A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing

Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang
to appear in IEEE/ACM Transactions on Networking, December 1997

***Abstract*— This paper[1] describes SRM (Scalable Reliable Multicast), a reliable multicast framework for light-weight sessions and application level framing. The algorithms of this framework are efficient, robust, and scale well to both very large networks and very large sessions. The SRM framework has been prototyped in wb, a distributed whiteboard application, which has been used on a global scale with sessions ranging from a few to a few hundred participants. The paper describes the principles that have guided the SRM design, including the IP multicast group delivery model, an end-to-end, receiver-based model of reliability, and the application level framing protocol model. As with unicast communications, the performance of a reliable multicast delivery algorithm depends on the underlying topology and operational environment. We investigate that dependence via analysis and simulation, and demonstrate an adaptive algorithm that uses the results of previous loss recovery events to adapt the control parameters used for future loss recovery. With the adaptive algorithm, our reliable multicast delivery algorithm provides good performance over a wide range of underlying topologies.**

1 Introduction

Several researchers have proposed generic reliable multicast protocols, much as TCP is a generic transport protocol for reliable unicast transmission. In this paper we take a different view: unlike the unicast case where requirements for reliable, sequenced data delivery are fairly general, different multicast applications have widely different requirements for reliability. For example, some applications require that delivery obey a total ordering while many others do not. Some applications have many or all the members sending data while others have only one data source. Some applications have replicated data, for example in an n-redundant file store, so several members are capable of transmitting a data item while for others all data originates at a single source. These differences all affect the design of a reliable multicast protocol. Although one could design a protocol for the worst-case requirements, e.g., guaranteeing totally ordered delivery of replicated data from a large number of sources, such an approach results in substantial overhead for applications with more modest requirements. One cannot make a single reliable multicast delivery scheme that optimally meets the functionality, scalability, and efficiency requirements of all applications.

The weakness of "one size fits all" protocