possible, however, that a receiver may be exposed to replies generated as a result of recovery initiated by receivers in other regions. An example is shown in Figure 4.10. In this example, a packet is

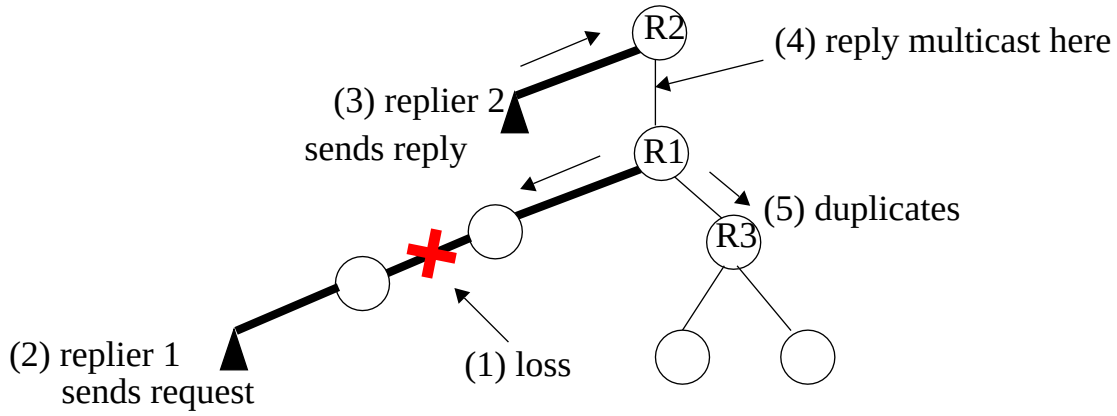

Figure 4.10: Exposure in LMS

lost on the path between R1 and replier 1. Replier 1 sends a request which reaches Replier 2. In response to the request, Replier 2 sends a directed multicast to R2, which multicasts the reply on

the downstream link leading to R1. The reply reaches all of R1's downstream links, causing duplicates on the subtree routed at R3.

Even though this problem does not inhibit recovery, it may lead to the "crying baby problem," where excessive loss experienced in one branch causes duplicates at a large number of other receivers. In LMS this problem is solved by using the cost field to select a replier that advertises the least loss. For example, R1 will select a replier from the right-hand-side branch if this branch experiences less loss, even though the replier on the left-hand-side branch may be closer.

4.9. Source Spoofing

Some routing protocols (e.g., DVMRP) create a separate multicast tree for each sender. With such protocols, the multicast reply resulting from a directed multicast must contain the original sender's address as the source address, otherwise it will not reach the appropriate receivers. To avoid this problem, we allow repliers to use the original source's address in the multicast packet (i.e., perform "source spoofing"). If desirable, to allow receivers to distinguish spoofed from real packets, routers ensure that spoofed packets are marked and include the replier's address in the message. Thus, source spoofing poses no additional security concerns since the real sender can always be identified by the recipient. Source spoofing is unnecessary with routing protocols that create shared trees.

4.10. Dealing with Shared Trees

While most of the multicast capable routers on the MBONE today use DVMRP, some new routing protocols like PIM-SM and CBT create shared multicast trees around a core or rendezvous point, and thus do not maintain per-source information. This enables them to scale to groups with many sources. In order to handle these protocols, we propose the following changes to our scheme, as depicted in Figure 4.11.

We calculate subtree repliers as before for the shared tree. A request is directed by the routers to the repliers as before. Requests from repliers are directed towards the core. In order to guarantee that the source will eventually receive the request, whenever the core receives a request it unicasts it to the source. The source in turn performs a directed multicast to the turning point as before. So

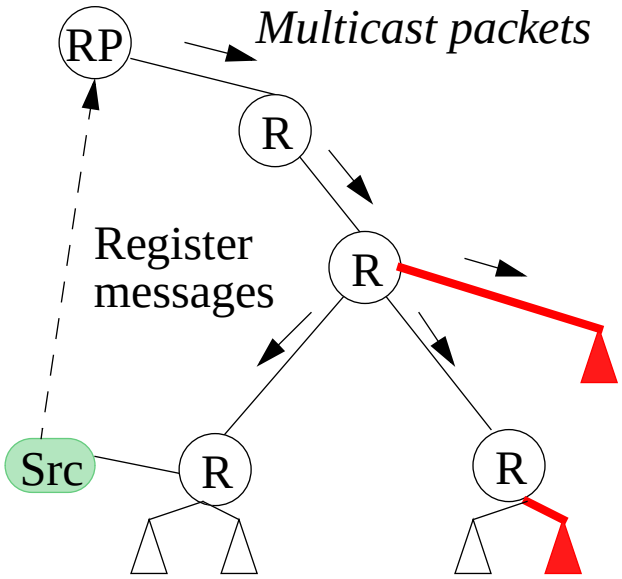

Figure 4.11: Dealing with unidirectional shared trees

rather than have the source directly connected to the root as in DVMRP schemes, the source is connected by a unicast path to the root.

The above modification works well for unidirectional shared trees like PIM-SM. Sources in PIM-SM initially send data to the core via register messages, which are then multicast by the core. Thus the core always lies upstream. If the core sends a join message to the source, then intermediate routers along the join path maintain per-source information, which allows correct routing of requests. However, in CBT, routers can no longer distinguish upstream and downstream links with respect to a source. Thus, LMS in its current state does not work with bidirectional shared trees, unless some per source information is added to the router state.

4.11. Replier Failure

The failure of a replier may disrupt recovery until the failed replier is detected. Note, however, that a failed replier will not always disrupt recovery; replier failure becomes a problem only if either the requestor below the loss or the replier above the loss fail. Thus if the break in the replier continuity does not coincide with the location of loss, recovery will proceed unaffected. Failed repliers will eventually be detected by routers when soft state expires. However, detection via soft state may

take too long; to enable fast detection of replier failure, we propose the following mechanism, depicted in Figure 4.12:

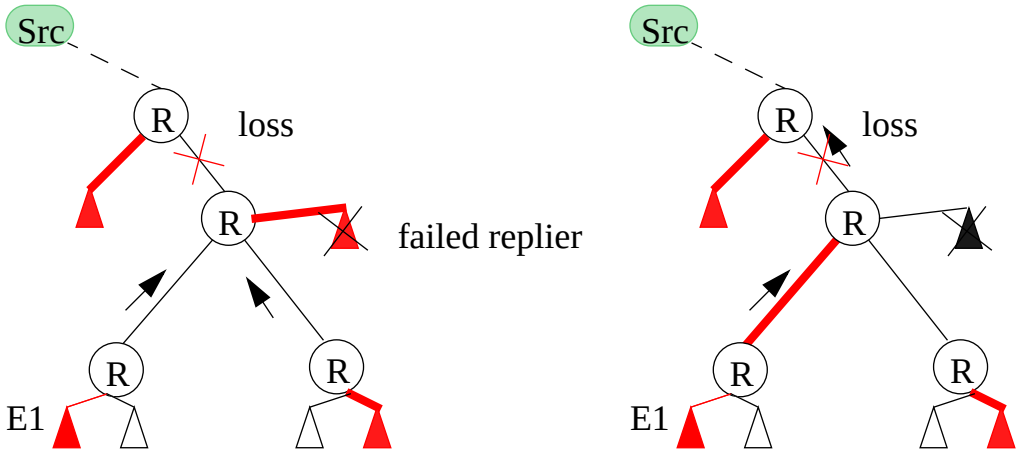

Figure 4.12: Detection of replier failure

- If after sending a request, the receiver's retransmission timer expires, the receiver sends another request; this process is allowed to repeat up to *N* times.

- If after *N* attempts there still no response, the receiver declares the replier unavailable. The receiver then, sends another message to the router at the turning point, alerting the router of possible replier failure.

While detection of a failed replier is taking place, the replier closest to the loss uses *heartbeat DMCASTs* to assure downstream receivers that the failure is being dealt with. A heartbeat DMCAST is special in the sense that the router forwards it on *all* downstream links, including the replier link. The next replier downstream the failed replier takes the responsibility to perform the necessary actions to restore the replier continuity. In the figure, this replier is E1. E1 knows that it is the closest to the loss and should initiate the replier recovery process because it receives no heartbeat messages, except its own. Receivers closer to the loss start their heartbeat earlier. All remaining receivers who receive a heartbeat, defer the repair to E1. These receivers can in turn monitor E1 through its heartbeat. E1 seizes its heartbeat after a retransmission is received, signifying that the replier continuity has been restored.

4.12. Selecting Repliers in a LAN

For simplicity, the previous sections have assumed that only one receiver resides at each router link. This, of course, is not always true. In many cases routers are connected to a LAN where multiple receivers may belong to the same multicast group. A possible LAN/replier allocation is shown in Figure 4.13. In such cases, receivers on the LAN run a simple election algorithm to elect a replier.

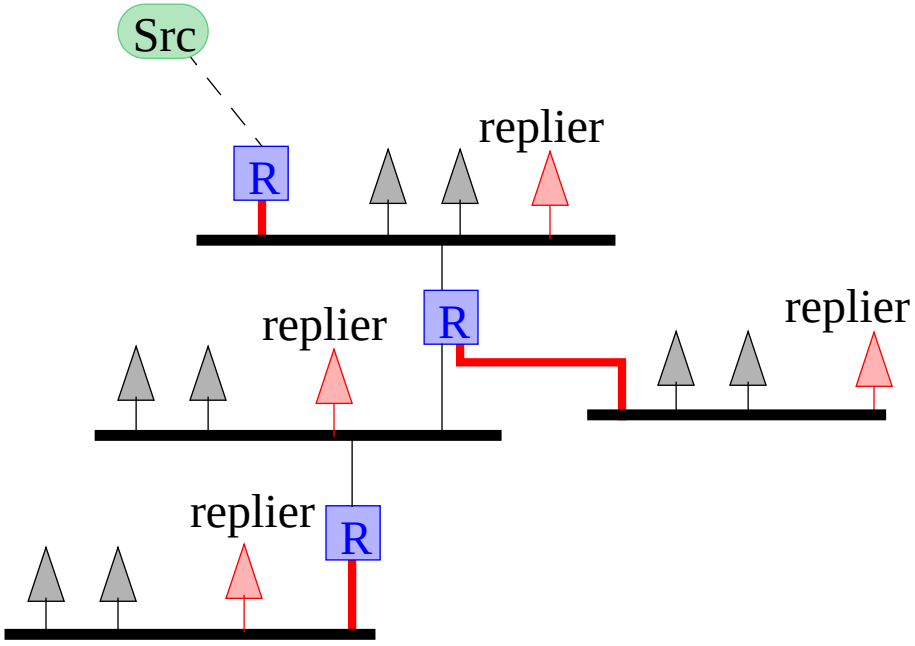

Figure 4.13: Repliers in a LAN

The election takes place without any involvement from routers. Receivers use local multicast (i.e., a multicast with TTL set to 1) for the election. Typically, the first receiver on the LAN becomes the replier; new receivers check for a replier by querying the LAN via a local multicast. If a replier exists, the replier responds with another local multicast. When the replier leaves the group, it sends a local multicast announcing its departure, which triggers an election by the remaining receivers to elect a new replier.

The method by a which a new member discovers existing repliers is depicted in Figure 4.14. Potential repliers upon joining, perform the following actions:

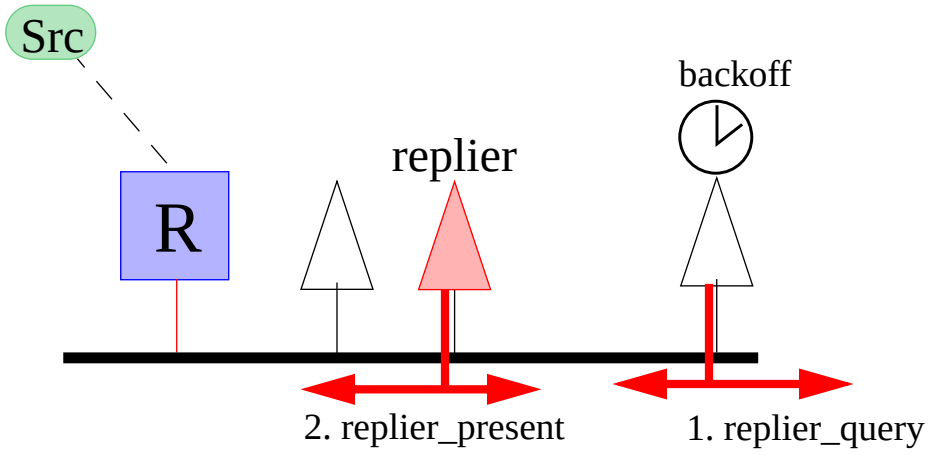

Figure 4.14: Electing Repliers on a LAN

- When a new member that is willing to be a replier joins a multicast group, the member multicasts a *replier_query* message on the LAN (TTL=1) and starts a *replier_query_timer*.
- If another replier exists, it multicasts a *replier_present* message on the LAN. The new member cancels its timer and sets a *replier_present* flag.
- If the replier_query timer expires, the new member declares itself as the replier by multicasting a *replier_present* msg on the LAN.
- Periodically the replier refreshes its status by multicasting a *replier_present* message to the LAN.

When a replier wants to leave the group, it takes the following actions:

- The replier multicasts a *replier_leaving* message on the LAN.
- The other members on the LAN start a new replier election. If multiple candidates are present, the election may be decided based on the advertised cost. Ties can be broken based on IP address.

- If the replier crashes without sending a *replier_leaving* message, the periodic *replier_present* messages will seize. Remaining members detect it and start a replier election.

The router does not participate in this election - it involves only receivers. The router does not need to know which receiver is currently acting as the replier, only that *some* receiver is acting as a replier. It is up to the receivers to ensure that a replier is present in the LAN.

Request Suppression on a LAN

When receivers on a LAN detect loss, they use back-off timers to delay sending requests. The replier, however, multicasts its request immediately, thus cancelling other receivers' requests. This is depicted in Figure 4.15. If loss was internal to the LAN, the replier repairs the loss with a local

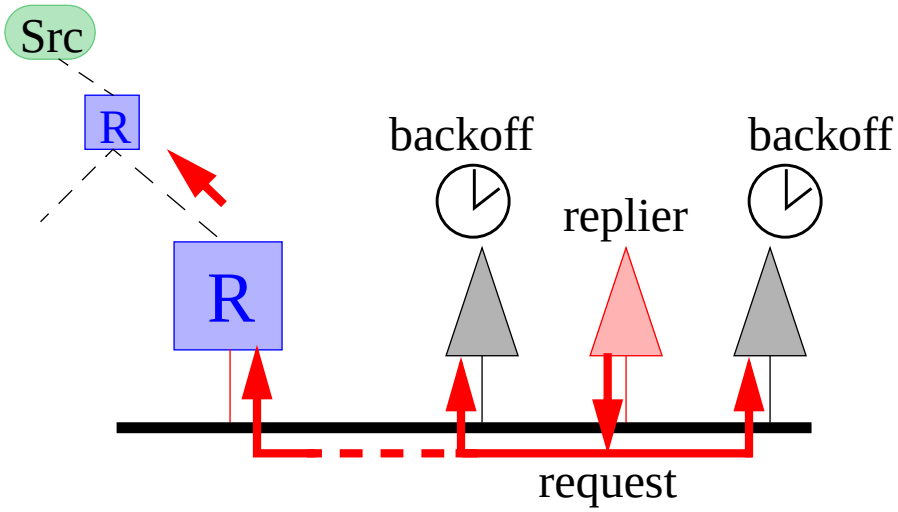

Figure 4.15: Suppressing requests on a LAN

multicast. However, if loss was specific to the replier, the replier's request will cause a duplicate to arrive on the LAN.

When a request arrives on a LAN, the router multicasts the request and all receivers receive it. However, only the replier responds to the request. Other receivers ignore it.

4.12.1. Coordinating Routers Interconnecting LANS

Routers interconnecting LANs need to maintain some extra state in order to forwards LMS messages correctly. An example is shown in the topology depicted in Figure 4.16, between LAN1

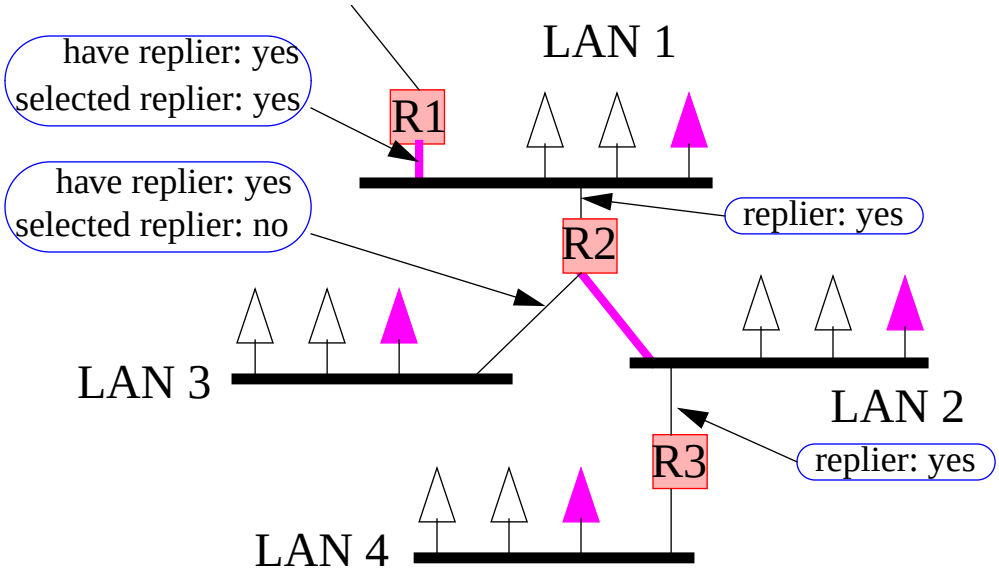

Figure 4.16: Routers connecting LANs

and R2. Suppose a request arrives at LAN via R1, destined for LAN1's replier. R1 multicasts the request on LAN1, where it is received by the replier as we described earlier. However, if R2 is not aware that LAN1 has a replier, then R2 may propagate the request to LAN2, to its own replier. In turn, R3 may do the same, and so on. This would result in multiple replies. To avoid this problem, R2 and R3 need to remember that the upstream LANs have a replier and thus if a request appears on these LANs these routers should not propagate them.

4.13. Routers with a Large Fan-out

If a router has a large fan-out for a specific group, the router's replier may receive a large number of requests from downstream repliers. To avoid this problem, the router may partition its links into smaller groups and select a replier for every group, as shown in the example in Figure 4.17. In this example, requests from links in group *D* go to replier *d*, but requests from replier *d* go to replier *c*, requests from replier *c* go to replier *b* and so on. Requests from replier *a* are forwarded upstream.

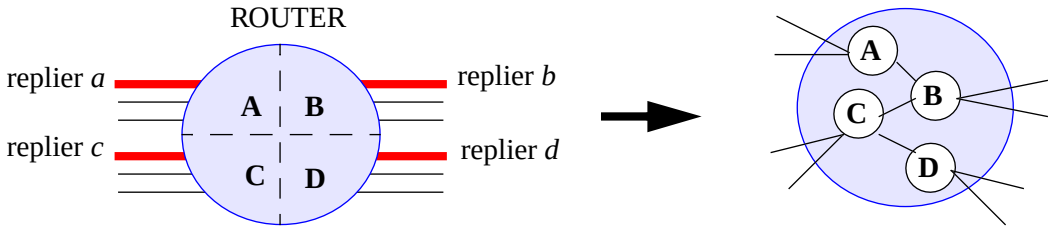

Figure 4.17: dealing with routers with large number of links

By partitioning links this way, the maximum number of requests a replier can receive is significantly reduced.

4.14. Improvements

There are some possible options receivers can utilize to improve the performance of LMS services. These improvements do not change the functionality of the services, only the manner in which receivers use these services. We describe these improvements next.

4.14.1. Proxy Directed Multicast

In the previous examples we assumed that the first replier who receives the request has the data and services the retransmission. This however, may not always be true. For example, since we employ a NACK-based protocol, the request may arrive after the buffers have been purged and even though it identifies the correct turning point, it cannot be served by this replier; or perhaps for security reasons, the retransmissions are only allowed to come from the original source. Even in such cases, the DMCAST service in LMS can still be used almost as effectively as before. We call this proxy directed multicast.

Once a request passes the turning point it contains enough information to uniquely identify the subtree that requires the retransmission. Thus, if a replier receives a request after its buffers have been purged, the replier can either forward the request upstream, or send it to the sender. In either case the request will eventually arrive at a replier which has the data (assuming there is one); this replier can then initiate a DMCAST to the original turning point specified in the request, thus reaching the correct subtree.

In order to preserve the original turning point, routers forwarding a request to a replier first check if the turning point field is empty. If it is not employ, the router does not overwrite the existing information, but leaves it untouched.

4.14.2. A DMCAST that Eliminates Exposure

As we have seen earlier, loss at a replier link may cause duplicate messages to reach some receivers. In this section, we describe a method to eliminate these duplicate messages, at the cost of adding an extra hop to the retransmission. This approach is shown in Figure 4.18. To eliminate

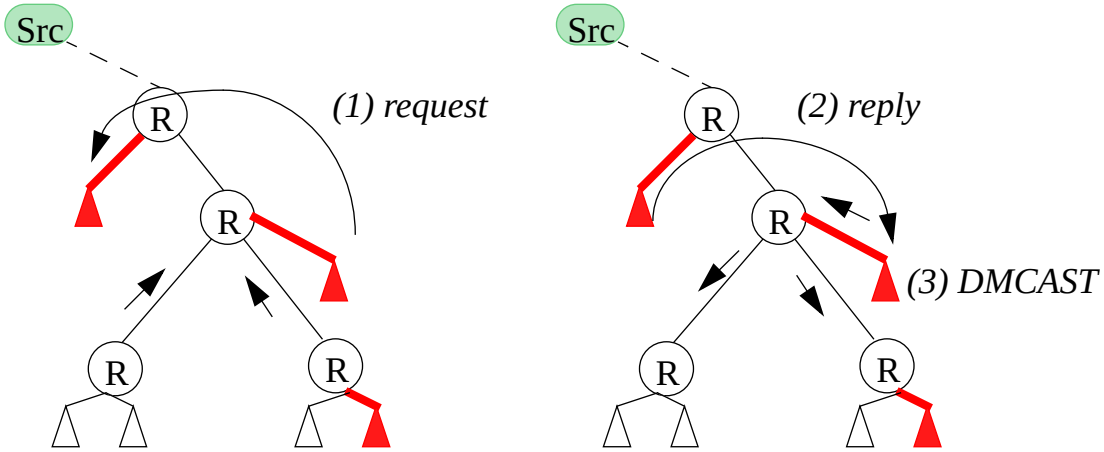

Figure 4.18: Eliminating Exposure

exposure, each request specifies that the reply should be unicast to the requestor rather than sent to the turning point with a DMCAST. If the requestor receives any other requests, either while waiting for the reply or soon after it receives it, then the requestor knows that this is a loss that affected more receivers than just itself. Therefore, it initiates a DMCAST to the remaining receivers. While it would be possible for the requestor to cover the loss by responding to each request with a separate DMCAST, another, perhaps simpler solution is to initiate a single DMACAST to all downstream links at the turning point.

4.15. Some Pathological Topologies

Some topologies may create pathological situations that may reduce the efficiency of LMS. In this section, we describe two such cases and propose possible solutions to overcome the problem. It is not clear how likely these topologies are to occur in real-life. The topologies used for our simulation experiments did not exhibit any such pathologies.

4.15.1. Long and Skinny Branches

These topologies may cause increased recovery latency in LMS. An example is shown in Figure

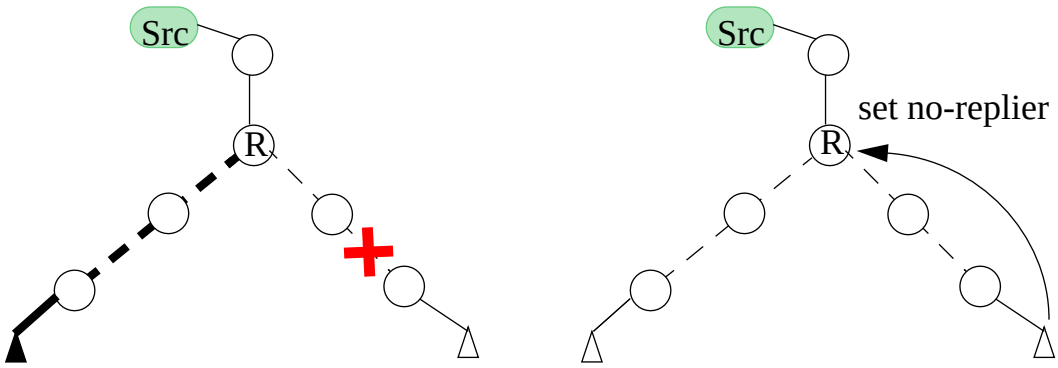

Figure 4.19: Long, skinny branches may increase recovery latency

4.19. These are essentially topologies composed entirely of long, skinny branches with no repliers in the middle. Latency may be increased in such topologies because router R has no choice but to select one of the long branches as the replier link. Thus, requests from a non-replier branch travel all the way back (possibly near the source) only to be diverted down another long branch to the replier. The same applies to retransmissions: they must follow the long path through the turning point.

A possible solution to this problem is the following: receivers that experience long recovery latency (compared to their RTT to the source) due to such topologies, advise the turning point router to release its replier and propagate requests from all downstream branches upstream, in the hope that another replier may be located closer to the turning point. The router may or may not follow

this advice: if the number of downstream links is small thus not risking implosion at the upstream replier, then this is a viable solution. Otherwise, receivers will have to deal with the increased latency. Note that if the router follows this advice, it should continue to insert the turning point information to requests so that dmcasts will be efficient. A disadvantage of this approach is that if loss happens upstream of the turning point, then one DMCAST per downstream link will be required at router R.

4.15.2. Topologies that cause Request Implosion

A pathological topology that may cause request implosion is shown in Figure 4.20. Recall that

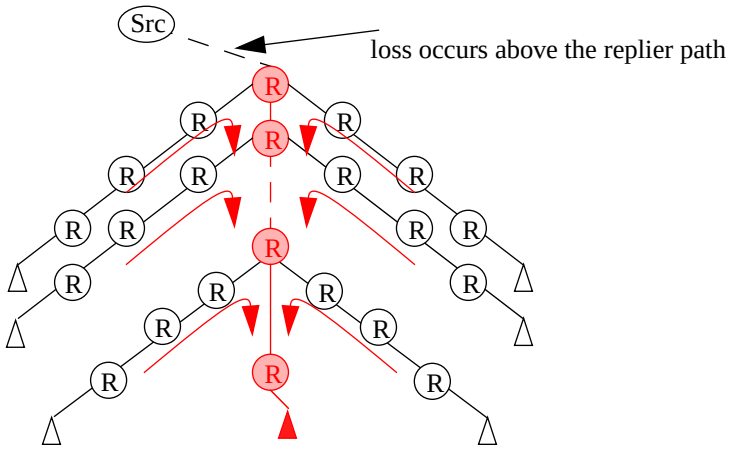

Figure 4.20: A Pathological topology that may cause Implosion

a router forwards at most one request on its upstream link. Thus, the maximum number of requests a replier can receive is typically bounded by the number of downstream links at the turning point. However, it is conceivable that in such pathological cases where many routers have selected the same replier, the replier may receive a potentially large number of requests. In the figure, a large number of neighboring routers (shaded) have selected the same replier (also shaded), forming a long replier path. Every request reaching a router on the replier path is now forwarded to the same replier, making the number of requests at the replier proportional to the sum of the downstream links of all the routers on the replier path.

The problem can be solved by modifying the replier cost to take into consideration the sum of the children of all routers along the replier path. At each hop, a router modifies the cost to reflect the number of its children (minus the replier link). Thus as it moves upstream, the cost becomes high and the next upstream router is forced to select a different replier link, thus shortening the replier path.

4.16. Other LMS Applications

One of the important advantages of LMS is that its mechanisms are general enough to be used for other purposes besides error recovery. For example, the services used for implosion control can be used by many other applications where receivers need to implement a *scalable collect* or *voting* service. Such a service enables a large number of receivers to efficiently convey some value to the sender without risking implosion. The fine-grain multicast capability provided by the DMCAST service can be used for targeting specific parts of the multicast tree, for example to announce the presence of a server in that region. We expand on these applications next.

4.16.1. Simple ANYCAST

ANYCAST [45] is a service used when a client wishes to discover one server (typically the nearest one) out of a group of servers providing a service the client is interested in. Examples are replicated file systems, or the DNS service. LMS can be used to implement a simple ANYCAST service by grouping servers in a well-known multicast group. Servers use LMS to notify routers that they want to be repliers, choosing perhaps distance as the replier metric. Clients searching for servers send requests to the multicast group, which are directed to the nearest replier (as with retransmission requests) thus establishing contact with the server. These steps are shown in Figure 4.21.

One small change required in the replier selection method to implement ANYCAST. Servers tell the routers to advertise the existence of a replier (server) in all links rather than just the upstream link as before. which ensures that routers find the nearest server in any direction.

4.16.2. Positive Acknowledgment-Based Reliable Multicast

Recall from the previous chapter, that NACK-based protocols cannot guarantee 100% reliability due to the lack of positive ACKs. To achieve complete reliability, one should either employ a

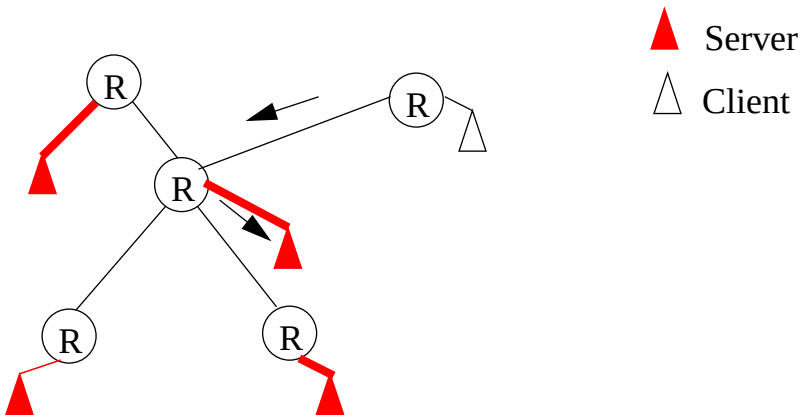

Figure 4.21: LMS implements a simple ANYCAST service

positive ACK-based protocol, or supplement NACKs with periodic synchronization via ACKs. The mechanisms for both approaches are similar; the main difference is the frequency of synchronization, which happens on every packet (or window) in an ACK-based protocol, instead of at a larger interval with a NACK/ACK combination.

LMS can be used to implement either a pure ACK-based protocol, or NACK/ACK protocol. In either case, repliers take the responsibility to collect ACKs from downstream receivers, fuse them, and send a cumulative ACK upstream. Thus the sender finally receives a single ACK containing the highest consecutive sequence number from all receivers.

LMS can help simplify the implementation of such protocols. For example, additional mechanisms that would be needed to assign children to parents (as done in existing protocols using static hierarchy [35]), have been eliminated; tree congruency, a sticking point with other protocols, is now automatic; a joining receiver searching for a parent can easily find one upstream by sending a request; switching to a new parent after the current leaves the group is again easy: it simply requires that downstream receivers send a probe message searching for a new parent; finally, the DMCAST service is still available to send retransmissions efficiently.

4.17. Security Issues

Current multicast security schemes employ encryption techniques to ensure security in multicast groups [46]. With encryption techniques, the receivers are given keys, which they use to decrypt packets sent by the sender. There are two possible problems that may arise with secure applications in LMS. These are the following:

- Since in LMS retransmissions may come from receivers, how can the requestors ensure that the retransmission sent by the replier is the same as the data sent by the source?
- How can requestors ensure that repliers will respond to their requests?

To address the first problem, in secure applications we require that repliers buffer copies of packets in their encrypted state, i.e., as they are received from the sender. Thus, when a request arrives, the entire packet is DMCAST in its original form. In addition, we require that DMCASTs contain the address of the replier in their body (or at least some indication that this is a retransmission) to inform receivers. This prevents a replier from faking original data from the source. This can be done by having routers check that retransmissions not coming from the source contain an IP option that clearly identifies it as a retransmission.

The second problem is addressed with the same mechanisms used to detect replier failure which we described earlier.

4.18. Incremental Deployment

Incremental deployment is an important issue in the deployment of any new scheme affecting routers in the Internet. The Internet has become so big, that a clear incremental deployment plan is essential.

The easiest way to deploy LMS is with the next generation of IP, namely IPv6[62, 42]. IPv6 is currently running on an experimental overlay known as the 6-Bone [63]. LMS can also be deployed following a similar approach as the one proposed for the incremental deployment of PGM [47]. PGM uses *Source Path Messages* (SPMs) to create an overlay of PGM aware routers. PGM signalling then occurs between these routers. The SPM approach works for LMS; it is depicted in Figure 4.22 and described below.

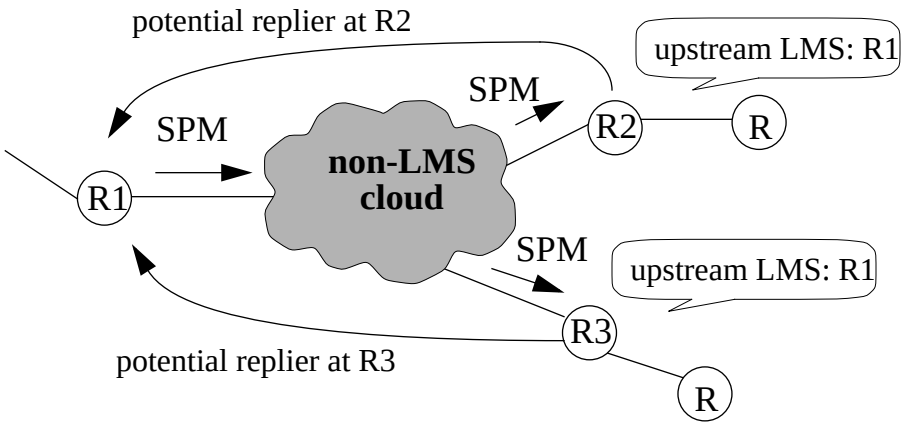

Figure 4.22: LMS Incremental Deployment

Periodically, the source multicasts SPMs to the group. These also carry the IP Router Alert option, so that all routers examine them. Non-LMS routers receiving these packets will simply ignore the option and forward the packets in the normal fashion. When an LMS router receives an SPM, it notes the address of the sender. If the sender is not its upstream neighbor on that link, then it notes the address of the upstream LMS element. It then writes its own address in the SPM and forwards it as usual. This process will eventually create an LMS overlay in the reverse direction. Messages destined for the upstream link in LMS will simply be sent to the upstream LMS element instead.

Note, however, that unlike PGM, LMS requires an additional step. LMS needs to send messages to repliers, so a router needs to know which of the possibly many downstream elements leads to the replier. To achieve this, replier updates are sent upstream through the LMS overlay as described above; when a router selects a replier that lies across a non-LMS cloud, instead of noting just the link it selects the address of the next downstream element. In our example in the figure, R1 receives solicitations from both R2 and R3 for replier selection. R1 will chose one of them, but will save the address of the chosen router as the next replier hop.

Incremental deployment has an effect on exposure. For example, a retransmission initiated at the turning point may reach receivers in a non-LMS cloud. If loss occurred upstream the non-LMS cloud, the retransmission will be useful to all receivers; however, if loss occurred downstream the cloud, receivers in the cloud will receive a duplicate.

4.19. Summary

In this chapter we presented the major contribution of this thesis, namely Light-weight Multicast Services (LMS) and how they can be used to implement a scalable reliable multicast protocol. We discussed how LMS addresses implosion and exposure, which are the main problems with reliable multicast, and showed how LMS solves these problems efficiently and with very little weight. We also identified some limitations of LMS and showed ways to overcome them. Then. we argued that services offered by LMS are general and can be used by other applications besides reliable multicast; such applications include ANYCAST and building a positive acknowledgment-based reliable multicast protocol. LMS offers mechanisms to scalably collect feedback from receivers in large multicast groups which are useful in several types of applications.

We conclude with a summary of the main strengths of LMS:

1. **LMS eliminates implosion:** we have shown that the maximum number of requests a replier will receive, is bounded by the number of downstream links at the turning point. While we expect this number to be generally small, we have shown how a router with a large number of downstream links can organize them in a logical hierarchy to ensure there is no implosion.

2. **LMS maintains low exposure:** while it is possible that some receivers may receive duplicate messages, LMS has the capability to reduce this occurrence by selecting appropriate repliers. However, as we will see in the next chapter, this problem is actually quite small to begin with.

3. **LMS maintains low recovery latency:** receivers in LMS will typically experience recovery latencies smaller than their unicast RTT to the source. Certain topologies, however, may cause latency higher than the RTT to the source. We have shown methods of dealing with such problems when they occur.

4. **LMS simplifies applications:** with LMS receivers do not need to maintain any state related to topology, except for their RTT to the source for purposes of recovering from lost retransmissions (similar to unicast). This is a great asset of LMS. It means that applications need not be concerned with discovering parents, maintaining RTTs to other receivers, or other similar complexity. The work required by applications is comparable to the work done in unicast.

5. **LMS is self-configuring:** this is yet another important asset of LMS: the replier hierarchy is built instantly as receivers join and leave the group, and with the exception of refreshing replier state (which can be done with the refreshing of group membership) is fully transparent to the receivers. Thus the replier hierarchy always mirrors the underlying topology.

6. **LMS requires very little state at routers:** the added state at the routers is minimal: it consists of an identifier for the replier link, and the cost value of the current replier. These can be as small as two bytes each. This is a minimal addition to the existing routing state.

In the next chapter, we use simulation to evaluate the performance of LMS and compare it with two other reliable multicast schemes.

Chapter 5

Simulation Results

Having described LMS in the previous chapter, we now proceed to evaluate the performance of LMS and compare it with two other reliable multicast schemes, namely SRM [17] and PGM [47]. Since SRM is a scheme that uses no network support, our intention is not to directly compare the performance of LMS and PGM with SRM. Such a comparison would be unfair. Rather we use our simulations as a means to demonstrate what is achievable with schemes that use network support with the performance achieved by schemes that do not. The final decision of whether we should introduce router assistance in the Internet or employ purely end-to-end mechanisms is, of course, highly complex and is well beyond the scope of this thesis. We merely hope to help the reader understand the trade-offs in making such a decision.

Most of the evaluation was done via simulation. In this chapter we present the details of our evaluation, including the chosen simulator, the metrics, the evaluation methodology and the simulation results. In our simulations, we compare the performance of LMS with SRM (which that comes bundled with the chosen simulator), and PGM, whose simulation was written with help from Sherlia Shi.

The simulations in this chapter aim to explore the trade-offs, performance and scalability of LMS. By reading this chapter, the reader will understand how LMS performs under various topologies and its limitations. The simulator chosen to carry out our evaluation is the UCB/LBNL/VINT Network Simulator - *ns* (version 2)[48]. The topologies used in the simulations were generated via

GT-ITM[49], which is quickly becoming accepted among the Internet community. Several simulation scenarios were used; the topologies generated although not very large (about 200 nodes), were the largest topologies possible, limited by the amount of memory in our machines. Therefore, we have not tested these schemes with extremely large topologies, but we believe that our conclusions still apply to such topologies.

Our simulation results demonstrate the behavior of LMS, SRM and PGM in terms of duplicates and latency. Our results show that both LMS and PGM perform significantly better than SRM, as one might expect, since LMS an PGM have the advantage of network support. For example, recovery latency in LMS is 30 - 60% of the unicast latency, in PGM is close to the unicast latency, but in SRM it is typically higher than twice the unicast latency. The difference between LMS and SRM is a factor of 5-10 times. LMS performs better than PGM by a factor of 1.5 - 3 in terms of recovery latency, because LMS allows receivers to send retransmissions, whereas in PGM only the source is allowed to retransmit. In terms of duplicates, PGM performs the best, but at the expense of maintaining per-packet state at the routers; LMS performs very well without such a penalty, with duplicates limited to only a few percent. SRM without local recovery generates 4-6 duplicates per lost packet, which is quite high and negatively impacts scalability.

This chapter is organized as follows: in the next section we give our reasons for using simulation to carry out our evaluation, followed by the justification of our selection of simulator. We then give our simulation methodology, and our evaluation metrics. We then briefly describe the topology generation tool, namely GT-ITM. Then we describe in detail the simulation parameters, followed by a discussion on simulation verification. Finally, we present the simulation experiments for each of the three schemes, and conclude the chapter with a summary and discussion of the results.

5.1. Why simulation?

The Internet is a very complex entity - one that we do not yet fully understand [50]. Therefore, evaluating schemes such as LMS, which aim to be deployed on the Internet at a global scale, is very hard. In deciding on how to evaluate LMS, we were confronted with the problem that plagues every newly proposed Internet scheme: proper evaluation can be done only after deployment, but large scale deployment is very costly, justified only after the scheme has gone through extensive testing and is relatively mature. Evaluating schemes like LMS that require modifications to the network

internals (routers) complicates matters even further, because it alters the existing network behavior, possibly changing some of our current limited understanding of the network.

An alternative to a large scale deployment would be to create an LMS-aware overlay over the current MBONE, (e.g., an LMS-Bone), and use it to evaluate LMS at a smaller scale. Such an overlay could be created using tunnels, just like the MBONE is an overlay over the existing Internet. This is the approach that we expect will be followed if LMS is incrementally deployed. However, adopting such an approach at this early stage poses several difficulties:

- we need access to machines on a large number of sites, preferably distributed over a wide geographical area, which is hard to achieve. Then we need to manually configure these machines.
- even if we could gain access to a large number of machines, deploying LMS requires that we modify their kernels. This poses two problems: network administrators, may not, in general, be eager to allow outsiders kernel access to their machines. Additionally, unless we port LMS to several platforms (a daunting task), we would be limited to machines running NetBSD, the current platform of our LMS implementation. Despite being an excellent research platform, NetBSD is not a very popular operating system. The source code for popular OS' like Windows and Solaris is either not easily available, or not available at all, which greatly limits our choices.
- ideally, we would like access to a site's access router, to avoid manually configuring machines to use artificial routes. However, modifying such routers is a dangerous proposition, since testing may create problems that can seriously affect regular Internet traffic.
- debugging a machine located far away is very difficult. For example, when the remote machine's kernel crashes, it is very hard to fix the problem quickly. Typically, kernel debugging is done by directing the console output of a machine to another via a serial line, which would double the number of machines required (that's assuming we could convince the system administrator to install such serial lines and give us access to more machines).
- even if we managed to overcome the above difficulties, testing LMS on a real network at this early stage is still not a good idea. The reason is that the Internet presents a very

complex environment that we have no control of. Thus, if we observed a certain behavior, it might be impossible to deduce if it is due to LMS or some other factors in the network. LMS must be understood well in the laboratory before being tested in the real network.

For the reasons cited above, we decided to use simulation to evaluate LMS. Choosing simulation, in addition to circumventing the above difficulties, offers several other advantages:

- allows an almost arbitrary number of endpoints and arbitrary topologies. A simulation can be configured to use practically any type of topology, either arbitrary, or one modeled after a real topology.
- simulation provides complete control over the environment. For example, during evaluation we like to control loss (both in terms of number of drops and their location). This is very easy to do with simulation.
- simulation allows sharing of code with others. Thus, results can be duplicated and verified by other researchers, who may also want to evaluate other scenarios, leading into a more complete evaluation.
- with simulation, other schemes can also be simulated using exactly the same parameters (topology, loss, transmission rate, etc.). This allows for a better and more meaningful comparison between schemes.

5.2. Why *ns*?

When it came to choosing a simulator, the choice was relatively easy. Initially, we started by creating our own simulator because we wanted to evaluate LMS in isolation; soon it became apparent, however, that this route would not allow us to leverage off a tremendous amount of support provided by simulators like *ns*.

Ns is a public domain simulator that has gained wide acceptance in the Internet community. It was chosen over other simulators for a number of reasons. Ns contains extensive support for existing Internet protocols, including the TCP/IP protocol suite, unicast and multicast routing, traffic generators, SRM, wireless and LAN support. Thus, *ns* offers a very rich simulation environment. Ns comes with the Network Animator (NAM), which is an invaluable tool in understanding and

debugging protocol behavior, and has proven indispensable during the development of LMS. The large *ns* user population encourages code sharing and allows simulations to be shared with other groups. Finally, *ns* is actively being supported, with new modules constantly being added to the distribution. The active support also ensures that bugs are fixed quickly.

Ns, however, has limitations, which the designers are actively working on eliminating. Currently, *ns* is implemented in C++ with a Tcl front-end. The marriage of these two languages, although allowing good flexibility in writing simulations using existing modules, results in a rather steep learning curve and significant complexity when adding new modules. Tcl was chosen to facilitate rapid code development, and as such presents a compromise, trading speed for flexibility. Sometimes this trade-off results in simulations that consume large amounts of memory and consequently, run quite slow. SRM, for example, contains a large portion implemented in Tcl, which severely impacts the size of SRM simulations. In our experiments, we could not run SRM with more than 120-150 nodes on the machines available to us. The LMS and PGM simulations, which are implemented mostly in C++, were much more efficient, both in terms of run-time and memory usage. LMS simulations with 200 nodes would typically finish in a few hours on a 64MB Pentium class machine, whereas SRM simulations would take up to 3 days for 120 nodes on a 1 GB UltraSparc machine.

Despite its limitations, *ns* has served our purposes well. While there is a relatively steep learning curve and much time was expended studying the code and understanding the internals (documentation is sparse but improving), the effort paid off in allowing us great flexibility in our simulations. During our encounter with *ns*, we have also identified some limitations that we believe should be addressed in future releases of *ns*. The most notable example was *ns*' lack of support for router processing. There is currently no way in Ns for routers to extract packets from a stream, process them and then put them back on the stream, as would happen for example in processing IP options. This was an important limitation when implementing LMS, and we had to modify the *ns* internals in our simulations.

5.3. Simulation Methodology

In this section we describe the simulation methodology used to evaluate LMS. The same methodology was used to evaluate SRM and PGM. However, since LMS is the main subject of this study,

our efforts were focused primarily on LMS. For example, we refrained from any attempts to modify SRM and PGM to improve performance, and we did not modify their designs to overcome limitations that were discovered (see the repeated retransmissions experiments with PGM, for example) during our evaluation as this is beyond the scope of our work. However, for experiments common to all three schemes, we kept all the relevant simulation parameters (e.g., topology, loss rate, source rate, etc.) identical.

None of the three schemes comes with a fully functional congestion control mechanism. LMS and SRM have no congestion control at all; PGM sketches out a congestion control mechanism in its specification. However, the authors admit that it is still at the research stage and far from being ready for deployment. While multicast congestion control is an orthogonal issue to error control - the problem that LMS, SRM and PGM are primarily trying to address - and is still the subject of on-going research, it has significant impact on the evaluation methodology. Lack of congestion control implies that we should be careful about how we introduce loss in our simulations: for example, it is of little value to induce loss by adding background load to simulate loss caused by congestion, as we expect to happen in the real Internet. But since these schemes would never be unleashed on the Internet without some congestion control mechanism, the simulation results would be of little value.

While the lack of congestion control means that these schemes must be evaluated further once such a mechanism has been added, it does not mean that useful evaluation cannot be carried out at this point. On the contrary, even if congestion control were available, it is highly beneficial to evaluate the error control characteristics of these schemes separately, in order to study the error control behavior in isolation. While there can be significant interaction between error and congestion control that might necessitate later design changes (for example, a slow-reacting congestion control mechanism combined with a fast-acting local recovery might send packets straight into heavy congestion by triggering retransmissions too fast), studying them separately keeps the parameter space smaller and more manageable. Moreover, multicast applications requirements are extremely diverse, implying that is highly unlikely that one error or congestion control scheme will dominate and completely displace all others. Thus, it is very likely that we will see the development of a variety of modules, each suited for a particular class of applications. The modular design means that

error and congestion control modules may be paired in various ways, and selecting a good combination requires a clear understanding the workings of each module individually.

In order to isolate the operation of error control, we opted to use artificial packet drops. In other words, drops were induced by deliberately creating loss on the target link. The overall link bandwidth was kept high and the source rate low in order to ensure that any queue buildup at the routers which might cause further packet drops was eliminated. Although our loss module could be configured to drop packets according to any loss model (e.g., random, exponential, bursty, etc.), we concluded that such types of loss would be of little value. The reason is that since we are only concerned with error control, we are not interested in determining results like throughput, which would be affected by the type of loss used; rather we concentrate on evaluating the overhead of recovery in each scheme, *after* a loss has occurred. For the same reasons, we model drops of original packets only; retransmitted packets are never dropped.

Numerous simulation runs were carried out, in order to explore the behavior of LMS in a wide range of topologies and scenarios. Identical scenarios were used where possible with SRM, and PGM, keeping all common simulation variables the same.

5.4. Evaluation Metrics

As mentioned in the previous section, our evaluation is mostly concerned with what happens after a loss is detected. We do not use traditional metrics like throughput because the overall bandwidth seen at each endpoint depends on the aggregate loss, which in turn depends on loss type and location and are very hard to predict. While some studies have been carried out to attempt to characterize loss in the Internet [39, 51], their results have not been conclusive. Moreover, it is not clear that studying the MBONE at its current state, i.e., a sparse overlay, with (frequently) restricted multicast bandwidth, will yield results which will still be valid when multicast becomes commonly used in the Internet.

To avoid making our own (and most likely flawed) assumptions about the future loss characteristics of the MBONE, we limit our study to just two performance metrics that approximately characterize the overhead associated with each of the schemes. This is a similar approach followed by the SRM designers in their evaluation. For consistency, we use the same metrics that have been used

in evaluating other reliable multicast schemes [31], but then we add some new metrics. These metrics are, (a) *normalized recovery latency,* (b) *exposure* (for LMS), (c) *requests/repairs per drop* (for SRM), and (d) *repeated retransmissions per drop* (for PGM). Of these metrics, normalized recovery latency and requests/repairs per drop have been used before; exposure and repeated retransmissions are new metrics that we have introduced. We describe these metrics next. The definitions appear in Figure 5.1.

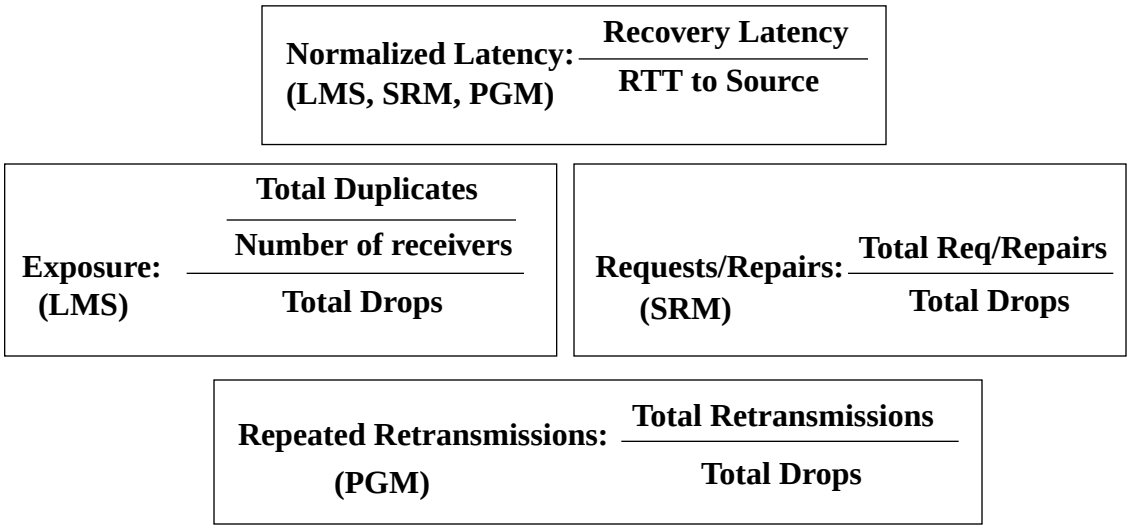

Figure 5.1: Simulation Metrics

5.4.1. Normalized Recovery Latency (LMS, SRM, PGM)

Normalized recovery latency is a metric first used by the SRM designers in evaluating the performance of SRM. It is defined as the latency a receiver experiences from the moment it detects a loss until the loss is recovered, divided by that receiver's round-trip time to the sender. Thus, a normalized recovery latency value equal to 1 means that recovery takes exactly one round-trip time; a value greater than 1 means recovery takes longer than the round-trip time, and a value less than one means recovery takes less than one round-trip time. While it may not seem obvious at first glance, recovery can take less than 1 RTT (from a certain receiver's perspective) when a receiver near the source initiates recovery by sending a NACK sooner than a receiver which is further away. Distant receivers benefit in this case, because a retransmission maybe initiated even before distant receivers have detected the loss. All three schemes use gap detection to detect missing packets. However, in

LMS the normalized recovery latency equals the latency from the moment an out of-sequence packet is received, which in LMS is the same as when a NACK is sent, until the packet is recovered, divided by the round-trip time of that receiver to the source. In SRM and PGM, NACKs are not sent immediately upon the detection of a gap, and thus normalized recovery latency includes the back-off time in these schemes.

Another way to look at normalized recovery latency is as a measure of recovery latency compared to the unicast case, where recovery takes at least one RTT. Thus, normalized recovery latency is a measure of how much better (or worse) latency a receiver experiences as a result of using multicast instead of unicast.

Recovery latency is an important measure of the scalability of a reliable multicast scheme. Since all of the described schemes use receiver-reliable semantics (where the receivers, not the sender, are individually responsible for recovery), high recovery latency may result in failure to recover because retransmission buffers were purged. Conversely, low recovery latency will allow the sender (and the receivers) to purge buffers early, resulting in lower resource requirements. Finally, while some applications may be less sensitive to recovery latency (like file transfer), for others latency may be critical (e.g., multimedia applications), thus lower the recovery latency may result in higher application utility.

5.4.2. Exposure (LMS)

The second metric we used in our evaluation is exposure. This metric had not been used in any other study before. It applies to LMS because LMS has local recovery. Exposure is the average number of duplicate messages received by a receiver as a result of loss at some part of the multicast tree which, may or may not have affected the receiver. It is similar to the next metric, duplicate requests/repairs, used by SRM, but it takes into account the presence of local recovery.

Exposure is a general metric, which attempts to capture the number of unwanted messages that reach a receiver as a result of recovery. Recall from earlier chapters that depending on the recovery scheme, a receiver may receive one or more requests or replies, none of which it may need. For example, a scheme like SRM which has no local recovery, will force all receivers to receive all packets generated in response to a loss. Moreover, because of SRM' s trade-off between latency and duplicates, receivers may actually receive multiple copies of the same request and the same reply.

LMS, due to its local recovery mechanism, only allows one request to escape, which reaches a single receiver outside the loss tree to ask for a retransmission, and therefore creates no duplicate requests *outside the loss subtree*; it allows multiple requests to reach receivers inside the loss tree, but we did not count these as duplicates because these receivers have been affected by loss[1]. On the other hand, LMS does not prevent retransmissions from reaching parts of the tree that do not require them; these we do count as duplicates.

We did not use exposure as a metric to evaluate PGM, because PGM (in most cases) will not allow a receiver to receive a retransmission unless it has specifically requested it. Therefore, PGM does not suffer from exposure.

Like latency, exposure is an important measure of scalability. Schemes with high exposure (like those without local recovery) are less likely to be as scalable as schemes with low or no exposure. Each unwanted message not only wastes network resources, but it may contribute to congestion and incur unnecessary burden at the receivers in the form of interrupts and protocol processing. For sufficiently large groups and non-negligible loss probability, high exposure may result in possibly doubling (or more) of the required bandwidth for the group, since nearly every packet may be lost at some link. High recovery traffic may also adversely impact congestion control since receivers must take into account both original and retransmitted data, in making congestion control decisions.

5.4.3. Duplicate Requests/Repairs (SRM)

Since SRM has no local recovery at the moment (see Chapter 2), we used the same metrics employed by the SRM designers in their evaluation. In addition to normalized recovery latency, we measured the total number of requests and repairs generated for each packet drop. Note that this metric is related to (but not the same as) the exposure metric used for LMS.

5.4.4. Repeated Retransmissions (PGM)

Recall from Chapter 3, that in PGM all retransmissions come from the source (ignoring the presence of DLRs for the moment). Also recall that NACKs set up state as they travel towards the source, and retransmissions erase that state as they travel towards the receivers. In some topologies,

1. In a strict sense, these requests constitute overhead, because they result in no useful work for recovery. However, they may be useful in another context (see discussion on other uses of LMS).

it is possible that the retransmission state created by a nearby receiver's NACK may be wiped out by a retransmission even before the NACK from a distant receiver has the opportunity to update the state by marking more links as needing the retransmission. In these cases, the sender is forced to send multiple retransmissions to repair the same loss. In our experiments we measure how often this situation arises and how many retransmissions the source has to send to repair a single loss.

In PGM, the source is already burdened with serving every retransmission request. Repeated retransmissions, if they occur at sufficiently high volume, will increase this burden even more. One way to reduce the problem is with the deployment of DLRs, which increases the resources required by PGM. Alternatively, repeated retransmissions can be reduced if the source delays a retransmission until NACKs are given ample opportunity to set up the appropriate state in the network; This, however, increases recovery latency. How to determine the appropriate value to delay the retransmission is a subject of on-going research by the PGM designers.

5.5. Topology Generation

Topology generation is an important aspect of our simulation experiments because the performance of multicast protocols is often sensitive to the underlying topology [52, 53, 54]. By topology we mean both link topology, i.e., the way links are connected together, as well as receiver topology, i.e., the way receivers are distributed over the multicast tree. Both link and receiver topology affect recovery in a similar manner. Consider, for example, the factors that determine how many receivers are served through a particular link. In wide-area networks connected through a backbone, few links are typically used to connect autonomous segments (ASs) to the backbone, and one AS to another. These links are more likely to carry packets that will be delivered to a large number of receivers, meaning that this link topology may result in many "heavy[1]" links. In a mesh topology, more links may be part of the multicast tree for a given set of receivers, and thus each link may have less receivers downstream, which results in "lighter" links. Receiver distribution affects recovery in a similar manner: tight clustering of receivers will result in more heavy links, whereas even receiver distribution along the multicast tree will result in lighter links.

1. We call them "heavy" in an attempt to capture a feel for the number of receivers "hanging" off them.

To summarize then, topology affects recovery because the location of a packet loss changes the number of affected receivers. Typically, a packet loss near the source results in most receivers not receiving it, but if a packet is lost near the edges, only a few receivers may miss it. In addition, topology may affect recovery in other manners which are protocol specific: for example, schemes that rely on receiver collaboration for recovery may perform better with receivers which are evenly distributed, or have widely varying RTT to the source.

5.5.1. Topology Generator: GT-ITM

The Georgia Tech Internet Topology Models (GT-ITM) [49] was born out of the observation that a large number of simulation studies of algorithms and policies on the Internet have been carried out using topologies generated by models that are far removed from real topologies. Such models included regular topologies (e.g., rings trees and stars), "well known" topologies of existing networks, and random topologies. These three models either offer poor approximations to real topologies, or apply only to specific networks.

GT-ITM attempts to rectify this problem by developing topology construction models that have characteristics which resemble real networks. The result is a topology generator that is both flexible and fast in generating arbitrary size topologies, with characteristics similar to real Internet topologies. Contributions of GT-ITM include the Locality Model, which uses edge length in determining edge probability to construct random graphs, and the Transit-Stub model, which is a hybrid generation method that combines smaller random graphs. The transit-stub model is a structure that closely resembles that of the Internet, and unlike random models, the TS model can generate large graphs while keeping the average node degree low. GT-ITM has been recently incorporated into the *ns* distribution for automated scenario generation.

5.5.2. Selected Topologies

For our simulations, we selected to experiment with three types of topologies: binary trees, random topologies and transit-stub topologies. Random and TS topologies were generated with GT-ITM. The binary tree topology, while an unrealistic topology (i.e., unlikely to be encountered frequently or at a large scale in the Internet), was chosen because it represents a regular, easy to visualize topology. It is also a good topology to test cases with many receivers, as the number of receivers scales exponentially with the height of the tree. Binary trees are a difficult case for both

randomized and hierarchical protocols: randomized protocols have difficulty selecting appropriate timer back-off values when the distance of all receivers from the source is approximately the same; hierarchical protocols have difficulties selecting appropriate helpers when all receivers are equally good (or bad) candidates, since they are located at the leafs. Thus, with binary trees both types of protocol are tested under adverse conditions, but with a topology that is easy to visualize.

Random topologies generated with GT-ITM, are representative of the topologies one might encounter in dense (richly connected) parts of the Internet, such as in a large organization or a campus network. We expected (and our results have confirmed it) that random topologies would be a relatively good case for both types (i.e., randomized or hierarchical) of protocol. Transit-stub topologies were also generated with GT-ITM, and are representative of wide-area topologies, composed of a relatively sparsely connected backbone leading into dense clusters. Transit-stub topologies are hierarchical, and we simulated 3 levels of hierarchy in our experiments, reflecting the common backbone-to-ISP-to-local-organization structure. TS topologies are a more difficult case than random topologies, because receivers within a cluster have similar latency from the source, but not as hard as binary trees. Sample topologies are shown in Figures 5.2 and 5.3.

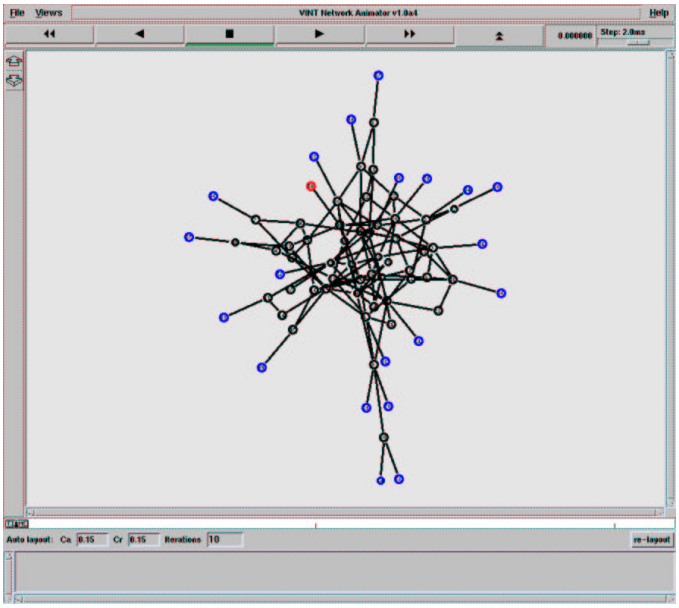

Figure 5.2: Sample random topology

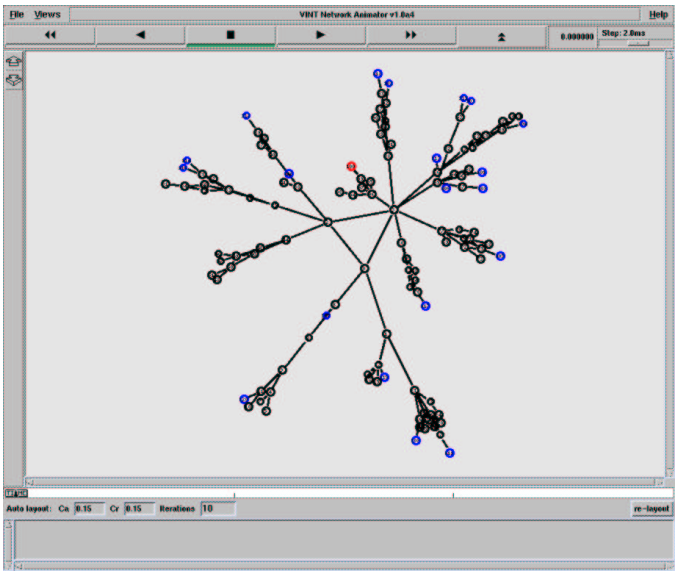

Figure 5.3: Sample Transit Stub topology

While other types of topology may be easily simulated (and we encourage users of our contributed *ns* code to do so), we do not feel that simulating more topologies at this point would have offered any further insight, at least for terrestrial networks. If LMS is to be used in other types of networks (i.e., satellite or wireless networks) then other types of topology may be more appropriate. We have not explored the performance of LMS in these types of networks.

5.6. Simulation Parameters

As described in the previous section, we considered three types of topology in our simulations, namely binary trees, random and transit-stub. The link bandwidth for all links was set to 1.5 Mbps, and the source rate at 10 packets/second, with packet size ranging from 240 to 1024 bytes. Note, however, that the actual values of these parameters are not important as long as they are selected such that no congestion is observed on any link. The reason is that we are interested in studying what happens after a loss occurs (see previous discussion). The link delays were set by GT-ITM at topology generation.

For binary trees we simulated trees of height ranging from 3 to 7 (8 to 128 receivers). For random and transit-stub simulations, we used topologies containing up to 200 nodes (100 internal

nodes and 100 receivers). We generated 10 random and 10 transit-stub topologies, each containing 100 nodes. We run simulations with 5, 20 and 100 receivers, randomly distributed on the internal nodes. For each topology we ran 10 simulations, each time with a different receiver allocation (generated by feeding a random seed to the random number generator). Thus, each presented plot is the result of 100 simulation runs. For random topologies the receiver placement was random on all internal nodes; for transit-stub topologies receiver placement was also random, but restricted to stub nodes.

As discussed earlier, the loss characteristics of the current MBONE are still unknown. Thus, we did not attempt to make any assumptions about the loss characteristics (e.g., loss duration and location); instead, we chose to collect results using a single packet loss; similarly for loss location, (which has significant impact on performance), we chose to investigate the following three cases:

- **Loss at the source:** with this type of loss a packet is lost on the link from the source to the network so that none of the receivers receives the lost packet. This case tests how a scheme responds to widespread loss, e.g., how it avoids NACK implosion.
- **Loss at each receiver:** here loss affects only a single receiver. We ran a simulation with loss on one receiver; then, we moved the loss to the next receiver and ran the simulation again, until all receivers were covered. This is the opposite of the previous case, and tests how the scheme reacts to localized loss, which affects only a small portion of the receivers.
- **Loss at each link:** this case attempts to cover the ground left unexplored by the previous two. Loss is moved from link to link with an entire simulation run each time, until all links are covered. This is equivalent to random loss where all links have equal loss probability.

While we certainly do not claim that these cases represent real loss characteristics of the MBONE, we believe that they provide sufficient information to attain a basic understanding of the behavior of the error control schemes under a variety of loss patterns. As we have pointed out in the previous chapters, for all schemes there are pathological topology/loss combinations that lead to severe degradation in performance. It is too early to tell if any of the pathological cases will actually appear in real networks. As we learn more about the loss characteristics of the MBONE, updated loss models can be plugged in our simulations to obtain more precise results.

5.7. Simulation Verification

Our simulations are composed of two separate components, namely the forwarding services offered by LMS, and the reliable multicast protocol. We verified the correct operation of these components using two separate methods: (a) with simulation traces, and (b) by visualizing the packet flow using the *Network Animator (nam)* [55].

5.7.1. Verification using traces

Our simulations contain a substantial amount of debugging code which can be turned on or off with a flag. When the debugging code is turned on, a detailed packet trace is generated as the packets flow between the sender, routers and receivers. We have generated numerous such traces and manually verified the correct operation of both the forwarding services and the error control mechanisms.

5.7.2. Verification using Nam

Nam is a companion tool to *ns*, which allows visualization of traces generated by *ns*. Nam provides operations like play, pause, fast-forward, rewind and reverse playback of *ns* traces. We have found nam to be an extremely valuable tool in debugging our simulations, and we have used it almost religiously. Nam was well suited for debugging the forwarding services because it allowed us to see at a glance if a packet was either misforwarded or not generated all. Similarly, it was invaluable in debugging our error control mechanism because of its deterministic nature. We have created numerous nam animations during the course of the development of our simulation and spent a substantial amount of time ensuring that the animations portrayed the expected behavior.

5.8. LMS Experiments

We have simulated LMS in a variety of topologies and various session sizes. We only simulated a single source sending data to many receivers (also referred to as a one-to-many communication). We do not expect that results would change with multiple sources.

It is very important to note that unless otherwise stated, in the LMS experiments the repliers were kept static. As we discussed earlier in Chapter 4, the reason is twofold: (a) without knowledge of the loss characteristics of the MBONE, it is hard to devise an efficient replier adaptation scheme,

and (b) we wanted to explore the performance of LMS with simple replier allocation. With static repliers, the results presented for LMS are not optimal. This is especially true for exposure, where the replier selection is the most important factor governing performance. As our experiments will show, LMS performs very well even with static repliers.

5.8.1. Binary Trees

In the first experiment we simulate LMS with binary trees ranging in height from 3 (8 receivers)

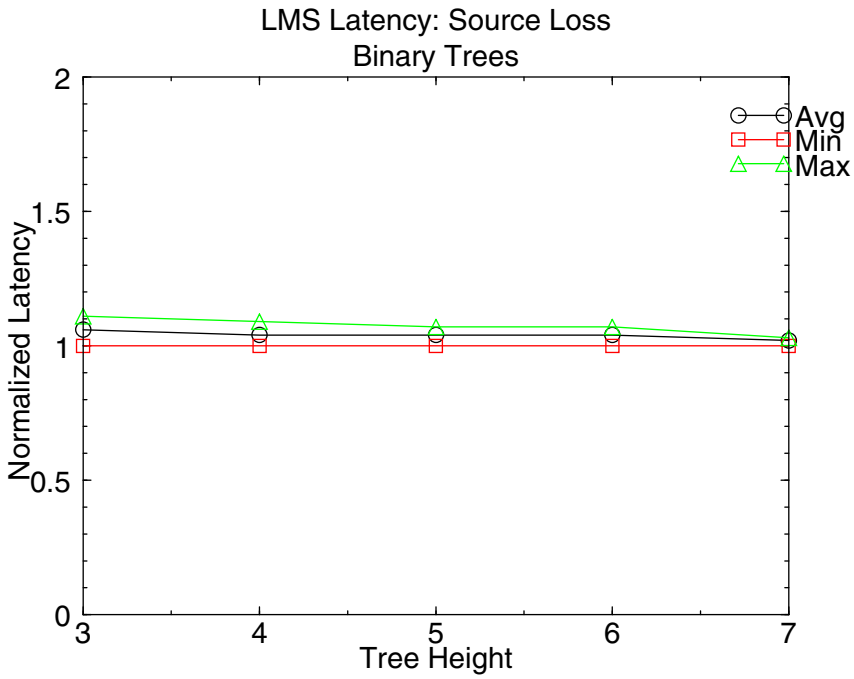

Figure 5.4: LMS latency, binary trees, loss at source

to 7 (128 receivers) and loss at the source. The results are shown in Figure 5.4. We plot the average, minimum and maximum recovery latency. The x-axis lists the five different topologies used in the experiment and the y-axis the normalized recovery latency. We observe that since all receivers have the same RTT to the source, the recovery latency is close to 1. The minor deviations are caused by slight queueing time due to synchronized requests. Note that the recovery latency does not change much as the tree height is increased.

In the next experiment we move the loss to the receivers. The results are shown in Figure 5.5.

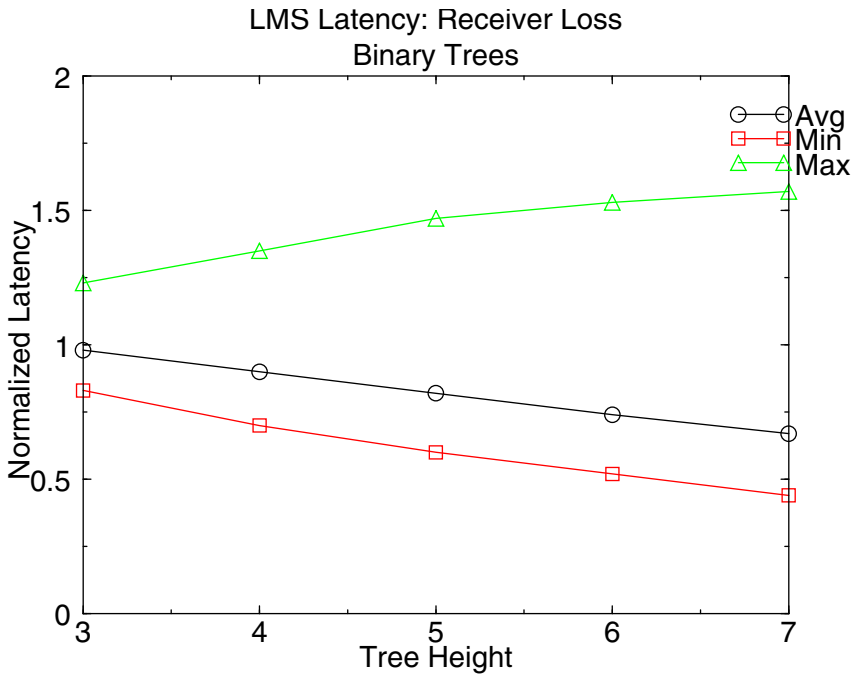

Figure 5.5: LMS latency, binary trees, loss at receivers

Here we observe that the average recovery latency decreases as the tree height increases, so for larger trees recovery is faster on average. The maximum latency increases, because loss at some receivers causes a NACK to propagate back towards the source only to be turned around and delivered to another receiver instead. This causes a loss to be recovered from a receiver that happens to be further away from the requesting receiver than the source, thus increasing the recovery latency to a value beyond 1. Note, however, that in binary trees the recovery latency can never exceed 2. The reason is as follows: the maximum distance (in number of hops) between any two receivers is given by the expression: $d = 2 \cdot (h-1)$. As the tree height increases, the maximum normalized latency becomes:

$$\lim_{h \to \infty} \frac{2 \cdot (h-1)}{h} = 2$$

In the final experiment to measure recovery latency, the loss is moved around from link to link

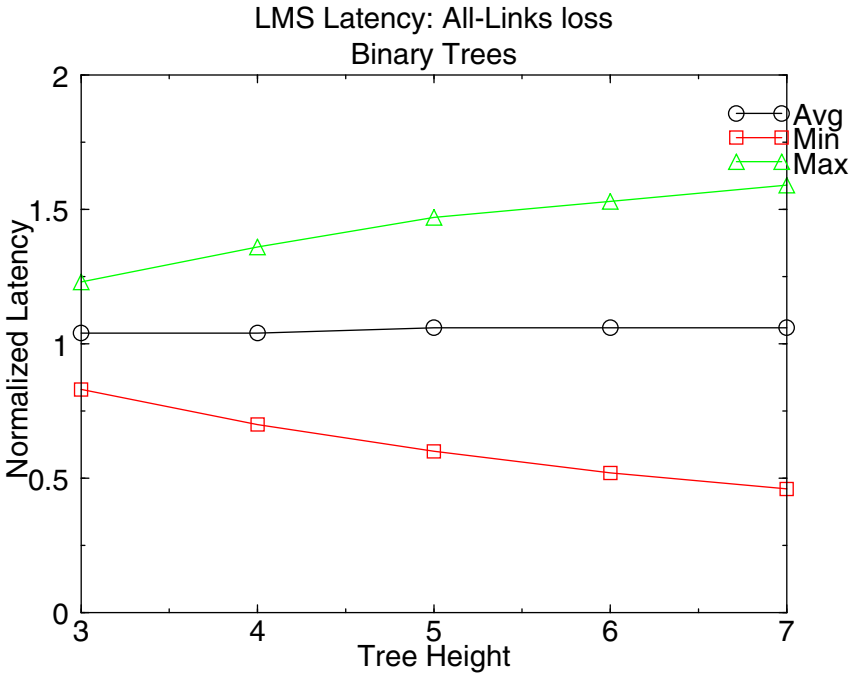

Figure 5.6: LMS latency, binary trees, loss at all links

until all links are visited. The results are shown in Figure 5.6. As the tree height increases, we observe that the average recovery latency remains unaffected and close to 1. The maximum latency increases, but for the reasons described earlier, it will never exceed the value of 2. The minimum latency decreases, because as the tree gets taller the minimum distance between two receivers remains the same while their distance to the source increases.

In the last LMS experiment with binary topologies, we measure the exposure as the tree height increases. Recall that in these experiments repliers are kept static. The results are shown in Figure 5.7. We measured exposure for all loss cases, namely source, receivers and links. Note that the exposure when loss is at the source is zero because all receivers lost the original packet and thus need the resulting retransmission. For the remaining cases, exposure not only starts out low (less than 15%), but it decreases quickly as the size of the tree increases, despite the fact that repliers are static.

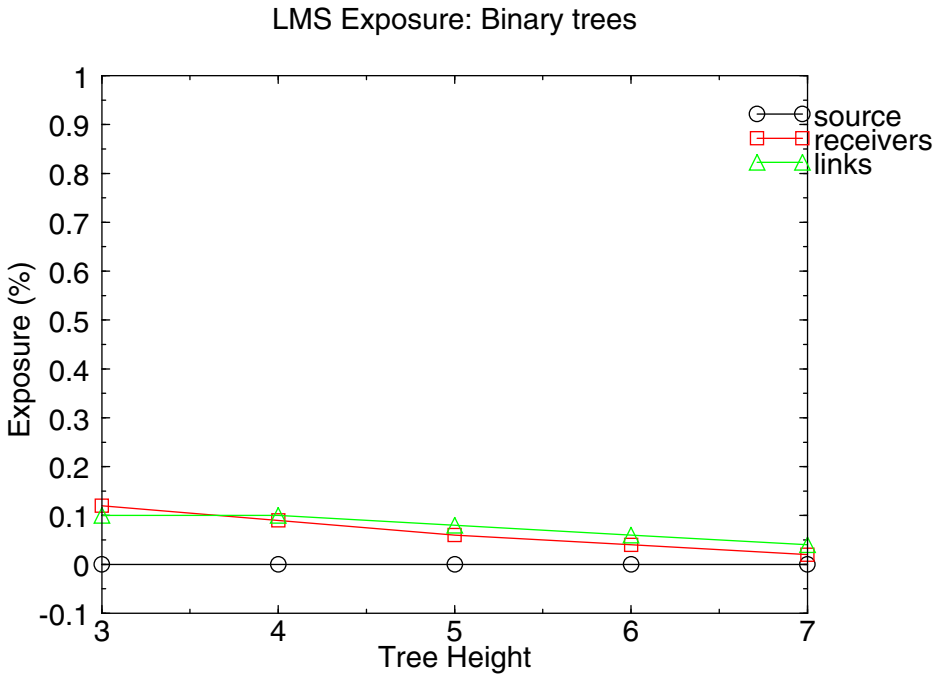

Figure 5.7: LMS exposure, binary trees, loss at all links

In summary, even though binary trees are a difficult topology for LMS because there are no helpers in the internal nodes to speed up recovery and reduce exposure, LMS still manages to perform well. It recovers losses at about one RTT, and keeps exposure very low.

5.8.2. Random Topologies

As mentioned earlier, the random topologies we used in our experiments were generated with GT-ITM. The edges in the topologies were assigned using a random distribution with probability 0.1. We generated topologies consisting of 100 nodes. Then, we randomly assigned 100 receivers and a source to these nodes, bringing the total number of nodes to 201. Each topology was used for 10 runs, each time removing and re-assigning the receivers to internal nodes. We used a total of 10 topologies, which resulted in 100 simulation runs for each experiment. The results are next.

As with binary trees, we start with experiments measuring recovery latency. The first set of results is with loss at the source, and are shown in Figure 5.8. In the figure the term R100-100 means "random graph with 100 nodes and 100 receivers." From the figure we see that on the average, recovery takes about 30-40% of the unicast RTT, which is significantly better than with binary tree

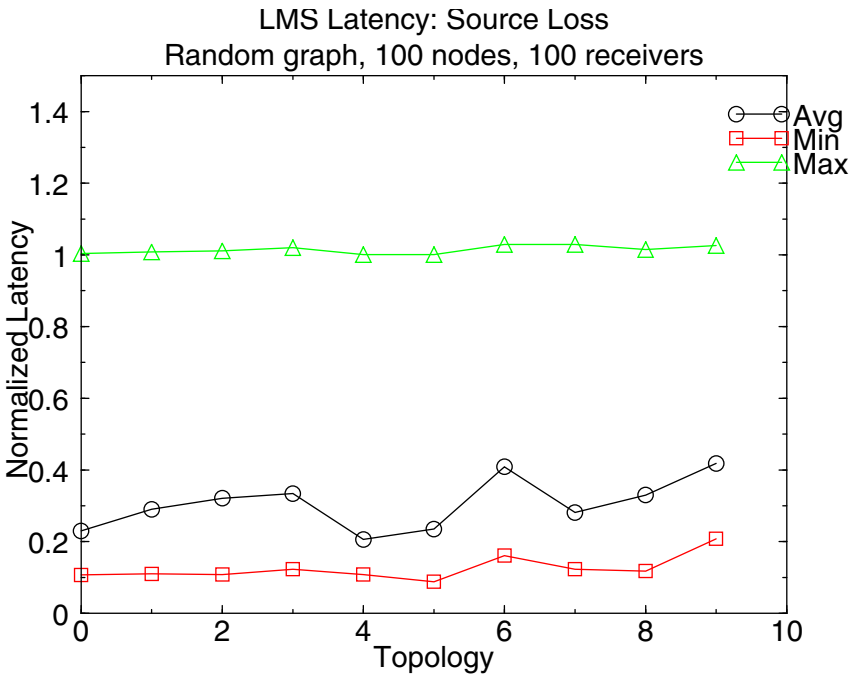

Figure 5.8: LMS Latency, Random topologies, loss at source

topologies. The reason is that in random topologies there are many helpers in the internal nodes, and thus recovery is initiated faster. The maximum latency is again around 1, and is experienced by receivers that are close to the source (note, however, that in absolute terms these receivers recover in less time than distant receivers). The average value is close to the minimum value, which implies that the latency distribution has a long tail.

Our next set of results, shown in Figure 5.9, shows recovery latency with loss at the receivers. Here we observe that the average latency has increased slightly to about 50%. The reason can be deduced by looking at maximum latency, which has increased slightly, to about 1.25. The increase is due to LMS selecting repliers that are located at larger distance that the source, as described in the previous chapter. With loss at all links, as shown in Figure 5.10, the results do not change significantly. The average latency again increases slightly to about 60%. Minimum and maximum latencies are essentially unchanged.

In Figure 5.11, we show exposure for different loss types. The results show that exposure is very low, under 2% in most cases, and well below 3%. For loss at the source, exposure is of course zero. The highest exposure occurs when loss is at the receivers, which is as expected. When loss occurs

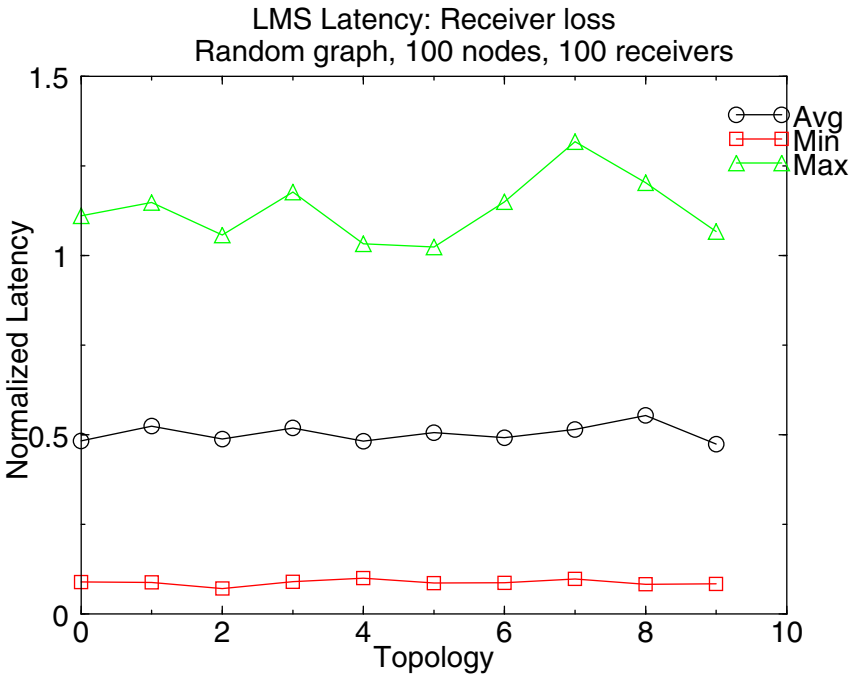

Figure 5.9: LMS latency, random topologies, loss at receivers

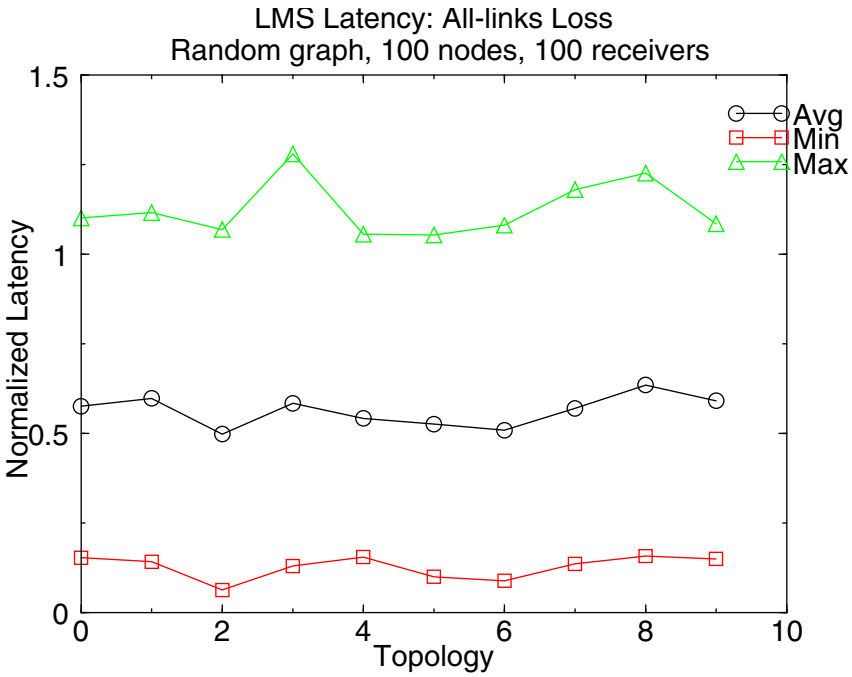

Figure 5.10: LMS latency, random topologies, loss at all links

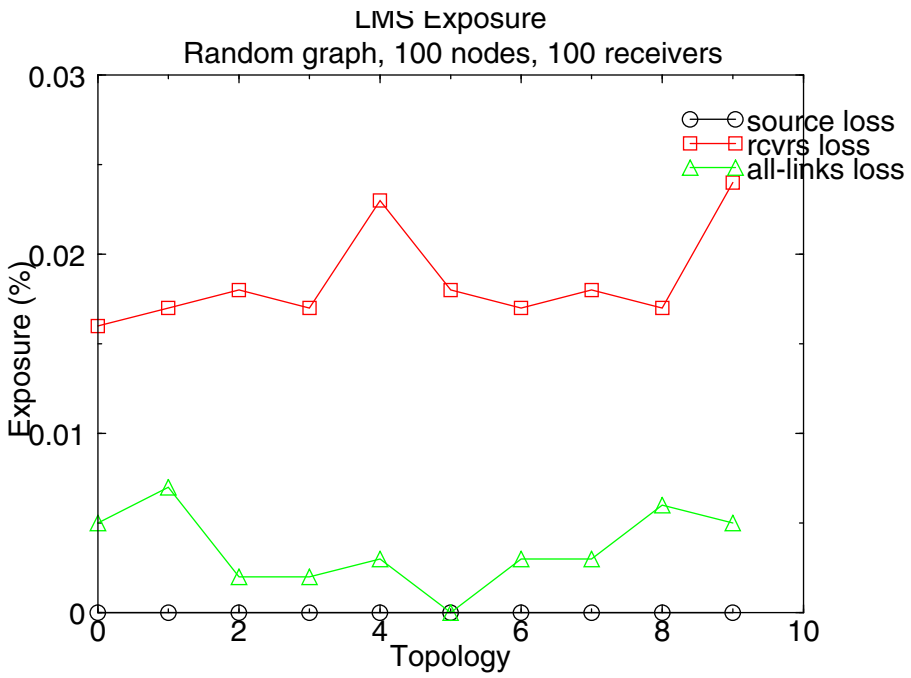

Figure 5.11: LMS exposure, random topologies

at a receiver acting as a replier, recovery causes duplicates to other receivers if they happen to lie downstream of the replier. When loss is equally distributed at the links, exposure remains very low, at under 1%. As with latency, exposure is much better with random graphs than with binary trees, again confirming our initial claim that binary trees are a difficult topology for LMS.

Varying the number of receivers

In the previous experiments, we have kept the number of receivers high to test the scalability of LMS, and have shown that LMS scales very well in large topologies and with large receiver populations. In this section, we present experiments that test the performance of LMS with small, sparse groups. We used the same topologies as in the previous experiments, altering only the number of receivers.

The results are shown in Figure 5.12, and Figure 5.13. We plot simulation results with 5, 20 and 100 receivers, the latter being taken from the previous experiments. We used loss at all links and plot only the average latency. From the results we notice that the recovery latency is inversely proportional to the number of receivers. By definition, recovery latency is 1 when there is only one

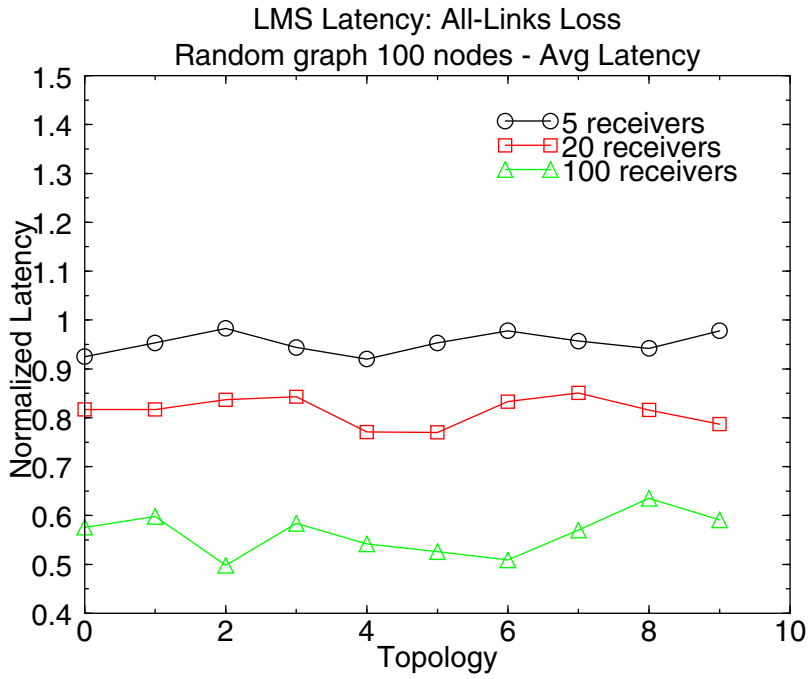

Figure 5.12: LMS latency, random graphs, different receiver population

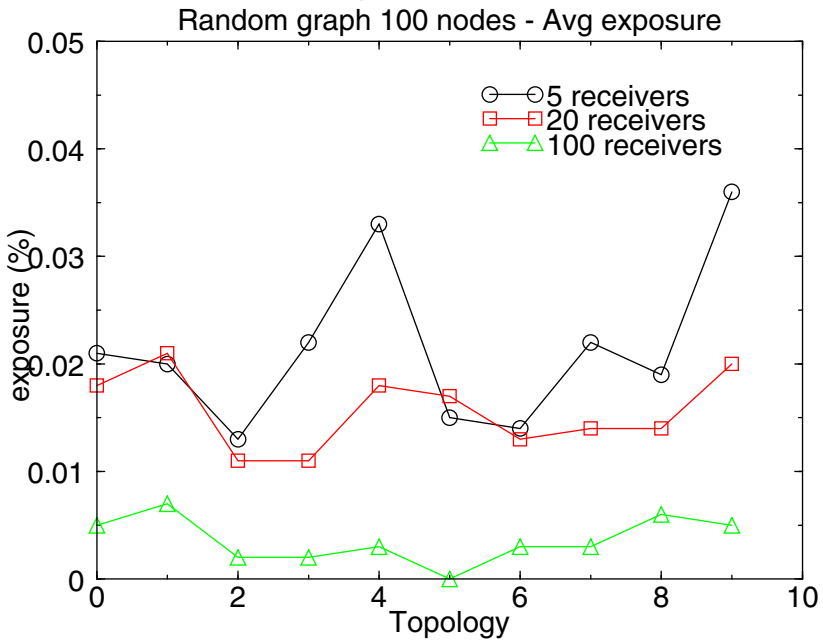

Figure 5.13: LMS exposure, random graphs, different receiver population

receiver, and gets progressively smaller as more receivers join the group. The same conclusion applies to exposure: larger receiver population leads to less exposure. Note, however, that even with few receivers exposure is still very low (less than 4%).

The fact that performance improves as the multicast group gets larger is good news. The reason performance improves with more receivers is that more helpers are available to initiate and send retransmissions, which improves latency; more helpers also means that there is a better chance to find a helper better located to serve retransmissions without causing exposure. The observation that performance improves with larger groups is an interesting result, which is actually not limited to LMS, but also applies to SRM and PGM, in varying degrees. For example, while in SRM latency improves as the number of receivers increases, in order to reduce duplicates from the higher receiver population the timer back-off values may have to be further increased, which can offset the latency gains. In PGM, the presence of more receivers will initiate retransmissions faster, but may worsen the repeated retransmissions problem. Recovery latency is the sum of the NACK propagation latency to the source plus the propagation latency of the retransmission to the receivers. The presence of more receivers may shorten the former, but not the latter. Also since PGM recovers from the source in most cases, the latency reduction will not be as great as with LMS.

5.8.3. Transit-Stub Topologies

In this section, we examine the performance of LMS with transit-stub topologies. While random topologies are a good approximation of dense, richly connected topologies, transit-stub topologies are a better approximation of the hierarchical structure of the Internet.

As with random topologies, we generated 10 topologies with 100 nodes each. The parameters fed to GT-ITM to generate the topologies are as follows: 1 top-level domain (the transit domain) with 4 transit nodes; each transit node had 3 transit-stub nodes; and each transit-stub node had 8 stub nodes. This brings the total number of nodes to $1 \times 4 \times (1 + 3 \times 8) = 100$ nodes. As with random topologies, we randomly assigned 100 receivers, but in this case the receivers were assigned to stub nodes only. Unlike random topologies, instead of simulating loss at the source and receivers, we opted to simulate loss at the different types of links. Thus, we studied loss at transit-transit, transit-stub and stub-stub links. The motivation was that we were interested in finding out how loss at

each type of link affects performance. However, as with random topologies, we also produced results with loss at all links.

The results are shown in Figures 5.14, 5.15, 5.16, 5.17, and 5.18. In all figures, ts100-100 means

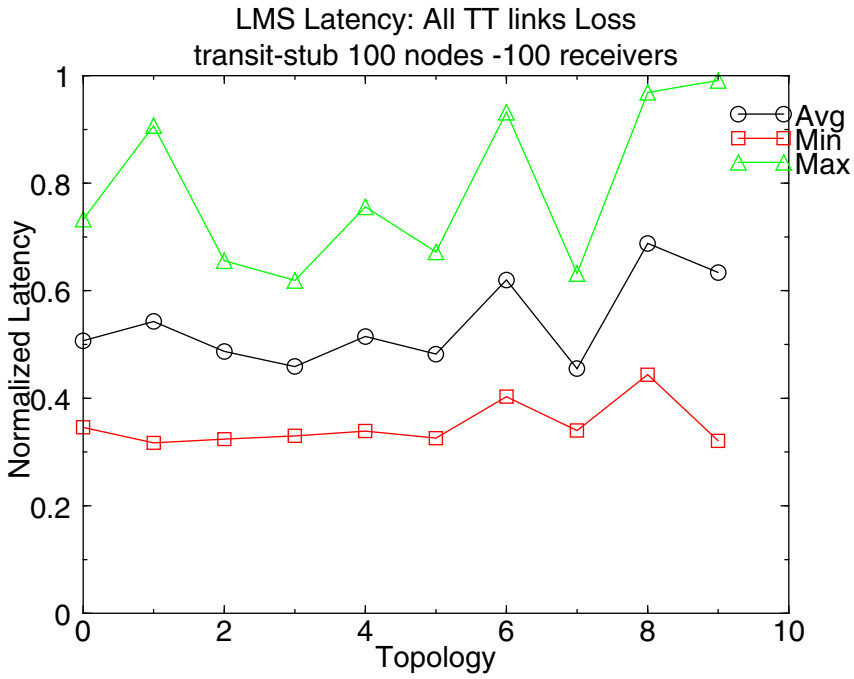

Figure 5.14: LMS latency, transit-stub topologies, loss at transit-transit links

a transit-stub graph with 100 nodes and 100 receivers. While latency remains at about 50% on the average, when all links are lossy, latency is a bit higher for loss at the higher levels (transit-transit links) and less so at the middle level (transit-stub links). However, the difference is small, and in general the results are on par with random topologies. The situation, however, is different with exposure. While exposure remains low with loss near the receivers (on stub-stub links), it increases significantly with loss at the higher levels (transit-transit and transit-stub), reaching peaks of 15-20%. While this is not alarmingly high, it seems to be highly dependent on topology and receiver allocation (for example topologies 0, 4, 7 and 9 have very low exposure, whereas topologies 5, 6, and 8 have somewhat higher exposure).

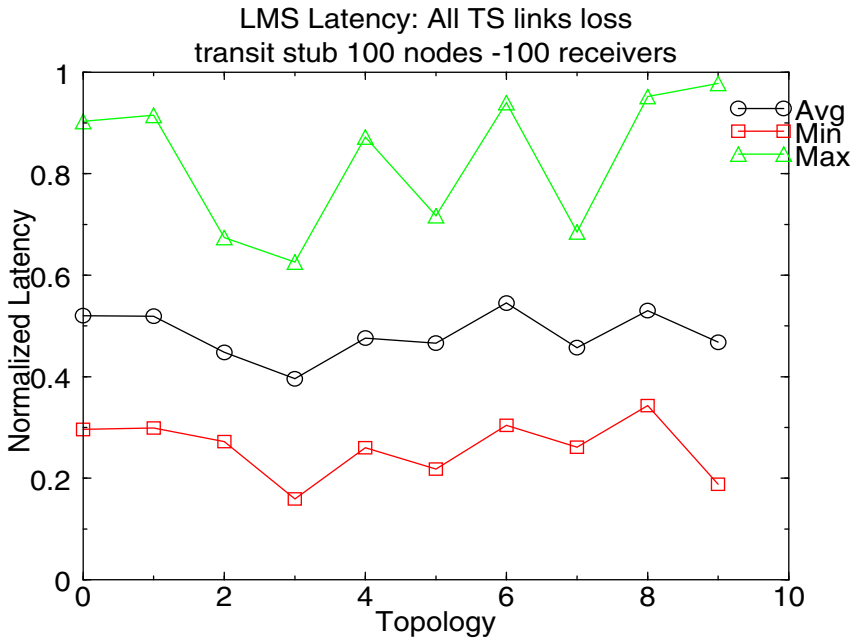

Figure 5.15: LMS latency, transit-stub topologies, loss at transit-stub links

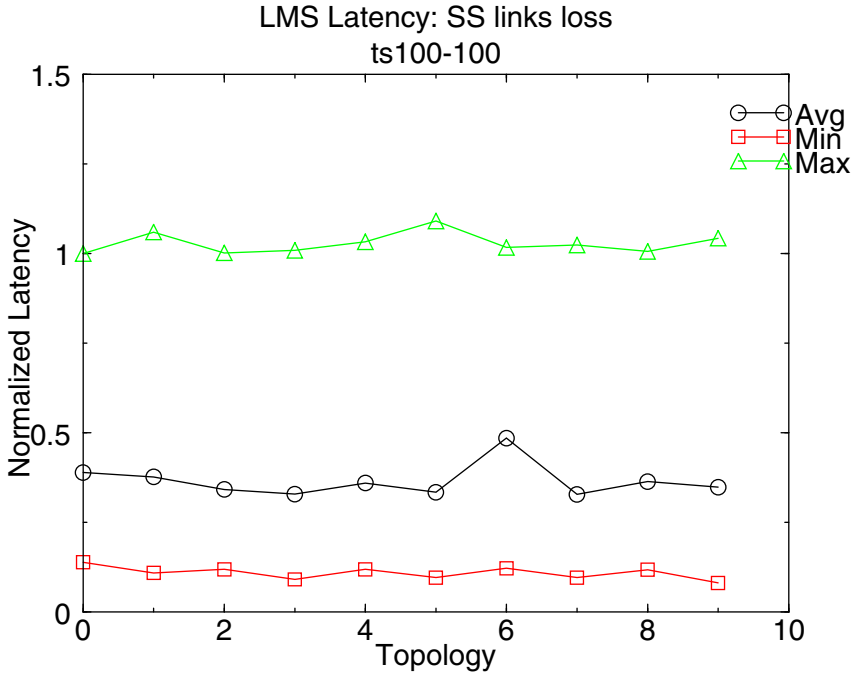

Figure 5.16: LMS latency, transit-stub topologies, loss at stub-stub links

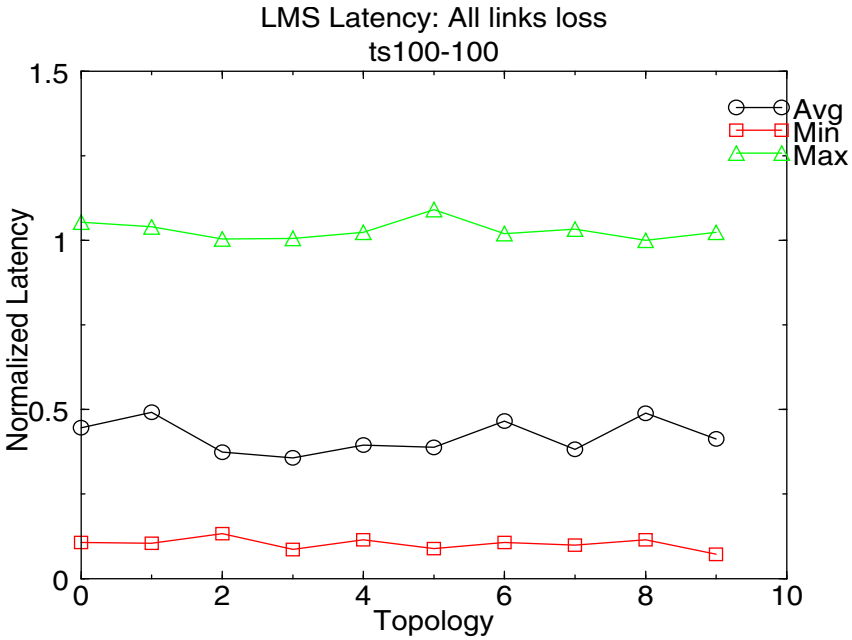

Figure 5.17: LMS latency, transit-stub topologies, loss at all links

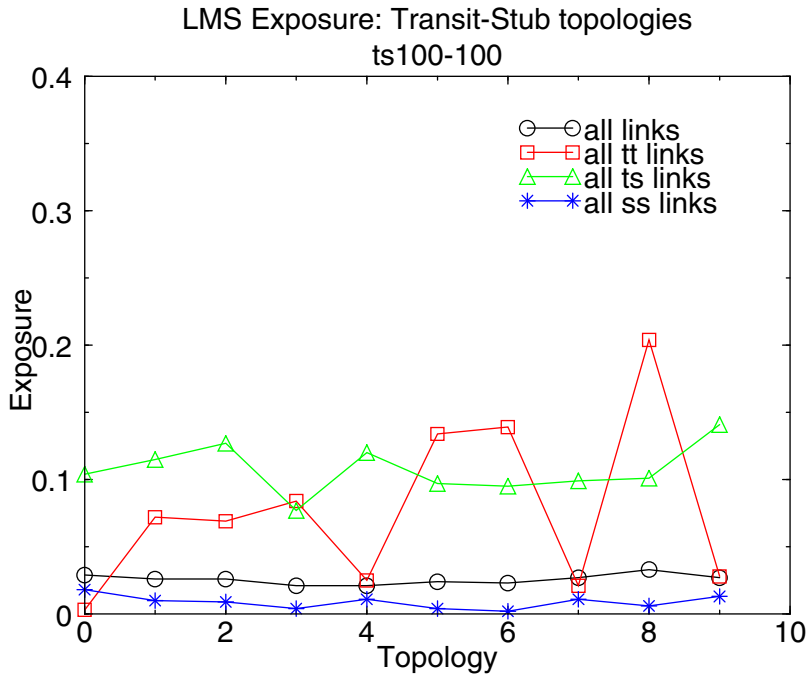

Figure 5.18: LMS exposure, transit-stub topologies

5.8.4. LMS Trade-offs

Having examined the performance of LMS via simulation, let us now summarize its trade-offs. The most important issue in LMS is the proper selection and maintenance of the replier state. Proper selection of repliers will minimize exposure and recovery latency. Frequent replier refresh will guard against replier failure.

Proper replier selection affects exposure because it controls duplicates. With optimal replier selection LMS can eliminate duplicates completely. However, making an optimal replier selection requires that we predict the location of loss, which we assumed we could not do. Therefore, LMS will always incur some amount of exposure as long as the location of loss is unpredictable.

Replier selection also affects latency. Some of our experiments revealed that recovery latency can exceed 1.0. This happens when LMS chooses a replier whose RTT from the turning point is higher that the RTT of the source from the turning point. In the previous chapter we described methods to deal with this scenario by considering repliers from upstream, but at the expense of risking implosion.

The problems above, as well as replier robustness, can be solved by frequent replier state refreshing. However, this may incur a large amount of control traffic, which aggregated over multiple sources and over multiple multicast groups may be unacceptable. We propose to allow receivers to trigger updates when conditions have changed sufficiently (e.g., exposure or latency have become excessive). Routers may simply employ a filter to allow a limited frequency of updates, to guard themselves from malfunctioning receivers.

5.9. SRM Experiments

SRM has already been extensively studied via simulation and results have been reported elsewhere [31]. Our goal here was not to repeat or extend already published results, but to run simulations of SRM on the same topologies used for LMS, thus eliminating one important variable in comparing the two schemes. However, we were not completely successful in achieve that goal. While we used the same topologies for LMS and SRM, we could not run SRM simulations with more than 20 receivers in the 100-node topologies. Attempting to use more receivers resulted in extremely long simulation runs and very high memory consumption. We ended up running the SRM

simulations on an UltraSparc with dual processors and over 1GB of RAM, which still took about 3 days to finish each set of 100 runs. The reason the SRM simulations are so slow is that the SRM implementation in *ns* is done mostly in Tcl. Clearly the goal is to maximize flexibility; unfortunately this comes at the price of speed and memory consumption.

Since our goal was to use the *ns* implementation of SRM "as is" and not to study SRM in depth, we decided not to attempt to improve the simulation running time (which would require re-writing the simulation in C++). The results we report are consistent with results presented previously, and thus we do not feel we would have contributed further to the understanding of SRM had we simulated larger topologies.

In the following sections we present results from simulating SRM in random topologies only. As mentioned before, further study of SRM is beyond the scope of this work. The random topologies are the same used with LMS, but here we use only 20 receivers for SRM, to keep memory usage and simulation running time manageable. We report results for normalized latency and the number or requests and replies generated for each lost packet. Recall that without local recovery, these messages will reach every member of the multicast group.

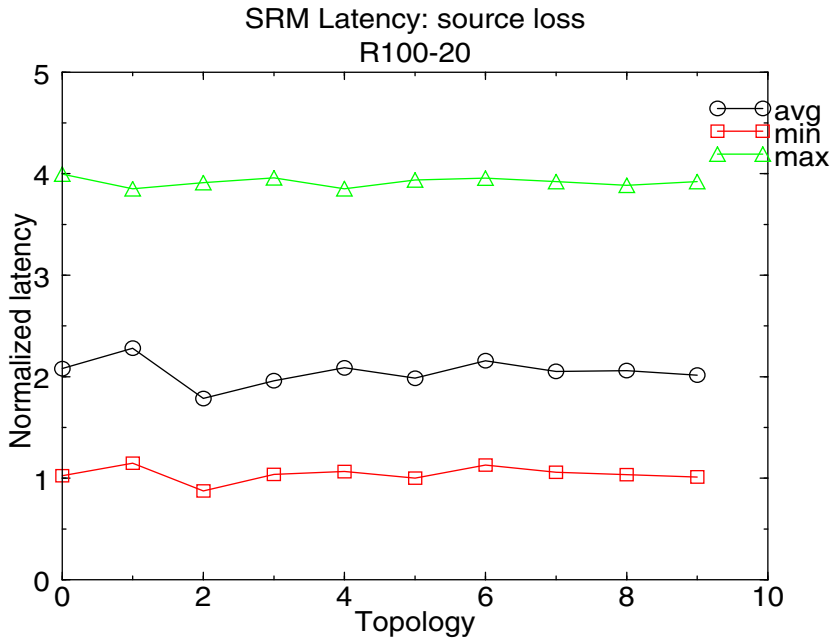

Figure 5.19: SRM recovery latency, random topologies, loss at the source

Figures 5.19, 5.20, and 5.21, show the recovery latency for loss at the source, receivers and links

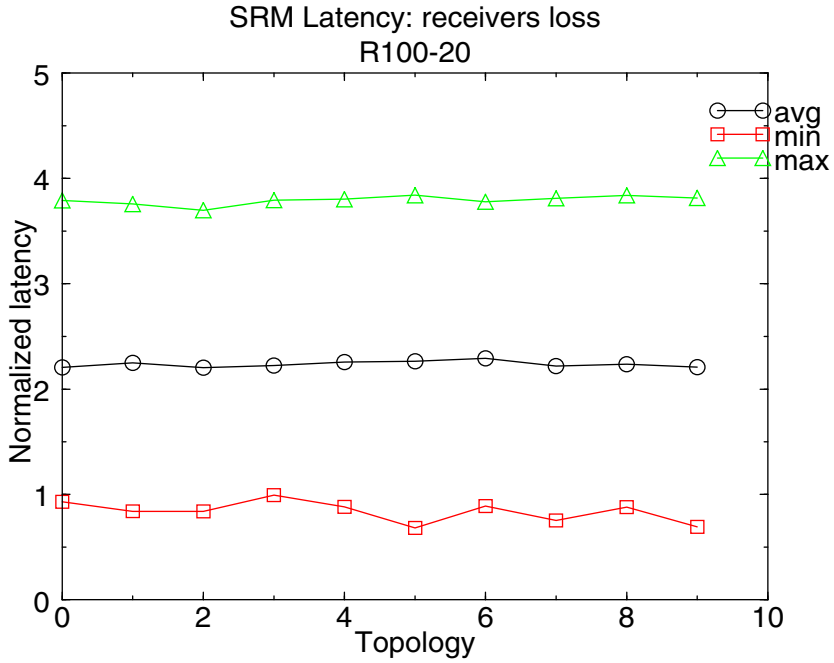

Figure 5.20: SRM recovery latency, random topologies, loss at all receivers

respectively. We note that on the average, SRM recovers from a loss in about 2 RTTs, or twice the unicast latency, with the maximum value being around 4 and the minimum around 1. We also note that the recovery latency is relatively uniform over all topologies. We believe that the reason is that the back-off timers absorb any differences that may arise due to a particular topology. In addition to being relatively insensitive to topology variations, SRM appears to also be insensitive to loss location. Thus, there are very small differences between results from loss at the source, receivers or links.

Figures 5.22, and 5.23, show the number of requests and replies generated on the average for each loss. For requests, the linear component in the back-off timers works well and keeps the number of requests low (a little above 1 for loss on all links). The number of requests are a bit higher when loss is at the source because all receivers compete in sending a request. The results, however, are significantly worse for replies where SRM may generate 4 -5 replies for each loss. For replies

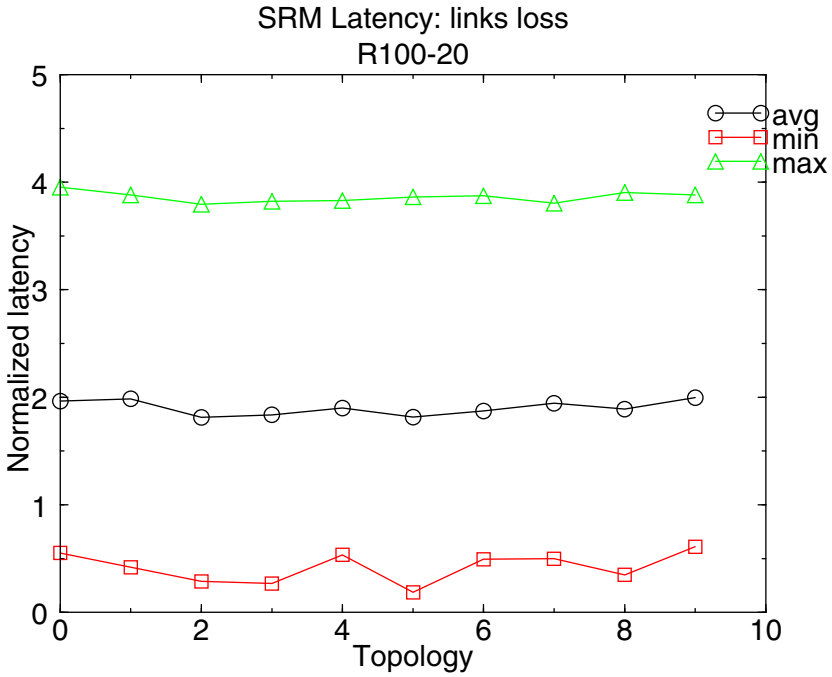

Figure 5.21: SRM recovery latency, random topologies, loss at all links

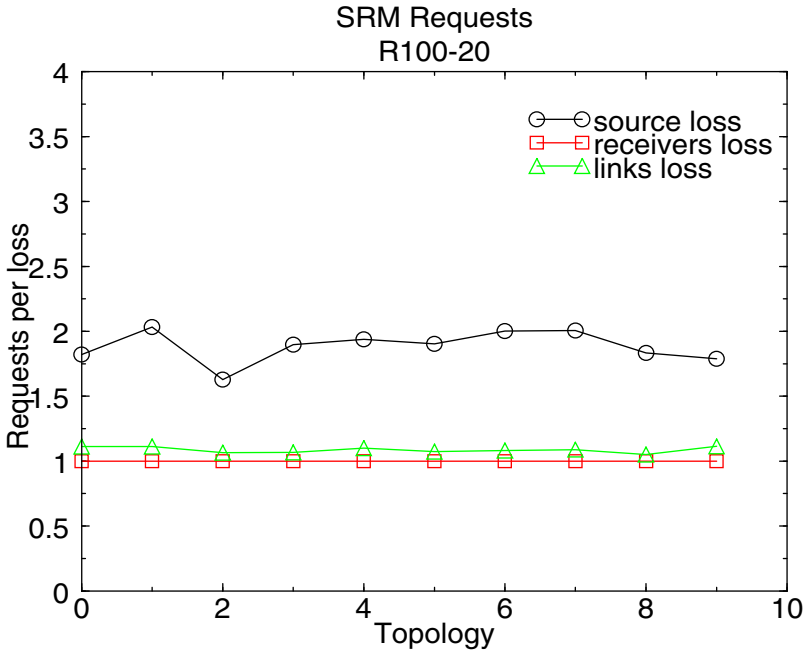

Figure 5.22: SRM requests, random topologies

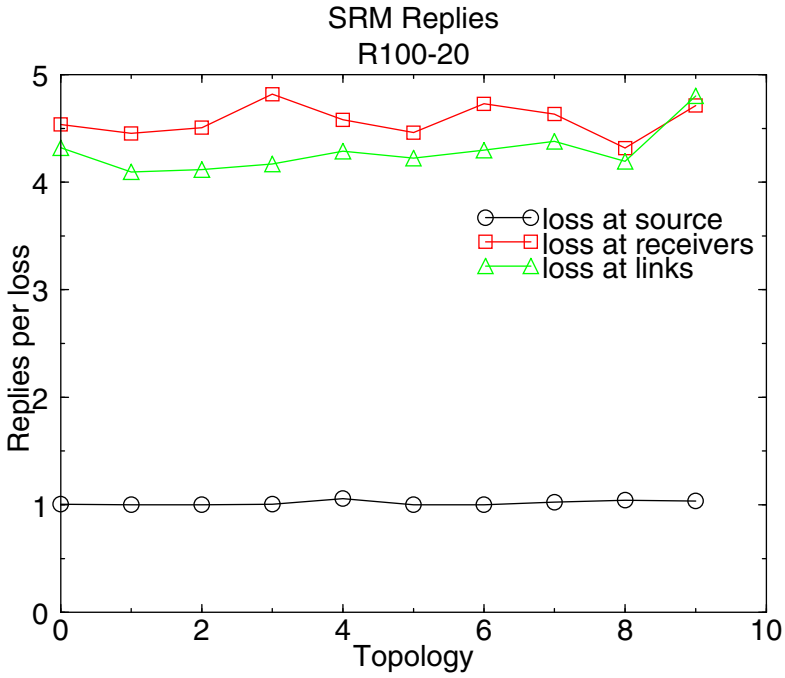

Figure 5.23: SRM replies, random topologies

the linear component is less effective since the inter-receiver RTT is smaller in general than each receiver's RTT to the source.

5.9.1. SRM with Adaptive Timers

As described in Chapter 3, for loss patterns that do not change frequently over time, SRM can make use of an algorithm to adapt the back-off timers and improve recovery latency while minimizing duplicate control messages. The algorithm was proposed by the SRM designers and aims to optimize the back-off timers based on the underlying topology. The SRM implementation in *ns* includes adaptive SRM, and thus we carried out some experiments with adaptive timers. The results are shown in Figures 5.24, and 5.25.

From the results we can see that adaptive timers have significantly reduced the number of duplicate replies in SRM. Without adaptive timers, there would be over 4 replies per loss, but with adaptive timers that number is reduced to less than 2. We do not see a significant improvement in the number of duplicate requests, but the number of such duplicates was low to begin with and would be hard to improve on it.

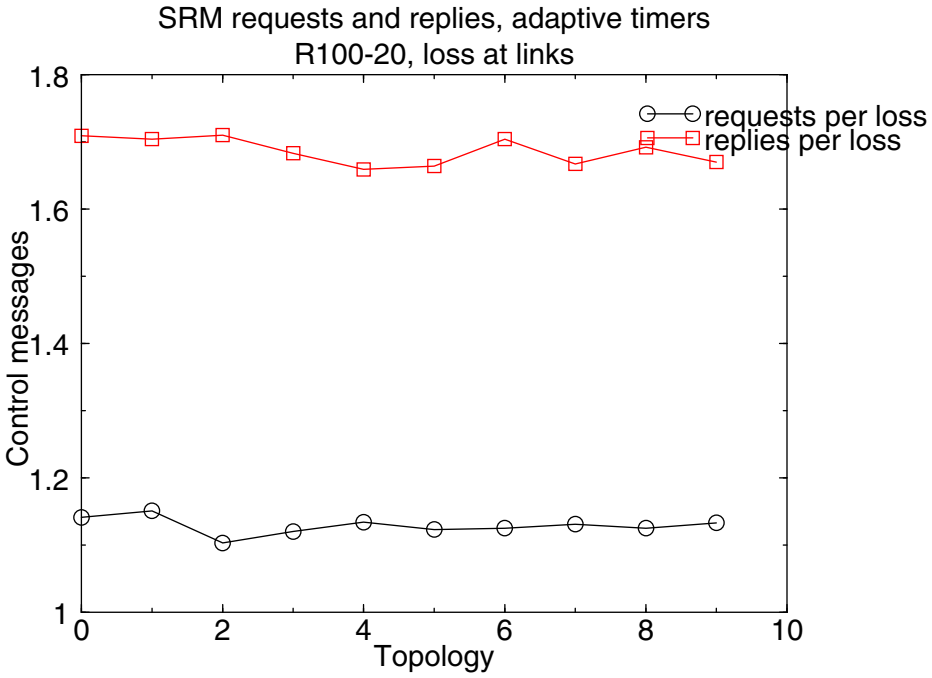

Figure 5.24: SRM requests and replies, adaptive timers

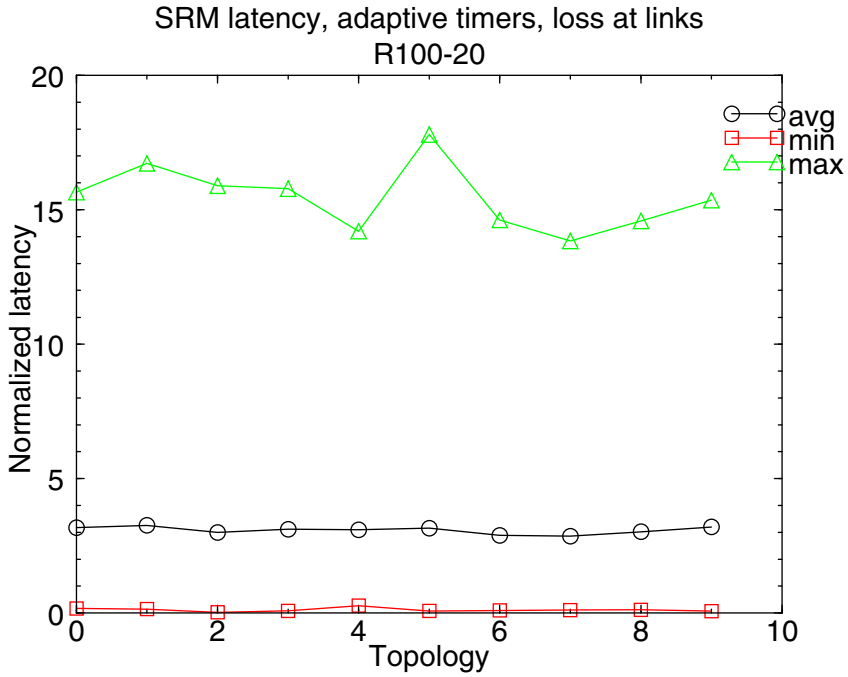

Figure 5.25: SRM Latency, adaptive timers

The penalty paid for reduced number of requests and replies can be seen in the latency results. The average recovery latency has gone up from about 2 without adaptive timers to about 3. The reason can be seen by looking at the maximum latency, which hovers around 15. We believe that the reason for these extremely high latencies, is the high timer values the algorithm assigned to some receivers when loss was at a particular location. When the loss moved to a different link, these receivers were caught with highly inappropriate timer values.

5.9.2. SRM Trade-offs

The most significant trade-off in SRM is trading latency for implosion. The larger the back-off timers, the better the probability requests and replies will be suppressed. However, given that loss may randomly move around, it is very hard to determine the optimal value of the adaptive timers do that loss is always minimal. In addition, setting up the RTT state to each member takes time, and in dynamic groups this state may never converge to a stable state.

5.10. PGM Experiments

To the best of our knowledge, our simulation was the first simulation ever written for PGM. Results from our simulation were first presented at the 4th Reliable Multicast Research Group meeting [56], where one of the PGM designers from Cisco was present. There, we were informed that PGM was already at an advanced stage of implementation.

The topologies and parameters used for the PGM experiments are the same as with LMS and SRM, except that we did not have the memory limitations we experienced with SRM (the PGM simulation was written mostly in C++) and thus our simulations used 100 receivers, as in LMS. Our PGM simulation did not include all the features described in the PGM specification. However, we believe our simulation included enough of the PGM functionality to capture the basic operation of PGM. Thus, we believe that the results we present here are applicable to a full-fledged PGM implementation.

Figures 5.26, 5.27, and 5.28, show the PGM latency with loss at the source, receivers, and all links respectively. With loss at the source, average recovery takes about 80% of the unicast latency. One might wonder why this experiment did not produce results similar to LMS, since in both cases recovery is done from the source; the reason is due to PGM repeated retransmissions, which result

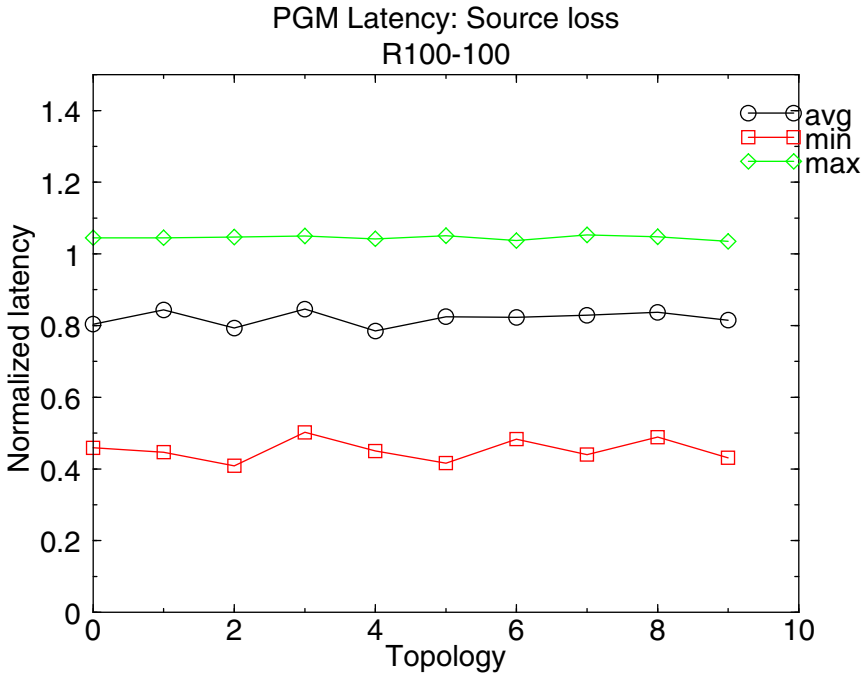

Figure 5.26: PGM Latency, loss at the source

in an increase of the average recovery latency. In addition, recall that in PGM receivers observe a random back-off before sending NACKs, which results in an increase in the overall recovery latency. We have used a back-off value of zero in our experiments; with larger back-off values, recovery latency will increase even more.

With loss at the receivers, the recovery latency is very close to 1, as expected. Note that with PGM recovery latency does not increase much beyond 1 because a retransmission always comes from the source. Thus, unlike LMS, PGM does not allow a receiver to send a retransmission which may result in a longer retransmission path. With loss at all links, recovery latency in PGM increases slightly (a trend also seen with LMS), to about 90% of unicast latency. This is about 30% more than what is seen with LMS. In summary, it appears that on average, allowing receivers to participate in recovery saves about 30 - 50% in recovery latency.

Figure 5.29 shows the PGM repeated retransmissions when loss occurs near the source. Recall that in PGM the source may send repeated retransmissions in response to the same packet loss if the RTT between receivers is such that a retransmission arrives before NACKs from downstream

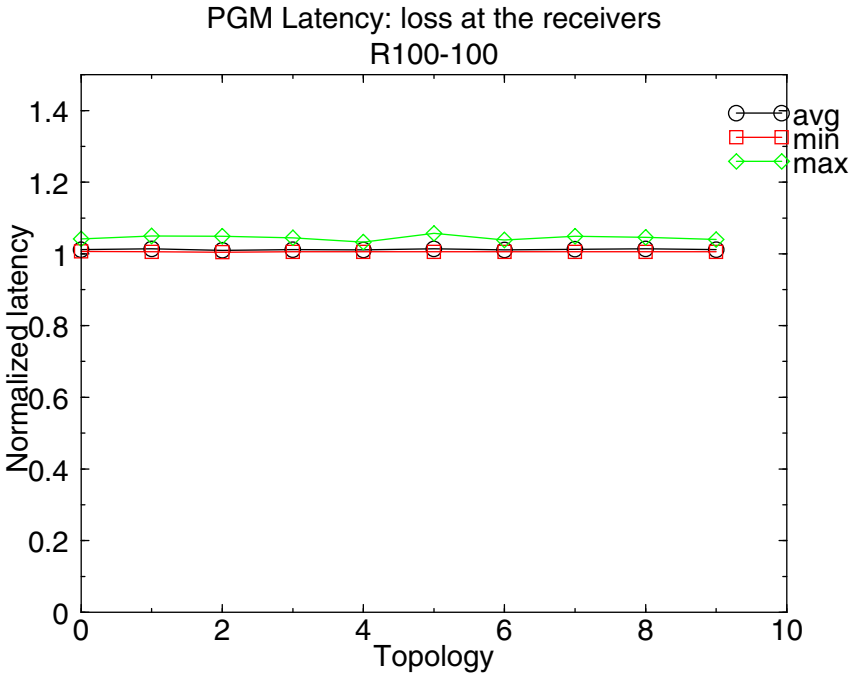

Figure 5.27: PGM recovery latency with loss at each receiver

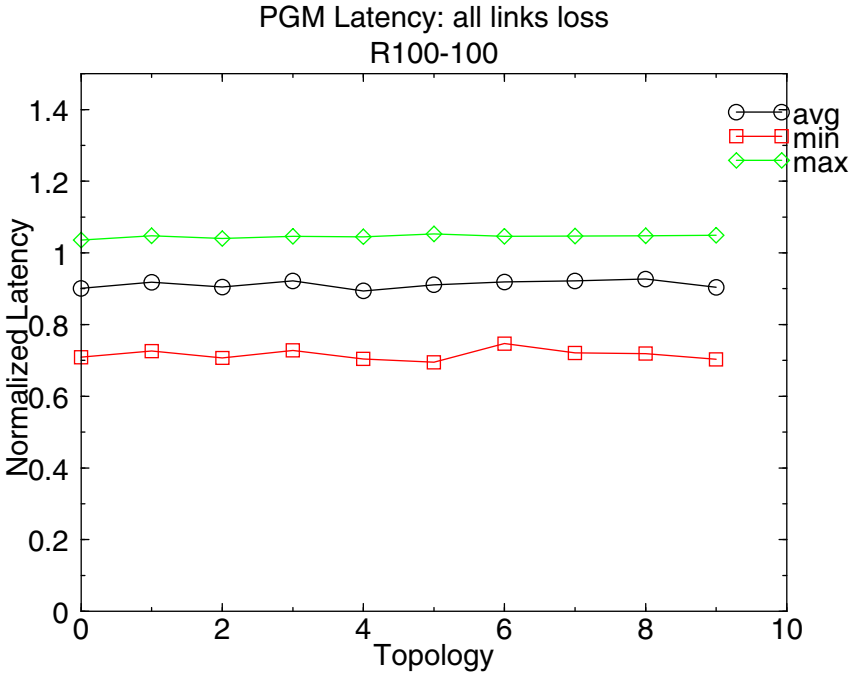

Figure 5.28: PGM recovery latency with loss at all links

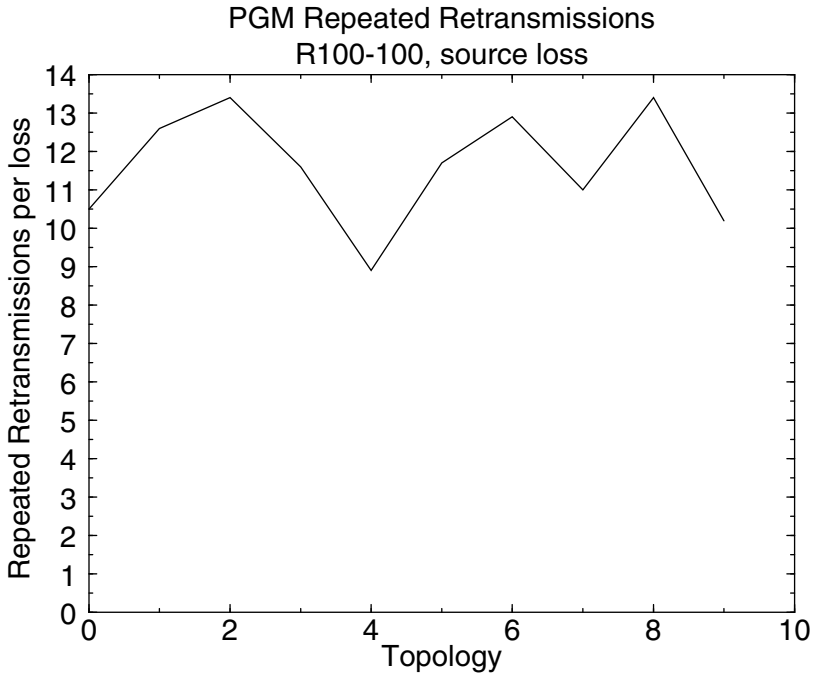

Figure 5.29: PGM repeated retransmissions with loss at the source

receivers at a particular router, wiping out the current retransmission state at the router. Then, when other NACKs arrive, they recreate the state all the way back to the source and trigger another retransmission. Loss that occurs near the source will create the maximum number of repeated retransmissions. Our results show that the number of repeated retransmissions can be quite high, between 9 and 13 retransmissions for each lost packet. This will invariably create a significant load at the source and may contribute to congestion.

This experiment did not include any back-off at the source, because it was not part of the initial PGM specification. Since then, the PGM designers have added a heuristic to reduce this problem, where the source delays the sending of a retransmission to allow NACKs to establish state at the routers. The difficulty in designing such a heuristic is how to determine the appropriate amount of back-off to minimize repeated retransmissions while not unduly increasing recovery latency.

Figure 5.30, shows the repeated retransmissions problem with loss distributed at all links. It is clear that the problem becomes much less pronounced in this experiment. It appears that the number of repeated retransmissions reduces by two orders of magnitude. Thus is hard to estimate the real-life effect of repeated retransmissions in PGM until we have a reasonably good loss model. What

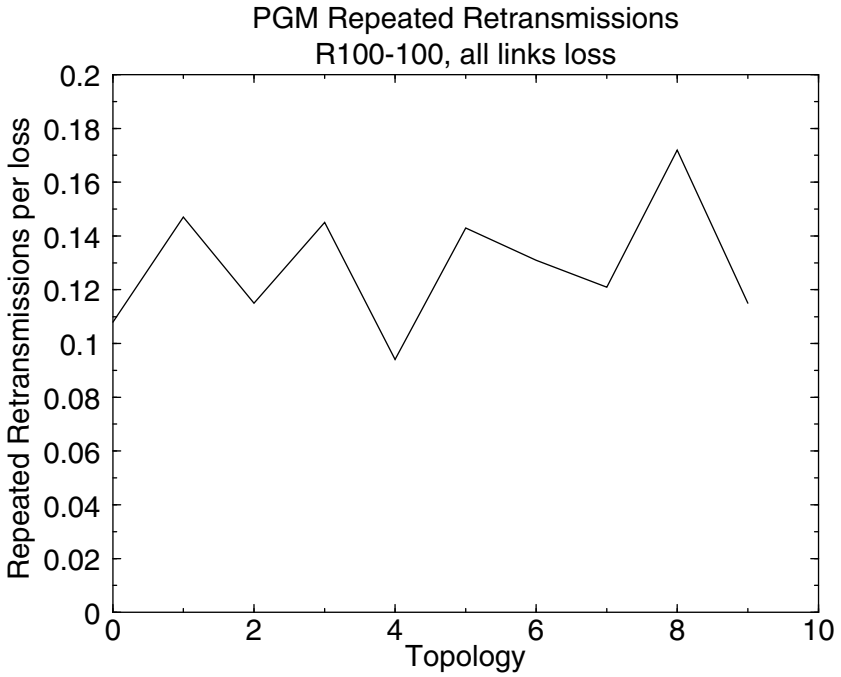

Figure 5.30: PGM repeated retransmissions with loss at all links

this experiment clearly demonstrates again is the sensitivity of multicast protocols to topology and loss location.

We do not present experiments measuring repeated retransmissions with loss at the receivers, because, clearly, there are no repeated retransmissions in this case.

5.10.1. PGM Trade-offs

PGM trades state at the routers for eliminating exposure and keeping recovery latency low. The state at the routers is proportional to the number of lost packets, and thus may increase significantly during periods of high loss. In addition, PGM needs to trade latency for eliminating repeated retransmissions, in cases where loss occurs near the source.

5.11. Summary and Discussion

In this chapter we presented simulation results measuring the performance of LMS, SRM and PGM. We used the *ns* simulator, which comes with SRM, and added implementations for LMS and PGM. The experiments included topologies generated via GT-ITM, a generator that generates

topologies that approximate Internet topologies. We used the same topologies to evaluate all three schemes. We measured recovery latency for all three schemes, the exposure and number of duplicates for LMS and SRM, and number of repeated retransmissions for PGM.

In general, LMS and PGM perform much better than SRM, due to the assistance they receive from the routers. Improvements can be seen in both recovery latency, exposure and duplicates. We believe that these performance advantages alone, are important enough to warrant serious consideration of their deployment; for example, their lower latency may allow the implementation of multicast error control for continuous media applications; and their limited or absent problems with exposure, are important assets for scalability. These two schemes have the further advantage of freeing receivers from maintaining any topology related state and performing any topology related operations. This is an important advantage in terms of reducing application complexity, and will greatly ease the deployment of multicast applications.

Comparing LMS and PGM, we note that while LMS is much simpler to implement and requires significantly less overhead than PGM, its performance is far from lacking. LMS has significantly lower recovery latency, while trading very little in terms of exposure. PGM incurs per-lost-packet penalties at the routers, which for large routers (like those on the backbone) it can be significant. In contrast, LMS incurs only a small fixed state penalty per multicast group at the routers. Thus we believe that in terms of scalability, LMS has the edge.

The performance results are summarized in Table 5.1. Our results show that LMS has the fastest

Table 5.1: Performance summary of all three schemes

SCHEME	Normalized Latency (% of unicast latency)	Exposure/duplicates	Repeated retransmissions
LMS	30 - 60%	0.5%	none
SRM	>200%	4-6 duplicates per loss	none
PGM	80 - 100%	none	up to 10 - 13

recovery: a receiver typically recovers from loss in about 30-60% of its unicast latency to the source.

PGM, which always recovers from the source is next, requiring about 80-90% of the unicast latency. Thus it seams that the penalty PGM pays in terms of latency by not allowing local recovery is about 30-50%. SRM, due to its duplicate suppression mechanism, requires the longest time to recover, which is at least twice the unicast latency.

In terms of duplicates, SRM generates about 4-5 duplicate packets per loss. Without some form of local recovery, these duplicates will be sent to all receivers. LMS and PGM only allow one retransmission to reach a particular receiver. However, whereas PGM will never allow a retransmission to reach a receiver that has not sent a NACK, LMS may allow unwanted retransmissions to reach receivers that do not need them. Our simulations show that the probability of a receiver receiving unwanted packets is not very high, even with static repliers: typically, under 0.5% of all generated retransmissions will reach receivers that did not need them. PGM, while preventing unwanted packets at the receivers, may force the source to send repeated retransmissions in response to a single packet loss. The number of these retransmissions can be high enough (about 10 - 13 per loss) to be of great concern; however, without a better understanding of loss characteristics on the MBONE, it is hard to quantify the overhead of repeated retransmissions. What is clear though, is that PGM will have to incorporate some delay before sending retransmissions, which will increase its recovery latency beyond what we have reported here.

Chapter 6

LMS IMPLEMENTATION

In the previous chapter we used simulation to evaluate the performance of LMS in large, simulated topologies. The simulations showed that LMS performs very well in terms of limiting implosion and exposure, and maintains low recovery latency. In this chapter we complement our simulation by describing the implementation and evaluation of LMS in the kernel of a real system, namely NetBSD Unix.

As we described in the previous chapter, large scale deployment and evaluation of LMS is a task of enormous proportions, one which we are not equipped to perform. Such a task requires access to a large, multi-node network, preferably under our complete control, which is very hard to achieve. Our resources only allowed us to deploy LMS over just four nodes. However, despite our limited setup, we believe that the implementation we present in this chapter contributes some important insights. Even though our implementation testbed is minimal, our work constitutes a significant portion of essential preliminary work towards a future wide deployment of LMS. In addition, our work is invaluable to anyone wishing to understand how to implement LMS in other platforms, and thus can help in promoting both a better understanding of LMS, and its dissemination. It is our hope that we can migrate our implementation to a larger size testbed, perhaps utilizing one of the few experimental networks in existence today.

In this chapter we present a software implementation of the LMS forwarding services, which is the major new component in LMS, and thus, the most interesting and important to understand. We

assume the following environment: a multicast group has been created and all its members have joined; the replier state has been established; and a transport protocol is in place that detects losses, and uses LMS to send retransmission requests and retransmissions in the form of directed multicasts (dmcasts). We will refer to this protocol as *LTP*; we will not implement this transport protocol again, since we have already done so in the previous chapter, in our simulations. We will simply focus on implementing the LMS forwarding services at the routers and the LMS support services at the endnodes that allow applications to use the LMS forwarding services.

Our main objective in evaluating the forwarding services of LMS using implementation is to study the feasibility of integrating LMS into the existing environment. Other objectives include capturing the effort that must be expended in integrating LMS into the existing, highly optimized and possibly delicately balanced system, namely the networking code, as well as any performance penalties. Specifically, the objectives of our implementation as presented in this chapter, are the following:

- Determine if LMS can be implemented. This objective is very important. It essentially seeks to determine whether LMS can be incorporated into the existing router architecture, or if there is a fundamental incompatibility between LMS and the existing environment.

- Determine how much effort it would take to implement LMS. This objective is to quantify the complexity of implementing LMS in terms of programming effort.

- Determine how much change it would require to the existing architecture, if any. This objective is to determine how much impact (if any) LMS has on the existing architecture. We are especially interested in determining what kind of support LMS needs, and whether implementing LMS will require major changes in the existing architecture.

- Quantify the overhead of LMS compared to normal packet forwarding. this objective is to evaluate the performance of LMS in terms of state and processing cycles.

From an implementation point of view, we can divide LMS into the following components:

- **LMS-FWD**: Forwarding component: this LMS component resides at the routers and implements the special forwarding of LMS packets. This component is needed only at routers and resides entirely in the kernel, specifically at the IP layer.

- **LMS-API**: Application API: this component is required at the endnodes to enable applications to use LMS (i.e., send and receive LMS packets). This component implements any changes needed to the existing application and protocol interface. Fortunately, LMS did not require any changes to the socket interface, only small changes at the UDP level.

- **LMS-ENCAP**: LMS encapsulation: this is the component required to encapsulate multicast packets in unicast packets in order to send directed multicasts (dmcasts). Its functionality is very similar to the IPIP encapsulation protocol used in tunnels, but generalized to work without the existence of tunnels; in other words, sending LMS encapsulated packets does not require the existence of a tunnel, and packets can be sent to any unicast address, not just the tunnel peer.

- **LMS-HIER**: Replier hierarchy component: this is the component that allows endpoints and routers to communicate in order to maintain the replier hierarchy. It includes some modifications to the join procedure, and inter-router message exchange protocol or mechanism to maintain the replier hierarchy. This component is shared among routers and endpoints and resides in the kernel.

Figure 6.1 shows the LMS components and their relation with each-other along with the relevant applications and protocols. On the left we show the LMS components at an endnode. The boxes labeled “application” and “LTP”, are the multicast application that requires reliable multicast, and the transport protocol, i.e., the module that uses LMS to provides the error control. Note that we do not consider LTP to be part of LMS; it is simply a user of LMS services. For the remaining of our discussion we assume that LTP is implemented in user space, either as part of the application or as a separate library. However, there is no reason why LTP cannot be implemented in the kernel, as another protocol in the protocol stack next to TCP and UDP.

LTP communicates with LMS via the LMS-API module by exchanging control information, such as the turning point location, and data information, such as requests and retransmissions. The

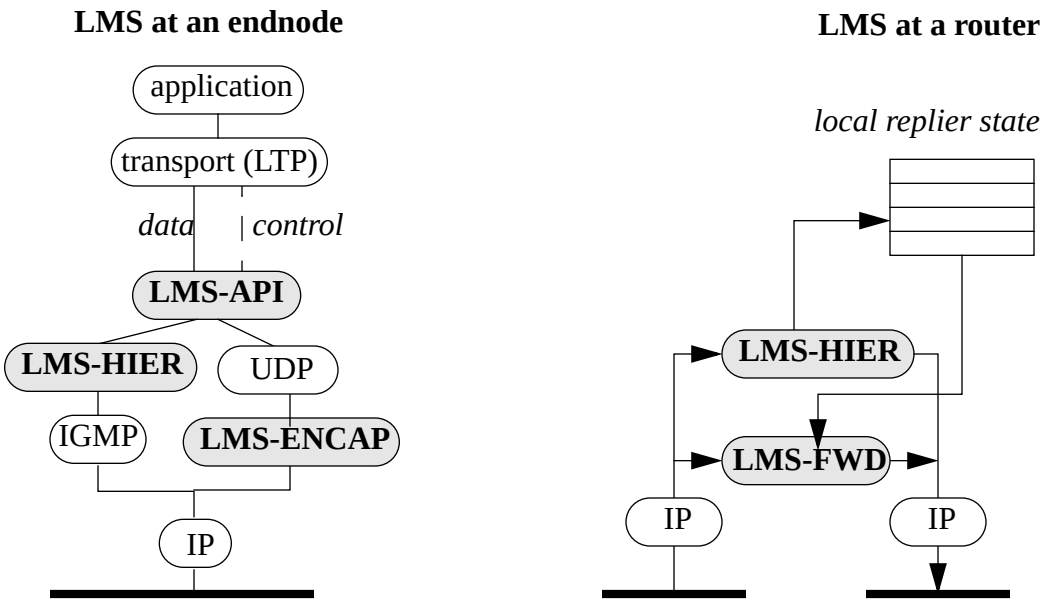

Figure 6.1: LMS components

task of the LMS-API module is as follows: on the sending side, it formats control data as an IP option and passes it to UDP/IP; on the receiving side, it extracts control data from the IP option attached to the LMS packet, and passes it to LTP. When LTP sends a dmcast, UDP passes the data and the control information to LMS-ENCAP, which prepares the encapsulated packet and passes it to IP for transmission.

LTP also uses the LMS-API module to communicate with the LMS-HIER module to update its replier status. The LMS-HIER module in turn uses IGMP to send these updates to the router.

The modules at a LMS router are shown on the right side of Figure 6.1. The LMS-HIER component receives updates either from endnodes or other routers and updates the local replier hierarchy state at the router. It then decides if the updates must be propagated to other routers, in which case it sends appropriate messages. Finally, the LMS-FWD module implements the heart of LMS, i.e., the forwarding services. This module receives LMS packets, consults the local replier state and forwards the packets accordingly after possibly adding the turning point information.

In this chapter, we present the implementation of three of the aforementioned LMS components, namely LMS-FWD and LMS-API, and LMS-ENCAP. We did not implement the fourth component, LMS-HIER, because it is not yet clear how much of it is needed and what its precise functionality should be. We discuss the issues around the implementation of LMS-HIER and sketch an implementation of this module at the end of this chapter.

In the remaining of this chapter we present our software implementation of LMS. We have modified the NetBSD Unix kernel to support handling of requests and directed multicasts. The kernel modifications include the addition of two new IP multicast options. The modified kernel components are: UDP output processing, IP and UDP input processing, and kernel multicast forwarding. The implementation of LMS presented no major problems and demonstrated that LMS can indeed be implemented. The implementation took about 250 lines of new C-code, which is less than the existing forwarding code; moreover, it took less that 2 weeks to write, most of that time spent on understanding the existing code. Our implementation presented no major disruptions to the existing networking code; however, it did point out some minor changes in handling IPIP encapsulated packets that may be needed to accommodate LMS. LMS processing did not affect the existing multicast forwarding, and the processing overhead of the added code is minimal.

This chapter is organized as follows: we first provide some background of the existing architecture of NetBSD, which will be useful in understanding our implementation. We then describe in detail the modifications we made to NetBSD to implement LMS.Then we present our experimental setup and the experiments we used to evaluate LMS. We discuss the issues in the module we did not implement, namely LMS_HIER and sketch a future implementation of this module. Finally, we conclude with a summary of the chapter.

6.1. Background

LMS requires operating system support in two areas: (a) in exchanging control information between the LTP and UDP/IP, and (b) special LMS forwarding support from hosts acting as a multicast routers. In this section we introduce the components and mechanisms available in NetBSD to support these operations. These are, (a) the system calls provided by NetBSD that allow control information exchange between LTP and the protocol stack (called *ancillary data* in NetBSD terminology); and (b) the NetBSD multicast forwarding architecture, upon which LMS builds. In the next

section we will describe the modifications we made to these components to support LMS. As we will see, these modifications are minimal and we were able to reuse a substantial part of the existing code.

6.1.1. Control Information Exchange: Ancillary Data

As we have seen before, LTP must convey some control information to the protocol stack in order to send a request or a directed multicast (dmcast); in turn, the protocol stack must relay some information back to LTP when a request or a retransmission is received. For example, the control information provided by LTP when sending a request includes the original sender's address, in order to identify the correct multicast tree; for a dmcast, LTP must provide control information which includes the address of the turning point router and the link id. Conversely, when receiving a request, the turning point information carried in the request must be passed to LTP as control information. It is important to note that since each message may carry control information, control exchange between the kernel and LTP must take place at a message level granularity.

The NetBSD socket interface provides two ways of exchanging information between user space (where we assume LTP resides) and the protocol stack. The first is via the system calls setsockopt and getsockopt; the second is via the system calls sendmsg and recvmsg. We describe both next, in order to determine which pair is best suited for LMS.

The set/getsockopt pair is used to set/get socket options that typically remain in effect for the lifetime of the socket. Examples include multicast group membership for the socket, the size of the socket buffer, the time-to-live (TTL) value for outgoing multicast packets, etc. Thus, the set/getsockopt pair is more appropriate for manipulating values that are long-lived rather than per-message operations, which is what LMS requires. If we wanted to use these calls LTP must follow every read or precede every write with the appropriate sockopt system call to exchange control information. However, not only is this method very cumbersome, it is also expensive since it requires twice the number of system calls. We thus conclude that the set/getsockopt pair is inappropriate for LTP/LMS.

Fortunately, as we will see, the send/recvmsg system call pair is perfectly suited for the requirements of LTP/LMS. These system calls essentially behave like normal read and write calls; however, in addition to data, these system calls accept additional control parameters called *ancillary*

data. These parameters are carried along with the normal data when a system call is made, as depicted in Figure. 6.2.

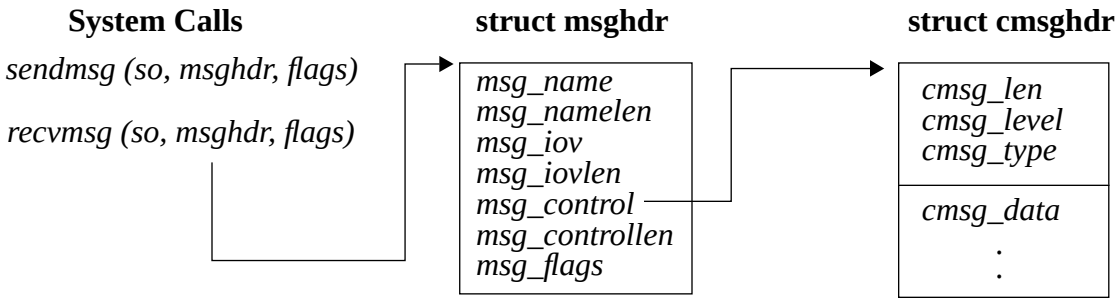

Figure 6.2: Ancillary data in NetBSD.

Control information is passed as follows: the send/recvmsg system calls take as an argument a rich data structure called msghdr. Structure msghdr contains the destination address of the message, the data for the message and a set of flags; in addition, this structure contains a pointer to another data structure of type cmsghdr which carries control data. The control data is preceded by a header containing the length of the data (cmsg_len), the level (or protocol) the data is destined for or received from (cmsg_level), and a protocol-specific type (cmsg_type).

The exchange of control data is depicted in Figure 6.3, and proceeds as follows: before every send/recvmsg system call, LTP prepares a cmsghdr structure; for a send call, data to be sent is pointed to by cmsg_data. For a receive call, an empty buffer is passed whose length is specified in msg_controllen. On sending, the data will be delivered to the protocol specified by the field cmsg_level; in the case of LTP/LMS this level is UDP. UDP passes the packet with the options unchanged to the IP layer, where the options are inserted into the packet, which is then passed to the network interface for transmission.

On the receiving side, the reverse takes place. The packet is delivered to the IP layer by the network interface, where the options are examined. If they are IP related options, they are processed by the ip_dooptions function. If the packet is not forwarded or it does not contain an error, processing of the packet continues at the IP layer, which finally delivers it to the UDP layer. If there are still options remaining in the packet and LTP has specified that it wants to receive such options

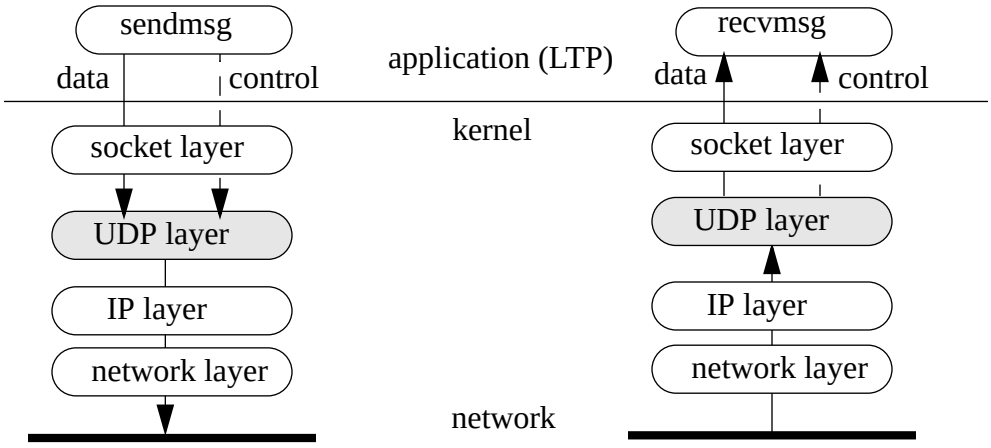

Figure 6.3: Exchange of control data

(appropriately done via the setsockopt system call; we will revisit this later), the options are appended to the socket buffer and delivered to LTP in the control portion of the recvmsg system call, in a cmsghdr structure.

We have briefly described how the send/recvmsg pair can be conveniently used to receive or deliver data and control information with every read or write request. Note that our use of this interface is consistent with what the others envision: the method we described is very similar to the method being considered for setting and retrieving information carried in IPv6 header options [41].

6.1.2. NetBSD Multicast Forwarding Architecture

Having described the mechanism by which LTP/LMS can exchange control data with the protocol stack, we turn our attention to multicast forwarding. Our aim is to describe what happens to a multicast packet once it has been received at the IP layer, which is where the multicast forwarding decisions are made. We will briefly cover route lookup and the multicast forwarding loop, where each interface is queried to see if there are members downstream that should receive the packet. We will not describe details like joining and leaving a group here; we will also not talk about routing, since LMS has no impact on routing whatsoever. We will simply restrict our discussion to the IP layer, since this is the only layer where changes are needed to accommodate our implementation of LMS.

In order to perform multicast forwarding, the NetBSD kernel must (a) be configured as a multicast router (i.e., compiled with the MROUTING option), and (b) must have two or more network interfaces. The kernel is only responsible for forwarding; all routing related operations like route discovery and update, are performed by a user-level program, the multicast routing daemon, or *mrouted*. mrouted runs the multicast routing protocol and exchanges routing information with neighboring mrouteds to update multicast routes. mrouted communicates with the kernel to update forwarding entries in the *multicast forwarding cache* (mfc), which is located in the kernel for performance reasons. The structure of an entry in the multicast forwarding cache is shown in Figure

```
(file: ip_mroute.h)
/*
 * The kernel's multicast forwarding cache entry structure.
*/
struct mfc {
        LIST_ENTRY(mfc) mfc_hash;
        struct   in_addr mfc_origin;            /* ip origin of mcasts */
        struct   in_addr mfc_mcastgrp;          /* multicast group associated */
        vifi_t   mfc_parent;                    /* incoming vif */
        u_int8_t mfc_ttls[MAXVIFS];             /* forwarding ttls on vifs */
        u_long   mfc_pkt_cnt;                   /* pkt count for src-grp */
        u_long   mfc_byte_cnt;                  /* byte count for src-grp */
        u_long   mfc_wrong_if;                  /* wrong if for src-grp */
        int      mfc_expire;                    /* time to clean entry up */
        struct   timeval mfc_last_assert;       /* last time I sent an assert */
        struct   rtdetq *mfc_stall;             /* pkts waiting for route */
    };
```

Figure 6.4: A multicast forwarding cache entry

6.4. There is one such entry in the cache for every <S, G> pair.

The multicast forwarding architecture is shown in Figure 6.5. A multicast packet arriving on a network interface is delivered to the IP layer through the function ipintr (like any other IP packet). If the kernel is configured as a multicast router kernel, multicast packets are directed through the function ip_mforward, which performs multicast forwarding. ip_mforward attempts to find the correct mfc entry using the macro MFCFIND; if no entry exists, mrouted is called to install one.

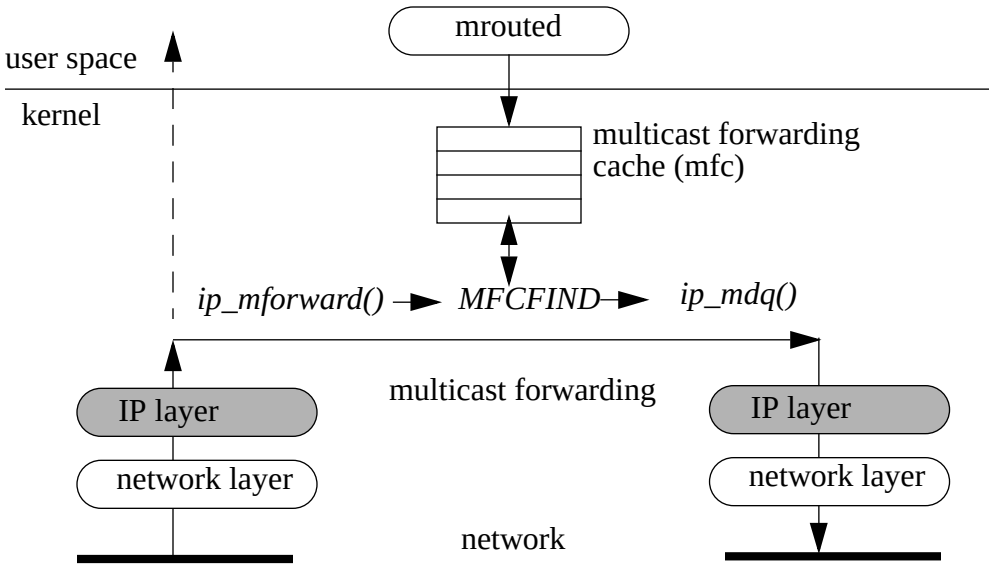

Figure 6.5: Multicast forwarding architecture in NetBSD

After an entry is found, ip_mforward calls ip_mdq, which checks if the packet arrived from the correct parent interface and loops through all outgoing interfaces sending the packet if (a) the packet's TTL exceeds the interface's threshold, and (b) there are group members downstream that interface. This function handles both real interfaces and tunnels. When ip_mforward returns, packet processing continues locally, in case there are multicast receivers on this host.

6.2. LMS in NetBSD

Having described the components in NetBSD relevant to LMS, we now proceed to describe how LMS is implemented. In our description we will list the files that were touched, describe the changes in existing functions and list new functions that we have written. We begin our discussion by describing the new IP options required to identify LMS messages. We then proceed to describe how LTP sends and receives messages containing LMS IP options and the information carried in them. We finish by describing how LMS messages are handled at multicast routers.

The files we changed in order to implement LMS are shown in Figure 6.6. Here we also list the

Files changed: *ip.h, ip_input.c, ip_mroute.h, ip_mroute.c, ip_var.h, udp_usrreq.c*

New IP Options: IPOPT_MREQ, IPOPT_DMCAST

New function name	Brief description
ipudp_encap	encapsulate a multicast packet into a unicast packet. Used by repliers when sending dmcasts
ip_domoptions	similar to ip_dooptions, this function processes multicast requests
ip_mfwdrequest	handles requests at the turning point router fills TP information
ip_dmcast	receives, decapsulates and dmcasts replies at the TP router

Figure 6.6: Summary of LMS implementation

new IP options and functions we added, the latter accompanied by a brief description of their operation. We will give more details of their use as we encounter them in our discussion below.

6.2.1. IP Options for LMS

LMS introduces two new types of messages that must be identified at the IP layer. These messages are (a) messages sent by LTP and destined for the replier, which we will call requests; and (b) messages sent by LTP containing directed multicasts, which are unicast messages encapsulating multicast messages and we will call dmcasts. We identify these types of messages by attaching to them two new IP options, named IPOPT_MREQ and IPOPT_DMCAST. Thus the first order of business is to define these two options, which is done as follows in the file *ip.h*:

```
/*
 * New LMS Options
 */
#define IPOPT_MREQ          138 /* LMS request */
#define IPOPT_DMCAST        139 /* LMS directed multicast */
```

The numbers we selected for these options are not important at the moment; we simply picked the next available option number from the options defined in ip.h. In a real implementation we would perhaps opt to introduce only one option (which could be called IPOPT_LMS) and then introduce sub-options to demultiplex LMS-related messages. However, for clarity, we will continue to use two IP options for the remaining of our discussion.

The next step is to define the structure of the IP options used by LMS. We can use the same structure for both requests and dmcasts, which we define in the file *ip_var.h*:

```
struct ipmopt_lms {
      u_int8_t   ipt_code              /* option type */
      u_int8_t   ipt_len;              /* option length */
      u_int16_t  ipt_tpif;             /* turning point iface */
      struct in_addr  ipt_tpaddr,      /* turning point addr */
                      ipt_src,         /* original source */
                      ipt_group;       /* multicast group */
}
```

The first field is the option type, which we described earlier. The second field is the option length, which includes the header. The third field is the identifier for turning point interface; in requests, it is filled by the turning point router; in dmcasts it contains the interface the packet should be multicast. The next three fields hold the internet addresses of the turning point router, the original source of the data requested or retransmitted, and the multicast group respectively. The last field, *ipt_group*, may appear redundant, since the address of the multicast group can be obtained from the destination address of the multicast packet. However, it is useful for repliers that may subscribe to multiple groups as an easy means to identify for which group the request is for.

6.3. LMS Implementation at Endpoints

So far we have defined two new IP options that distinguish requests and dmcasts, and we have also defined the structure of these options. We now proceed to describe how these options are used to send and receive requests and dmcasts at the endpoints. We start with requests.

6.3.1. Sending/Receiving Requests

LTP needs to send a request after is detects a gap in the sequence number of received packets. After a gap is detected, LTP creates a retransmission request with a list of the missing packets. The precise structure of the request is dependent on the particular transport protocol, but we expect that at a minimum it will contain the fields shown in Figure 6.7:

```
struct LmsNak {
      int   nak_lo;
      int   nak_hi;
      int   nak_seqn;
}
```

Figure 6.7: The data portion of a LMS request

The fields nak_lo and nak_hi specify the left and right edge of the gap; the field nak_seqn is the sequence number of the request (recall that the request may be sent multiple times in case of time-outs).

The above information is sent in the data portion of the request and is delivered as data to the replier; it is not examined by routers. Before the request is ready for transmission, LTP must prepare the control portion of the request which will be examined by each LMS router. Recall that the control portion will be delivered to the protocol via the sendmsg system call and will be carried with the request as an IP option. The control portion is prepared at the sending side as follows: LTP allocates a cmsghdr structure (see Figure 6.2), and fills its fields with the information shown in Figure 6.8. The cmsg_data field contains the information that will be placed in the IP option. In our implementation we chose to structure the control information as an IP option at LTP, in order to simplify the job of the transport protocol: thus we format the control information as the structure ipmopt_lms (see previous page); all UDP has to do once it receives the control information is strip the top portion and pass the cmsg_data portion to IP as an option. Following the creation of the IP option, LTP then allocates a msghdr structure, attaches the data portion of the request to the msg_iov field and the control portion to the msg_control field and finally calls sendmsg to send the request.

struct cmsghdr

cmsg_len = sizeof (struct cmsghdr) + sizeof (struct ipmopt_lms) *cmsg_level = IPPROTO_UDP* *cmsg_type = IPOPT_MREQ*
cmsg_data = struct ipmopt_lms *ipt_code =IPOPT_MREQ* *ipt_len = sizeof (struct ipmopt_lms)* *ipt_tpif = -1* *ipt_tpaddr = 0* *ipt_src = addr of sender* *ipt_group = addr of multicast group*

Figure 6.8: The control part of a LMS request

In the kernel the call reaches UDP, where the control portion is extracted and passed as an option to ip_output along with the actual request, as shown in Figure. 6.9. ip_ouput then forms

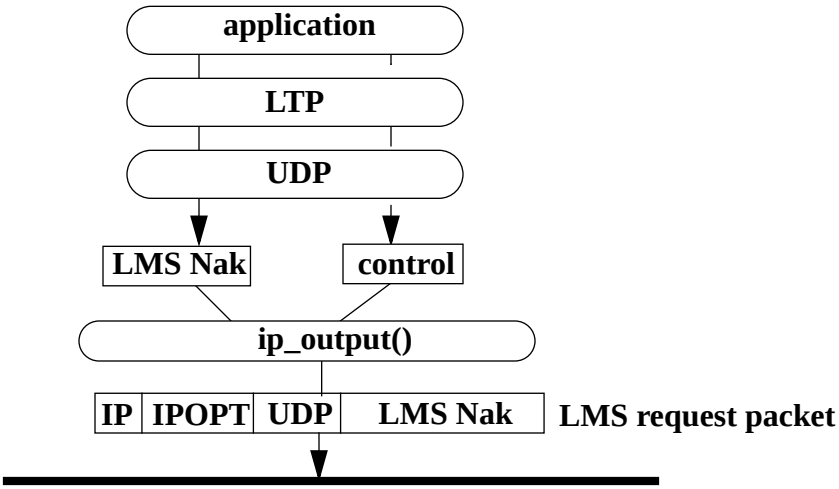

Figure 6.9: Combining control and data in a LMS request packet

the actual request packet, which is then multicast through the normal path.

The previous discussion described the sending of a request. We now proceed to describe what happens when a request is received. As one might expect, the operations are symmetrical. One

important difference, however, is that for an application like LTP to receive control data, it must explicitly tell the socket layer via a setsockopt system call. This is done as follows:

```
/*
* enable reception of IP options
*/
on = 1;
if (setsockopt (sock, IPPROTO_IP, IP_RECVOPTS, (char *)&on,
      sizeof (on)) < 0) {
      perror (“setsockopt IP_RECVOPTS”);
      exit (1);
}
```

With receiving of options enabled, reception of a request proceeds as follows: a request reaches the IP layer via the ipintr function. The IP packet looks exactly like the request in Figure 6.9, except that the turning point information has been filled when the request passed through the turning point router. IP separates the data and the IP option from the packet, and passes both to UDP. Since LTP has specified that it wants to receive options, UDP calls the function sbappendaddr which appends the data and the control information contained in the option to the receive socket buffer. LTP then retrieves both with the recvmsg system call.

Only one file was changed to accommodate the above implementation, and that is udp_usrreq.c. Changes required to the existing code to accommodate option processing for LMS as described above, were minimal: NetBSD version 1.40, which is the version we used, did not allow sending of data together with IP options. We simply removed the code that rejected options, and added simple code to strip the cmsghdr from the control information. Since the control data was correctly formatted as an option by LTP, the control data was passed directly to the lower layer.

6.3.2. Sending/Receiving DMCASTS

We now move to the description of how dmcasts are sent and received at the endpoints. Recall that a replier who wishes to send a reply targeted to a multicast subtree does so by sending a multicast packet encapsulated in a unicast packet at the turning point router.

The process of sending a dmcast is depicted in Figure 6.10. In our current implementation, LTP sends a dmcast in a manner similar to sending a request: LTP creates a message containing the reply

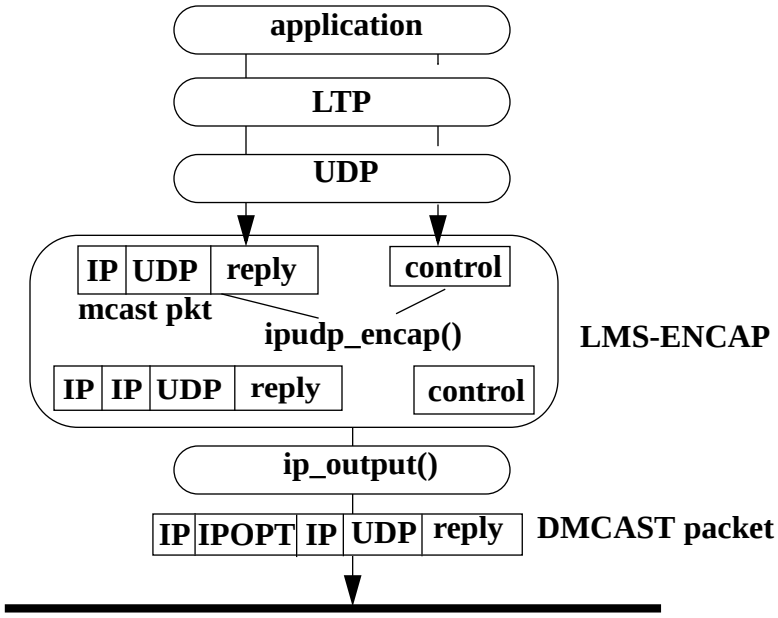

Figure 6.10: Sending a directed multicast at an endpoint

and a control message of type cmsghdr containing an ipmopt_lms structure with type IPOPT_DMCAST; unlike a request, where the turning point information is left empty, for a dmcast LTP fills the turning point fields with the control information received in the earlier request. LTP then calls sendmsg with both the reply and the control information, and multicasts the packet to the group.

When the packet reaches UDP it is intercepted, and passed to a new LMS function, ipudp_encap. This function receives from UDP the multicast packet containing the reply and the UDP/IP headers, and then prepares the IPOPT_DMCAST option by copying in the relevant control information provided by LTP. Then, it creates the encapsulating IP header, with the destination set to the router at the turning point (also provided by LTP). The encapsulating header's protocol field is set to IPPROTO_IPIP. Finally, the new packet and the option are passed to ip_output which attaches the IP option and unicasts it to the turning point router.

At the receiving side, no special operations take place since a dmcast becomes a regular multicast after the turning point router.

The services provided by LMS greatly simplify LTP's implementation over non-assisted protocols. When loss is detected LTP simply multicasts a message with the appropriate IP option, sets a timer, and waits for a retransmission. There is no topology state to guesstimate, no back-off timers for duplicate suppression, no need to scope retransmissions, no parent/children state, and no recovery groups to join. In that respect, LTP is more reminiscent of a unicast rather than a multicast protocol.

6.4. Handling LMS Messages at Routers

In the previous sections, we described how LMS requests and dmcasts are sent and received at the endpoints. In this section we describe how requests and dmcasts are handled at the routers. We start by describing the new state added at each LMS router, and continue to describe the new forwarding operations introduced by LMS.

6.4.1. New Router State

The new state LMS requires at the routers consists of two items: (a) the replier interface identifier, and (b) the replier cost. We added these items as new fields in the multicast forwarding cache entry, which was described earlier. We repeat the MFC entry here, in Figure 6.11, with the LMS fields included (shown in bold). Note that only four bytes are added for the LMS fields in the multicast forwarding cache entry. No other state is required for LMS.

6.4.2. Handling LMS Packets at a Router

The handling of LMS packets at the routers is depicted in Figure 6.12. When a multicast packet arrives at a router and is delivered to the IP layer, the function responsible for handling it is ipintr. We have modified the multicast handling portion of the input function ipintr to check for options in multicast packets. When a multicast packet is detected as carrying an option, it is passed to a new LMS function called ip_domoptions. Similar to the existing IP function ip_dooptions, ip_domoptions processes multicast options one at a time. Since currently the only two multicast options defined are ours, the packet is simply demultiplexed to one of two functions depending on the option it is carrying: ip_mfwdrequest, which handles multicast packets carrying the multicast option IPOPT_MREQ, or ip_dmcast, which handles multicast packets carrying the multicast option IPOPT_DMCAST. Shaded functions are new functions added by LMS.

```
(file: ip_mroute.h)
/*
 * The kernel's multicast forwarding cache entry structure, including the LMS state.
*/
struct mfc {
        LIST_ENTRY(mfc) mfc_hash;
        struct   in_addr mfc_origin;            /* ip origin of mcasts */
        struct   in_addr mfc_mcastgrp;          /* multicast group associated */
        vifi_t      mfc_parent;                 /* incoming vif */
        vifi_t   mfc_replier;                   /* the replier interface */
        u_int16_t mfc_replier_cost;             /* the replier cost */
        u_int8_t mfc_ttls[MAXVIFS];             /* forwarding ttls on vifs */
        u_long   mfc_pkt_cnt;                   /* pkt count for src-grp */
        u_long   mfc_byte_cnt;                  /* byte count for src-grp */
        u_long   mfc_wrong_if;                  /* wrong if for src-grp */
        int      mfc_expire;                    /* time to clean entry up */
        struct   timeval mfc_last_assert;       /* last time I sent an assert */
        struct   rtdetq *mfc_stall;             /* pkts waiting for route */
    };
```

Figure 6.11: LMS state at the router

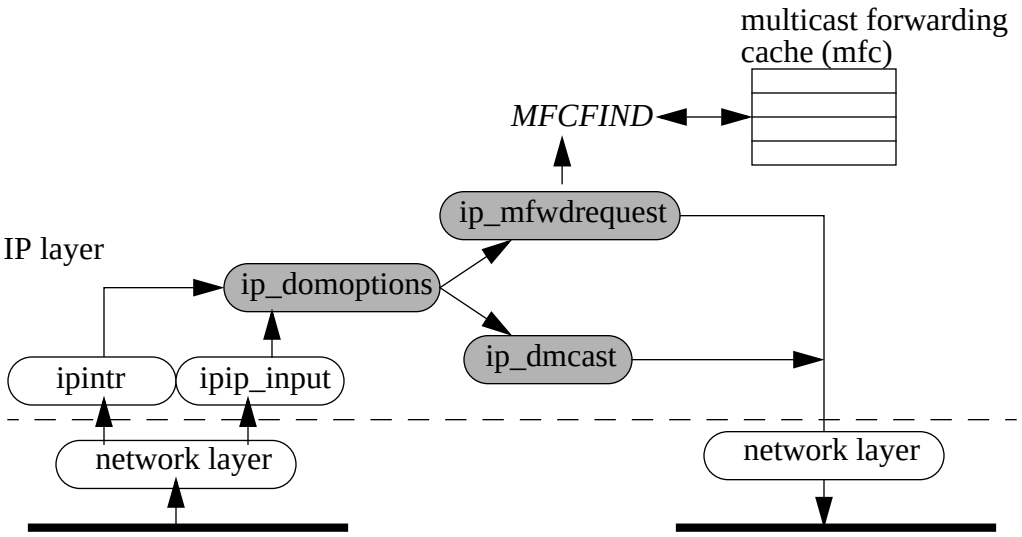

Figure 6.12: Handling of LMS packets at a router

6.4.3. Handling Requests at a Router

Requests are handled by ip_mfwdrequest. This function first copies the source and group addresses present in the option IPOPT_MREQ and uses them to perform a route lookup using the macro MFCFIND. The lookup returns the MFC entry for the original source. Then, the mfc_replier field in the entry is compared to the request's incoming interface. If they match, the packet came from the replier link, in which case it is sent to the mfc_parent interface; otherwise it is sent to the mfc_replier interface. In the latter case, before forwarding the packet, ip_mfwdrequest fills the turning point information in the IPOPT_MREQ option.

6.4.4. Handling DMCASTs at a Router

Having described how requests are handled at the router, we now proceed to describe the handling of dmcasts. Recall that a dmcast arrives at the router as a multicast packet encapsulated in a unicast packet. The encapsulation protocol is IPIP.

We have modified the ipip_input function to detect packets with multicast options and pass them to ip_domoptions. Thus, when a packet containing the IPOPT_DMCAST option arrives, it is passed to the function ip_dmcast, as shown in the previous figure. This function retrieves the interface index from the option, strips the unicast IP header and the option from the packet, and, if the interface index is valid, forwards the multicast packet out the specified interface.

6.4.5. A Conflict with NetBSD's Handling of Encapsulated Packets

Our current implementation of the dmcast service, pointed out an area in the NetBSD networking code that must be modified to accommodate dmcasts. Specifically, it relates to the way IP encapsulated packets are handled. NetBSD currently assumes that if an IPIP encapsulated packet is received, then the host must have a tunnel. Therefore, ipip_input, which is the function responsible for receiving IPIP packets and performing decapsulation, rejects any IPIP packets unless at least one tunnel exists at the host, and the originating host was the other end of the tunnel. Thus, clearly, a dmcast packet arriving at a host would be promptly rejected unless it arrives from the other side of a multicast tunnel. We avoid this problem in our implementation by adding the multicast option to the encapsulating IP header, detecting its presence in ipip_input, and immediately switching over to LMS processing. Ideally, we would like to see NetBSD being able to handle IPIP packets that

may arrive at a router outside a tunnel; when this capability is present, the multicast option that is now attached to the encapsulating header should be moved inside, to the encapsulated header. Thus, the encapsulated packet will pass through IP twice, once to remove the encapsulation header, and once more to process the multicast packet and its option.

6.5. Evaluation

Our evaluation is comprised of two parts: in the first part we state the LMS overhead in terms of control bytes that are exchanged between the endpoints and the routers. In the second part, we run experiments to compare the actual processing overhead of the LMS forwarding services with normal packet forwarding at the routers.

6.5.1. Request Overhead at Endpoints

The overhead in terms of bytes for sending and receiving requests is summarized in Figure.

Send overhead	Receive overhead
28 control bytes 16 byte IP option	28 bytes at recvmsg 16 bytes into sockbuf 16 byte IP option

Figure 6.13: Request overhead at the endpoints

6.13. The table shows overhead *in addition* to the normal sendmsg/recvmsg overhead. At the sending side, the overhead consists of 28 control bytes, out of which 16 bytes are for the IP option and 12 bytes are for cmsghdr. This is followed by a call to ip_pcbopts to prepare the IP option which is 16 bytes (option type and length, src addr, router addr, group addr, and router link). At the receiving side, the 16 bytes of the IP option are copied into a cmsghdr structure and delivered to LTP via the recvmsg system call in a cmsghdr structure.

6.5.2. Dmcast Overhead at Endpoints

The overhead for sending a directed multicast is similar to sending a request, except that now we have an additional IP header for the encapsulation. There is no additional receiving overhead, however, since dmcasts are received as regular multicast packets. The overhead is summarized in Figure 6.14.

Send overhead	Receive overhead
28 bytes at sendmsg encapsulation IP hdr 16 byte IP option	None

Figure 6.14: Dmcast overhead at the endpoints

We observe that the LMS overhead in terms of bytes is very small. At the host, LMS requires 28 more bytes of control data to be delivered to the kernel. If we assume an application the uses 1K size packets, this is only a 3% overhead. The number of bytes that actually reach the network, however, is even smaller, only 16 bytes. We believe this is negligible overhead, even for requests. For example, a TCP ACK packet is 40 bytes; an LMS packet including the LMS option and UDP/IP headers is only 44 bytes, without counting the actual NAK carried in the packet. The size of the NAK is 12 bytes; note, however, that the TCP SACK option[57] takes a minimum of 10 bytes (8*n+2 bytes, where n is the number of gaps specified in the option).

6.5.3. Processing Overhead

We have evaluated the processing overhead of our implementation of LMS using the testbed shown in Figure 6.15. Even though our topology is very simple, it still allows the measurement of processing cycles required for all LMS forwarding operations. We measured the forwarding overhead at the LMS router; we did not measure the additional processing at the hosts because our changes at the hosts were minor. Moreover, processing of a LMS packet at the endnode happens only twice, during send and receive, while processing at the routers can potentially happen many times, depending on how many routers a LMS packet visits. All three machines in our testbed were Pentium II class machines, connected together with a 155 Mbps ATM network. The multicast

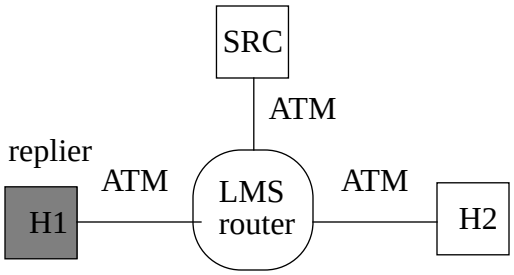

Figure 6.15: Experimental Testbed

sender was on host marked SRC; hosts H1 and H2 are the receivers. Host H1 (shaded) was designated as the replier.

The purpose of our experiments was twofold: (a) to verify that the forwarding operations in LMS worked correctly, and (b) to estimate their overhead and compare it with normal multicast forwarding. For the latter, we configured the processor at the LMS router to record the number of cycles spent on forwarding packets.

After setting up a multicast group that included all 3 hosts, we first verified that all receivers have joined the group successfully, and were receiving packets sent by SRC. Then, we conducted experiments to verify the correct operation of LMS. Specifically we ran experiments to test the following scenarios:

Scenario 1: Host H2 multicasts a request message. We verified that the request at the endpoint was sent correctly and contained the appropriate IP option, the LMS router received the message, filled out the turning point information, and forwarded it towards the replier interface. We verified that H1 received the message correctly and printed out its contents. We checked the turning point information and verified it to be correct.

Scenario 2: Host H1 multicasts a request message. We again verified the correct operation at the host, as before. We verified that the LMS router picked up the message and correctly forwarded it towards SRC. SRC received the message and printed out its contents, verifying that the packet was forwarded correctly.

Scenario 3: Host H1 sends a dmcast to the LMS router. We verified that the router received the correct message, decapsulated and multicast the message towards H2. Host H2 received the multicast packet correctly and printed out its contents.

After verifying that all LMS forwarding functions worked correctly, we set up our experiments to measure processing overhead. The measurements were taken using the processor cycle counter register in the Pentium processor. We measured the processing at the entire IP layer, from the moment a packet was received at IP until the packet was passed to the network interface. The machines we used were all 300MHz Pentium II class machines. The number of cycles was counted from when a packet entered the IP layer (at the beginning of function ipintr), until the packet exited the IP layer (right before ip_output calls the first network layer function).

Baseline Experiments

We ran two baseline experiments: in the first experiment, we sent about 6 million packets from SRC while only H1 was a member of the multicast group; in the second experiment we sent the same number of packets from SRC, but with both H1 and H2 being members. We measured the number of cycles spent at the router to forward packets in both experiments. These numbers provided us with a baseline estimate of how many cycles it takes to forward a regular multicast packets. The results are shown in Table 6.1.

LMS Experiments

Following our baseline experiments, we measured the processing overhead of the forwarding operations at the LMS router. Specifically, we ran two experiments: one to measure the processing overhead to forward a request, and another to measure the overhead for a dmcast. In the first experiment, host H2 sent about 6 million requests to the replier, which the router received and forwarded to H1. In the second experiment, host H1 sent about 6 million dmcasts which were multicast on the

interface leading to H2. The results of these experiments along with the baseline experiments, are summarized in Table 6.1.

Table 6.1: Forwarding cost: Normal v.s. LMS processing (300MHz Pentium II)

Normal IP mcast forwarding to 1 receiver	Normal IP mcast forwarding to 2 receivers	Forwarding a LMS request	Forwarding a LMS dmcast
3702 cycles	6686 cycles	3979 cycles	3734 cycles
12.3 µs	22.3 µs	13.3 µs	12.4 µs

The Table shows the average number of cycles spent at the IP layer to process each packet in the four experiments we described. The Table also shows the average number of microseconds taken to process each packet, which we obtained simply by dividing the number of cycles with the processor speed. A more detailed breakdown of the results is shown in in the plot in Figure 6.16.

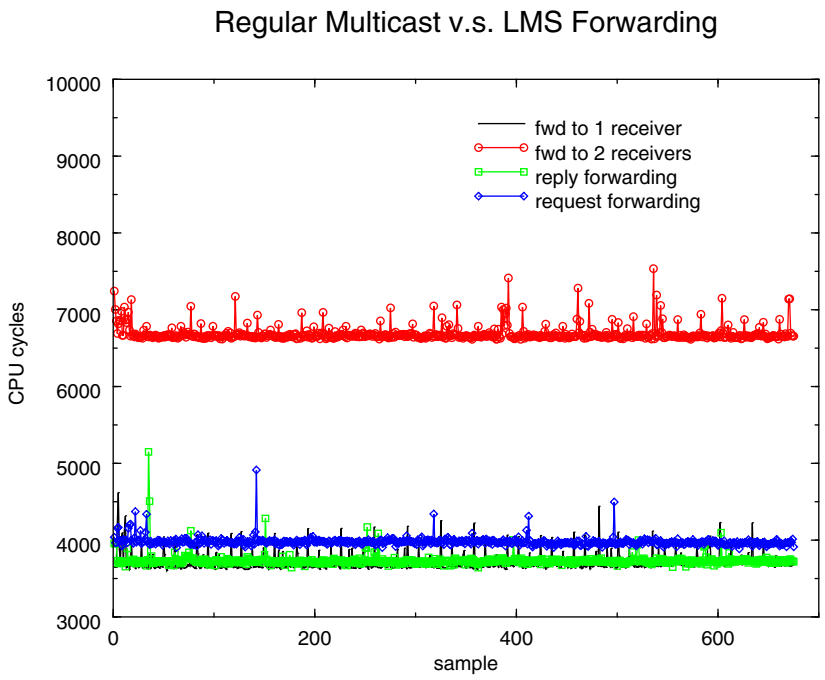

Figure 6.16: Cost of forwarding normal and LMS packets at a router

As we can see from both the table and the plot, the cost of forwarding LMS packets is approximately the same as the cost of forwarding a single multicast packet. It appears that the forwarding

cost of regular multicast packets increases almost linearly as the router has more member interfaces. The cost of LMS packets however, by design remains constant, regardless of how many member interfaces the router has.

The important result of this section is that the cost of forwarding LMS packets is at *most* on par with the cost of forwarding a regular multicast packet; moreover, it remains constant regardless of the router fan-out. This proves that LMS processing at the routers is *not* a bottleneck.

6.6. Summary

In this chapter we presented the implementation of LMS into the networking portion of NetBSD Unix. With our implementation we were able to provide answers to the questions we posed at the beginning of the chapter. Specifically:

Can LMS be implemented in software?

The answer to this question is “yes.” We have described how LMS can be implemented and we have identified the necessary modifications and supplied the new code that must be added to the existing networking code.

How much effort did it take?

The effort to introduce LMS to the existing architecture turned out to be minimal. The operations required by LMS are simple and easy to understand, and we were able to reuse many of the components of the existing architecture.

How much change did it require to the existing architecture?

Some. The modifications we had to make are, (a) allow LTP to specify options that should be inserted in multicast packets, and allow IP encapsulated packets to carry options. The first modification is very simple, and we expect that it will be in place soon to support IPv6 options [41]. The second, however, is a bit more involved. The current NetBSD code does not allow the reception of IPIP packets unless a tunnel is already in place; we got around this problem by switching over to LMS processing as soon as an IPIP packet with options was detected and performing IP decapsulation in the LMS portion. However, we believe that this should change in the future, if we are to

allow the implementation of services like dmcast. In terms of programming effort, this is a minor change.

How much overhead for LMS packet forwarding?

As we have seen from our experiments, the cost of LMS packet forwarding is on par with normal multicast forwarding when the router has only one downstream member interface. As more interfaces join the group, the cost of normal multicast forwarding increases almost linearly with the number of interfaces, but the cost of LMS forwarding remains the same. Thus, LMS processing at worst, is still the same as normal multicast forwarding. This is an important result because it negates any concerns about LMS adding too much processing at the routers.

In summary, in this chapter we have shown that LMS is not only implementable, but it requires very little effort to implement, only minor changes to existing systems and introduces no more processing overhead than normal multicast. Therefore, our work has provided very strong evidence that there are no serious obstacles in implementing LMS.

6.7. Future Work

In this section we discuss two areas of future work in LMS. The first is the implementation of the LMS_HIER module, and the second is an exploration of whether LMS can be incorporated in the router fast path.

As we mentioned at the beginning of this chapter, we did not include the LMS-HIER module in our implementation. The reason is that it is not yet clear what the functionality of this module should be. For example, one may argue that LMS may require no such module, and routers can simply pick repliers at random with no feedback from receivers. Such an option is attractive in backbone routers, which may typically handle hundreds of thousands of groups at a given time. Our simulation experiments have indicated that the performance of LMS in the presence of random loss is good, even if repliers are picked at random.

Another example where the LMS-HIER component may not be needed is a randomized approach is proposed by Costello and McCanne, called Search Party [30]. This work, which builds on our work by aiming to provide LMS with better robustness and better load distribution at the

expense of increased duplicates and latency. To do so, Search Party allows routers to randomly distribute multiple copies of requests between the downstream and the upstream links. In such an approach, the router does not maintain a specific replier link, and thus does not require the LMS-HIER component.

The reason our experiments and the Search Party approach indicate that the LMS-HIER component may not be required, is that for error control there is a certain flexibility as to where data is recovered from; in other words, in error control, who sends the retransmission is less important; what is of primary importance is that a retransmission is sent. Who sends a retransmission, however, becomes important when we consider issues like security and latency. In addition, as we described in Chapter 4, we envision that LMS will be used for other applications besides error control where tight control of the replier selection may be crucial; thus, if LMS is to be used in such applications, the LMS-HIER component must be implemented. Below we sketch a possible implementation of an LMS-HIER component that maintains tight control over the replier selection. The implementation details of the LMS-HIER module are left as part of future work.

6.7.1. The LMS-HIER Component

Recall that the main purpose of LMS-HIER is to create, disseminate and maintain the replier information among all endpoints and routers participating in a multicast group. This information is maintained on a per-multicast tree basis, i.e., for each <*src, group*> pair in the case of source-based trees, or each <*group*> in the case of shared trees. To do so, LMS-HIER needs to communicate information between (a) the endpoints and the local router, and (b) the network routers that participate in the multicast tree. We call the former the *host-router component*, and the latter the *router-router component*.

The Host-Router Component

To implement the host-router component, we propose two small modifications to the Internet Group Management Protocol (IGMP). IGMP is the protocol used between hosts and routers on the same physical network to tell all systems which hosts belong to which multicast groups. This information is required by routers to know which multicast packets to forward on which network. IGMP is defined in [43].

Briefly, IGMP works as follows: to join a group, a host multicasts an IGMP REPORT message with destination address set to the address of the group it wants to join. Routers by default receive all multicast messages, and are thus notified of the wishes of this host and take appropriate actions to join the group. To maintain group membership, the local router periodically generates an IGMP QUERY message, which it multicasts to the *all-hosts* group address (224.0.0.1), which all multicast-capable hosts are required to join; hosts respond with an IGMP REPORT message for every multicast group they currently subscribe to. A simple randomized back-off algorithm with duplicate suppression ensures that the router is not flooded with reports. IGMP version 2[43], introduces a group-specific query message, and an IGMP LEAVE message. IGMP version 3[58], which is still at a preliminary stage of development, extends IGMP v.2 with GROUP-SOURCE REPORTs and LEAVEs, to allow receivers to receive packets only from a specified set of sources rather than everyone in the multicast group.

To support the LMS-HIER component we propose to modify IGMP v.3 source-group reports to carry an additional field, namely the replier cost. This field is set by receivers wishing to act as repliers. In addition to this new field, one more change is needed to IGMP: in response to a router's IGMP QUERY message, elected repliers instead of participating in the randomized back-off contest, respond immediately with an IGMP REPORT message; other receivers' responses are suppressed as before. If a potentially large number of distinct groups with distinct repliers exist on a particular network, the randomized back-off contest may be re-introduced, but receivers who are not repliers scale the randomized back-off interval to allow repliers to contest first. Repliers are also allowed to send unsolicited IGMP reports (i.e., ones that were not triggered by IGMP queries) when their replier cost has changed. The router accepts these reports and updates its replier cost accordingly.

We believe that the above modifications are simple and will not disrupt the existing operation of IGMP.

The Router-Router Component

Recall that the router-router component of LMS-HIER requires that routers send information to upstream routers to update the replier cost. One approach, shown in Figure 6.17, is to use existing messages from multicast routing protocols such as DVMRP as vehicles to carry the LMS replier

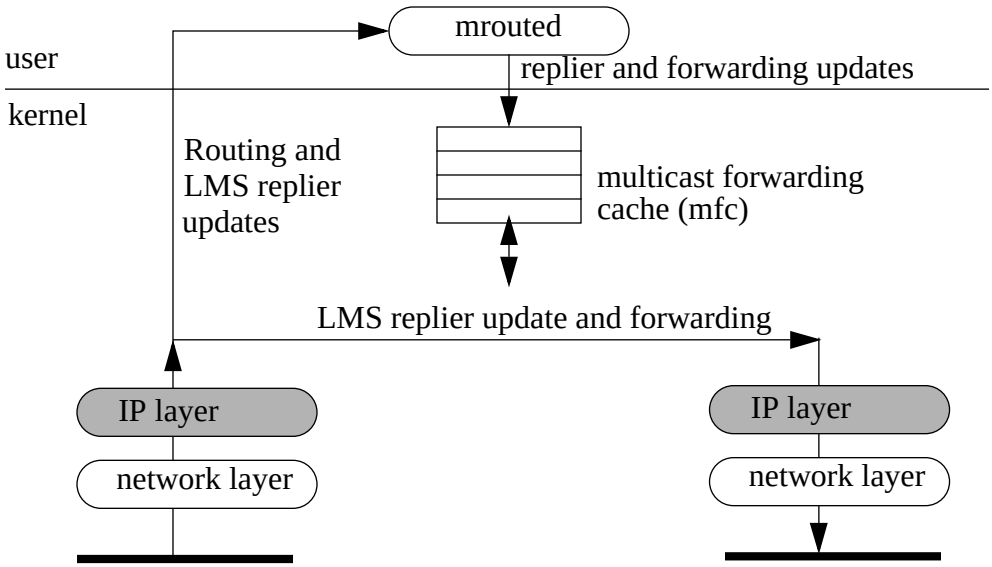

Figure 6.17: Combined routing/LMS updates at a router

information. Such messages are exchanged frequently between routing daemons like mrouted. By piggybacking replier information to routing updates, replier updates can be exchanged between mrouteds, which in turn can update the replier information by making the appropriate changes to the kernel multicast forwarding cache. One problem with this approach is that replier updates are scheduled only when routing updates take place, which may not be frequently enough to guard against replier failure.

Another approach, which responds faster to replier failure, is to create a new replier update protocol for the exclusive use of LMS. The protocol would be similar to protocols used for routing updates, in that it would exchange messages between neighboring mrouteds. This protocol should allow both periodic and on-demand replier updates. Periodic updates would prevent soft state from expiring, and on-demand updates would allow fast convergence of the replier hierarchy.

6.7.2. LMS in the Fast Path

In addition to the implementation of LMS-HIER component, another important question we are not able to answer in this chapter is whether LMS can be incorporated into routers which employ a *fast forwarding path*. A fast path is a highly optimized version of the portion of the forwarding code

used during the common case forwarding of a packet; for speed reasons, routers (like high-end backbone routers) implement this portion in hardware. However, evaluating whether LMS can be deployed in a router's fast path is a difficult without access to information about the hardware architecture of such routers. While in this chapter we can only deal with the issue of integrating LMS into an existing software architecture, our work is still useful for the following reasons:

- Not all routers in the Internet have a hardware-supported fast path, and thus our software implementation may be adopted directly by such routers.

- Our software implementation shows that the added code is similar in complexity, structure and overhead to the existing forwarding code. This leads us to speculate that if the existing forwarding operations can be implemented in hardware, then it is very likely that LMS can also be implemented in hardware.

- Many routers come with line cards which contains processors running software. Such line-cards can very easily accommodate LMS.

As future work, we would like to evaluate the possibility of a fast path implementation of LMS and in router line cards.

Chapter 7

Conclusions and Future Work

In this thesis we addressed the problem of providing error control for two important classes of emerging Internet applications. These classes are (a) interactive continuous media applications (CM) and (b) large-scale multicast applications.

7.1. Interactive Continuous media Applications

We proposed an error control scheme for interactive CM applications that uses retransmission, an approach which was deemed inappropriate until a few years ago. By using a number of techniques including playout buffering, gap-detection and fast, conditional retransmission, we showed through implementation and experimentation, that despite popular belief, such an approach is not only feasible but also highly effective in reducing observed loss. Our experiments recorded improvements of several orders of magnitude, without any violation of the latency constraints of interactive CM applications.

7.1.1. Contributions

Our work on error control for interactive CM applications, was one of the first to demonstrate the feasibility of retransmission. Since then more work has been done by others that had lead to similar conclusions [5, 6]. Our contributions are as follows:

- We demonstrated the feasibility of retransmission by designing, implementing and testing a full-fledged continuous media transport protocol in the kernel of NetBSD Unix.

- We showed via analysis and experimentation over random, bursty and MPEG traffic, that our protocol reduces observed loss by orders of magnitude without violating the timing constraints of continuous media applications. Moreover, we have shown that our scheme sends retransmissions only when they are useful, which incurs minimal cost.

- We subjectively verified the effectiveness of our scheme using a raw video stream, and the results were observed to be markedly superior to the results without retransmission.

In summary, we believe we have presented strong evidence that retransmission can be effective in reducing loss in a wide range of CM applications and produced a real protocol to prove our hypothesis.

7.1.2. Future Work

We have already identified one area of future work in Chapter 2. We repeat it here and list more possible areas:

- **Extending to multiple retransmissions**: An important enhancement of our protocol is to extend it to attempt multiple retransmissions. Such an enhancement will be of great value in environments where the propagation delay between the sender and the receiver is small; examples include LANs and Campus networks. Having presented the design of this extension, we can implement and test it similar to the basic scheme.

- **Large-scale, continuous media multicast**: We can merge the error control schemes we presented in this thesis to develop an error control scheme for large scale CM multicast applications. This is another important class of applications, which includes distance learning, conference meetings, and multi-player games. We believe that LMS can provide a solid platform on which to develop such a scheme, because it provides the lowest recovery latency of all retransmission-based[1] schemes.

- **Enhancing CM applications to use protocol assistance**: Currently, most CM applications do not assume any help from the transport layer to aid in concealment; however,

1. excluding those using a FEC/retransmisison combination.

our protocol provides information that potentially can be very valuable to an application. We can modify an existing application to use protocol assistance and evaluate the benefits.

7.2. Large-Scale Multicast Applications

In order to solve the problem of scalable multicast error control, we enhanced the IP Multicast service model with a set of Light-weight Multicast Services (LMS). LMS enhances routers with a novel, yet simple forwarding functionality, which enables routers to structure receivers in a hierarchy with minimal or no involvement from receivers. A hierarchy, is an efficient way of achieving scalability. Locating LMS at the routers allows automatic placement of receivers on a recovery tree that exactly mirrors the underlying multicast tree, greatly improving the efficiency of error control. Schemes that do not use router assistance in building the recovery tree, resort to imprecise heuristics which degrade efficiency. LMS allows children to easily communicate with their parents to request retransmissions without implosion; it also allows parents to easily identify the subset of the children that suffered loss and accurately aim the retransmission.

We have demonstrated through simulation that with LMS, the performance of error recovery improves significantly over non-assisted schemes: implosion is eliminated, exposure is kept at negligible levels, and recovery latency is nearly optimal. In addition to improving performance, LMS greatly simplifies receivers by freeing them from the burden of topology discovery. We have also demonstrated through implementation and experimentation that LMS is easy to implement, integrates well with the existing multicast architecture, and its performance is as good or better than normal multicast.

7.2.1. Contributions

Our approach to the problem of scalability in error control for large multicast applications came from a new angle. We realized that most of the complexity in existing solutions comes from the difficulty in dealing with a dynamic, but largely unknown group topology. Without knowledge of topology, it becomes hard to coordinate group members to avoid problems like implosion, and certain penalties must be endured in order to achieve scalability. It is perhaps not an exaggeration to say that a disproportionately large part of the effort expended in most reliable multicast protocols

proposed today, deals with gathering information about topology, either through measurement or heuristics. Our research showed how to solve this difficult problem as follows:

- We made the observation that forwarding and error control are two clearly separable components, and great benefits can be realized by decoupling and placing each one where it is more beneficial. We believe that the forwarding component belongs to the routers (after all this is what routers do best), and the actual error control component (detection, notification and recovery) belongs to the receivers. This separation is very clean, i.e., it does not violate any layering principles like the end-to-end argument [38] - the routers do not see any transport layer information and the endpoints know nothing about topology.

- We showed how to enhance the IP Multicast model to accommodate a new set of forwarding services (LMS) that are highly beneficial for efficient and scalable error control. The enhancements are in the form of simple, easy to understand high-level operations, like “send a request to the parent”, or “send a retransmission to part of the multicast group”.

- We designed and implemented the LMS algorithms, and showed that the programming effort involved is minimal.

- We designed and created a simulation of LTP, a transport protocol that uses LMS. We determined that by using LMS LTP becomes much less complex than non-assisted protocols. Its complexity is in fact on par with an equivalent unicast protocol.

- We implemented and verified the operation of LMS; our evaluations showed that the processing overhead of LMS at the routers is at least as good as regular multicast processing, thus dispelling any doubts as to whether LMS can be implemented in the routers. Our implementation showed that LMS requires minimal changes to existing code (about 250 lines of new C-code)

- We used simulation to evaluate the performance of LTP/LMS and compare it with SRM and PGM; we demonstrated that the performance of the LTP/LMS combination is three to ten times better than both these protocols, in terms of latency. In terms of exposure, LMS is only slightly worse than PGM, which eliminates exposure. SRM without local recovery

suffers dramatically from exposure compared to LMS, which offers orders of magnitude lower exposure than SRM.

- We discussed how LMS can be useful for other applications, like building an ACK-based protocol, scalable voting, simple anycast and maintaining congruency in other tree-based protocols.

In summary, one of the more significant contributions of our work is that it has provided us with a vantage point from where to try to solve other important, yet difficult problems like multicast congestion control.

7.2.2. Future Work

We have already discussed some areas of future work in LMS, in Chapters 4 and 6. We recap our discussion here, and add some further thoughts:

- **Replier State**: As we discussed in Chapter 6, we did not implement the LMS component responsible for propagating the replier state in the network (the LMS-HIER component). We sketched an approach on how to implement this component, utilizing small modifications to IGMP and the multicast routing daemon (mrouted). Completing the design and implementation of this component would be an important addition to LMS.

- **Incremental Deployment**: It is highly unlikely that LMS can be deployed in all the routers on the Internet at once. Thus, any scheme like LMS that proposes modification to the existing multicast model, must develop a plan for incremental deployment. We have discussed an incremental deployment approach in Chapter 4, using source-path messages, similar to PGM. In addition, we believe it is possible that LMS can gain leverage off the plan for deployment of IPv6[62]. We have not worked out the details of such a task yet; however, it is important to study its feasibility and propose a deployment plan.

- **LMS in the router fast-path**: In Chapter 6, we discussed the possibility of introducing LMS into the hardware fast-path of certain high-end routers. We claimed that since we have demonstrated that LMS is no more complex than normal multicast forwarding, it is

very likely that it can be implemented in hardware. We can verify our claim by studying the architecture of such routers and determining the feasibility of such a venue.

- **Security**: As we discussed on Chapter 4, LMS introduces no security holes in the current IP model. However, further studies may be conducted to determine how to integrate LMS with currently proposed security schemes.

- **Other applications and services**: We have claimed that LMS offers services that are general enough to be used by other multicast applications. LMS, as currently presented, enables general services including a *scalable collect service*, which enables scalable feedback from any number of receivers to the sender, and *fine grain multicast*, where a message can be accurately aimed towards a subset of the receivers in a multicast group. Other general services can be implemented on top of the LMS services, and a detailed study can be carried out to define them. In addition, encouraged by the clean separation of functionality that LMS achieves with error control, we can explore the possibility of doing the same for other difficult applications, for example congestion control. In the process, we may define other useful building blocks to add to LMS.

- **Integration with existing services**: There have been some recent proposals for router services to improve the performance of existing protocols, including ECN[59], RED[60], and CBQ[61]. We can investigate the possibility of integrating LMS with such services.

7.3. Source Code Availability

The source code for our *ns* simulations and Implementation can be obtained by contacting the author at http://www.ccrc.wustl.edu/~christos.

7.4. Closing Remarks

Will multicast ever become ubiquitous in the Internet? That is a question whose answer seemed clear a few years ago, when researchers believed in the obvious advantages of multicast. Despite the initial optimism, multicast has not taken off yet. Deering, the chief architect of IP Multicast along with many other prominent researchers, have recently expressed skepticism and disappointment[84] on the current state of multicast, but maintained their optimism about its fate. There are

many theories about why multicast is off to a slow start, ranging from the lack of a “killer application,” fears from Internet Service Providers (ISPs) about users flooding their networks, to ISPs not knowing how to charge for multicast services. Recent work has tried to address some of these issues. Express [82] and Simple Multicast [83] are two very recent approaches that propose to simplify the existing multicast model to satisfy some of the ISPs’ concerns.

We are at a point where the Internet has been heavily commercialized, and other issues are beginning to emerge that were mostly brushed away when the Internet was a predominantly academic entity. These include billing, security, routing policy, and more. The Internet has also grown so big, that making any fundamental changes is becoming increasingly difficult (some will say impossible). This is probably the major criticism of our work: even if LMS satisfies the skeptics, will it be possible to deploy it? Our answer is that Cisco is already attempting to do just that with PGM, which faces similar obstacles. Thus, it may still be possible to make changes inside the network, we just hope that you do not have to be a giant router company to do so.

REFERENCES

[1] Tanenbaum, A., Computer Networks, Prentice Hall, 1988.

[2] Comer, D., Internetworking with TCP/IP, Prentice Hall, 1991.

[3] Frederick R., Network video (nv) Xerox Palo Alto Research Center, ftp://ftp.parc.xerox.com/pub/net-research/nv-3.3beta/.

[4] McCanne, S., Scalable Compression and Transmission of Internet Multicast Video, Ph.D. thesis, University of California Berkeley, UCB/CSD-96-928, December 1996.

[5] Peijhan, S., Scwartz, M., Error Control Using Retransmission Schemes in Multicast Transport Protocols for Real-Time Media, IEEE/ACM Transactions on networking, pp. 413-427, Vol. 4, No 3, June 1996.

[6] Rhee, I., Error Control Techniques for Interactive Low-bit Rate Video Transmission, Proc. of ACM Sigcomm, pp. 290-301, October 1998.

[7] Ammar, M., Cheung, S., Li, X., “Improving fairness in Multicast Video Distribution,” Tenth Annual IEEE Workshop on Computer Communications, September 1995.

[8] Biersack, E., “Performance Evaluation of Forward Error Correction in ATM Networks,” ACM Sigcomm ‘92, pp. 248-258, August 1992.

[9] Bolot, J., Turletti, T., “A Rate Control Mechanism for Packet Video in the Internet,” IEEE Infocom ‘94, pp. 1216-1223, June 1994.

[10] Brady, P., “Effects of Transmission Delay on Conversational Behavior on Echo-free Telephone Circuits,” Bell System Technical Journal, Vol. 50, No. 1, pp.115-134, January 1971.

[11] Cheriton, D., “VMTP: Versatile Message Transaction Protocol,” RFC 1045, Stanford University, Feb. 1988.

[12] Chiang, T., Anastassiou, D., “Hierarchical coding of digital television,” IEEE Communications Magazine, vol. 32, pp. 38-45, May 1994.

[13] Clark, D., Lambert, M., Zhang, L., “NETBLT: A Bulk Data Transfer Protocol,” RFC 998, MIT, 1987.

[14] Delgrossi, L., Halstrick, C., Herrtwich, R., Hoffmann, F., Sandvoss, J., Twachtmann, B., “Reliability Issues in Multimedia Transport,” Second workshop on High-Performance Communication Subsystems (HPCS’93), Williamsburg, VA, September 1993.

[15] Dempsey, B., Liebeherr, J., and Weaver, A., “On Retransmission-based Error Control for Continuous Media Traffic in Packet-switching Networks,” Technical Report CS-94-09, University of Virginia, Charlottesville, Virginia, Feb. 1994.

[16] Doeringer, W., Dykeman, D., Kaiserswerth, M., Meister, B., Rudin, H., Williamson, R., “A Survey of Light-Weight Transport Protocols for High-Speed Networks.” IEEE Transactions on Communications, Nov. 1990.

[17] Floyd, S., Jacobson, V., McCanne, S., Liu, C., Zhang, L., “A Reliable Multicast Framework for Light-Weight Sessions and Application Framing,” Proc. of ACM Sigcomm ‘95, pp. 342-356, Cambridge MA 1995.

[18] Gong, F., Parulkar, G., An Application-oriented Error Control Scheme for a High-Speed Transport Protocol,” Technical Report WUCS-92-37, Washington University, 1992.

[19] Grossglauser, M., Keshav, S., Tse, D., “RCBR: A simple and Efficient Service for Multiple Time-Scale Traffic”, ACM Sigcomm ‘95, pp.219-230, August 1995.

[20] Jacobson, V., Braden, R., Borman, A., “TCP Extensions for High Performance,” RFC 1323, May 1992.

[21] Jacobson, V., Rudin, H., “Protocols for High Speed Networks”, Tutorial notes, ACM Sigcomm ‘90, Philadelphia, September 1990.

[22] Lin, J., Paul, S., “RMTP: A Reliable Multicast Transport Protocol”, Infocom ‘96, pp.1414-1424, March 1996.

[23] Netravali, A., Roome, W., Sabnani, K., “Design and Implementation of a High-Speed Transport Protocol,” IEEE Transactions on Communications, Nov. 1990.

[24] Oguz, N., Ayanoglu, E., “Performance Analysis of Two-Level Forward Error Correction for Lost Cell Recovery in ATM Networks”, Infocom ‘95, pp. 728-737, April 1995.

[25] Papadopoulos, C., Parulkar, G., “Implosion Control in Multipoint Transport Protocols”, Tenth Annual IEEE Workshop on Computer Communications, September 1995.

[26] Ramamurthy, G., Raychaudhuri, D., Performance of Packet Video with Combined Error Recovery and Concealment,” Infocom’95, pp. 753-761, April 1995.

[27] Shulzrinne, H., Casner, S., Frederick, R., Jacobson, V., “RTP: a Transport Protocol for Real-Time Applications,” Internet Draft draft-ietf-avt-rtp-07, March 1995.

[28] Masahiro Wada, “Selective recovery of video packet loss using error concealment,” IEEE Journal on Selected Areas in Communications, vol. 7, pp. 807-814, June 1989.

[29] Deering, S., “Host Extensions for IP Multicasting,” RFC 1112, January 1989.

[30] Costello, A., McCanne, S., “Search Party: Using Randomcast for Reliable Multicast with Local Recovery”, Proceedings of INFOCOM ’99, March 21, 1999, New York, NY.

[31] Floyd, S., Jacobson, V., McCanne, S., Zhang, L., Liu, C., “A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing,” Proc. of ACM Sigcomm’95, pp. 342-356, September 1995.

[32] Holbrook, H., Singhal, S., Cheriton, D., “Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation,” Proceedings of ACM Sigcomm’95, Vol. 25, No. 4, pp. 328-341, October 1995.

[33] Levine, B., Garcia-Luna-Aceves, J.J., “Improving Internet Multicast with Routing Labels”, Proc. of IEEE ICNP, Atlanta, GA, Oct. 1997, http://www.cse.ucsc.edu/research/ccrg/publications.html.

[34] Papadopoulos, C., Parulkar, G., Varghese, G., “An Error Control Scheme for Large-Scale Multicast Applications”, Proc. of IEEE INFOCOM’98, San Francisco, CA pp.1188-1196, March 1998.

[35] Paul, S., Sabnani, K.,. Buskens, R., Muhammad, S., Lin, J., Bhattacharyya, S., “RMTP: A Reliable Multicast Transport Protocol for High-Speed Networks,” Proceedings of the Tenth Annual IEEE Workshop on Computer Communications, September 1995.

[36] Pingali, S., Towsley, D., Kurose J., “A Comparison of Sender-initiated and Receiver-initiated Reliable Multicast Protocols,” JSAC, April 1998.

[37] Postel, J., “Transmission Control Protocol - Darpa internet Protocol Program Specification,” RFC 793, September, 1981.

[38] Saltzer, J.H., Reed, D.P., Clark, D.D., “End-to-End Arguments in System Design,” ACM Transactions on Computer Systems, Vol. 2, No. 4, November 1984. pp 277, 288.

[39] Yajnik, M., Kurose, J., Towsley, D., “Packet Loss Correlation in the MBONE Multicast Network: Experimental Measurements and Markov Chain Models,” IEEE Global Internet Conference ‘96.

[40] Yavatkar, R., Griffioen, J., Sudan, M., “A Reliable Dissemination Protocol for Interactive Collaborative Applications,” Multimedia’95.

[41] Stevens, W., Thomas, M., “draft-stevens-advanced-api-03.txt”, work in progress.

[42] Gilligan, R., Nordmark, E., “Transition Mechanisms for IPv6 Hosts and Routers”, RFC 1933, April 1996.

[43] Fenner, W., Internet Group Management protocol, Version 2, RFC 2236, November 1997.

[44] Katz, D., Atkinson, R., IPv6 Router Alert Option, Internet Draft draft-cisco-ipv6-router-alert-01.txt, work in progress.

[45] Partridge, C., Mendez, T., Milliken, W., “Host anycasting service,” RFC 1546, November 1993.

[46] Mittra, S., “Iolus: A Framework for Scalable Secure Multicasting,” Proc. of the ACM Sigcomm ‘97, Cannes, France, September 1997.

[47] Speakman, T., Farinacci, D., Lin, S., Tweedly, A., “Pragmatic General Multicast (PGM) Transport Protocol Specification”, draft-speakman-pgm-spec-03.txt, work in progress, June 1999.

[48] UCB/LBNL/VINT Network Simulator - ns (version 2), Software on line, http://www-mash.cs.berkeley.edu/ns/.

[49] Zegura, E., Calvert, K., and Bhattacharjee, S.,. "How to Model an Internetwork." Proceedings of IEEE Infocom ’96, San Francisco, CA.

[50] Paxon, V., “End-to-End Internet Packet Dynamics,” IEEE/ACM Transactions on Networking, Vol. 7, No. 3, pp277-292, June 1999.

[51] Handley, M., An Examination of MBone Performance USC/ISI Research Report: ISI/RR-97-450, January 1997.

[52] Doar, M., Leslie, I., “How bad is naive multicast routing?” Proc. IEEE INFOCOM ‘93, pp. 82-89, 1993.

[53] Mitzel, D., Shenker, S., “Asymptotic resource consumption in multicast reservation styles,” Proc. ACM Sigcomm’94, pp 226-233, 1994.

[54] Wei, L., Estrin, D., “The trade-offs of multicast trees and algorithms,” Proc ICCCN’94, 1994.

[55] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network Visualization with the VINT Network Animator Nam. Technical Report 99-703, University of Southern California, March, 1999. <http://www.isi.edu/~johnh/PAPERS/Estrin99d.html>.

[56] The Reliable Multicast Research Group, http://www.east.isi.edu/rm/.

[57] Mathis, M., Mahdavi, J., Floyd, S., Romanow, A., “TCP Selective Acknowledgment Options,” RFC 2018, October 1996.

[58] Cain, B., Deering, S., Thyagarajan, A., “Internet Group Management Protocol, Version 3,” Internet Draft draft-ietf-idmr-igmp-v3-01.txt, work in progress.

[59] Ramakrishnan, K.K., and Floyd, S., A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481, January 1999.

[60] Floyd, S., and Jacobson, V., Random Early Detection gateways for Congestion Avoidance IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p.397-413.

[61] Floyd, S., and Jacobson, V., Link-sharing and Resource Management Models for Packet Networks IEEE/ACM Transactions on Networking, Vol. 3 No. 4, pp. 365-386, August 1995.

[62] Internet Protocol, Version 6 (IPv6) Specification <draft-ietf-ipngwg-ipv6-spec-v2-02.txt> (replaces RFC 1883), December 1998, work in progress.

[63] 6Bone, http://www.6bone.net/.

[64] B.N. Levine, David Lavo , and J.J. Garcia-Luna-Aceves, ‘‘The Case for Concurrent Reliable Multicasting Using Shared Ack Trees,’’Proc. ACM Multimedia 1996 Boston, MA, November 18--22, 1996.

[65] B.N. Levine, S. Paul, and J.J. Garcia-Luna-Aceves, ‘‘Organizing Multicast Receivers Deterministically According to Packet-Loss Correlation,’’ Proc. Sixth ACM International Multimedia Conference (ACM Multimedia 98), Bristol, UK, September 1998.

[66] Fenner, W., Casner, S., “A traceroute facility for IP Multicast” Internet Draft draft-ietf-idmr-traceroute-ipm-02.txt, work in progress, 1997.

[67] Birman, K., Joseph, T., “Reliable Communication in the presence of failures:, ACM Transactions on COmputer Systems, 5(1):47-76, February 1987.

[68] Cheriton, D., Zwaenepoel, W., “Distributed Process Groups in the V kernel”, ACM Transactions on Computer Systems, 3(2):77-107, May 1985.

[69] Chang, J., Maxemchuck, N., “Reliable Broadcast Protocols”, ACM Transactions on COmputer Systems, 2(3):251-273, August 1984.

[70] Kaashoek, M., Tanenbaum, A., Humel, S., Bal, H., “An Efficient Reliable Broadcast Protocol”, ACM Operating Systems Review, 23(4), October 1989.

[71] Laurence C. N. Tseung, “Guaranteed, Reliable, Secure, Broadcast Networks”, IEEE Network magazine, Nov. 1989, pp. 33-37.

[72] Crowcroft, J., Paliwoda, K., “A Multicast Transport Protocol”, Proc. of ACM Sigcomm’88, pp. 247-256, August 1988.

[73] Ramakrishnan, S., Jain, B., “A Negative Acknoledgement Protocol with Periodic Polling for Multicast over LANs”, Proc. of IEEE INFOCOM’87, pp. 502-511, March 1987.

[74] Hofmann, M., Home page of the Local Group Concept (lGC), http://www.telematik.informatik.uni-karlsruhe.de/~hofmann/LocalGroups.html.

[75] Miller, K., Robertson, K., Tweedly, A., White, M., “StarBurst Multicast Transfer Protocol (MFTP) Specification”, Internet Draft, draft-miller-mftp-spec-02.txt, work in progress, January 1997.

[76] Chiu, D., Hurst, S., Kadansky, M., Wesley, J., “TRAM: A Tree-based Reliable Multicast protocol”, Sun Technical Report SML TR-98-66, Sun Microsystems, July 1998.

[77] Vicisano, L., Crowcroft, J., “One to Many Reliable Bulk Data Transfer on the MBONE”, Third International Workshop on High Performance Protocol Architectures HIPPARCH ‘97, Sweden, June 1997.

[78] Nonnenmacher, J., Biersak, E., Towsley, D., “Parity-based loss recovery for Reliable Multicast Transmission”, ACM/IEEE Transactions on Networking, 1998.

[79] Rubenstein, D., Kurose, J, Towsley, D., “Real-Time Reliable Multicast Using Proactive Forward Error Corection”, NOSSDAV’98.

[80] D. Li and D. R. Cheriton. OTERS (On-Tree Efficient Recovery using Subcasting): A Reliable Multicast Protocol, Proceedings of 6th IEEE International Conference on Network Protocols (ICNP'98). October 1998, Austin, Texas, pp 237-245.

[81] Liu, C.-G., Estrin, D., Shenker, S., and Zhang, L., Local Error Recovery in SRM: Comparison of Two Approaches, USC Technical Report 97-648, January 1997.

[82] Holbrook, H., Cheriton, D., "IP Multicast Channels: EXPRESS Support for Large-Scale Single-Source Applications" Proc. of Sigcomm'99, Boston, MA, August 1999.

[83] Perlman, R., Lee, C-Y., Ballardie, A., Crowcroft, J., Wang, Z., Maufer, T., Diot, C., Thoo, J., Green, M., "A Design for Simple, Low-Overhead Multicast," Internet Draft, draft-perlman-simple-multicast-02.txt, February 1999, work in progress.

[84] "The State of the Internet: Roundtable 4.0," IEEE Specrum, vol.35, No. 10, October 1998.

[85] Papadopoulos, C., Parulkar, G., "Retransmission-based Error Control for Continuous Media Applications," Proceedings of the Sixth International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'96), pp. 5-12 1996.

[86] Floyd, S., and Jacobson, V., "Random Early Detection gateways for Congestion Avoidance," EEE/ACM Transactions on Networking, V.1 N.4, August 1993, pp. 397-413.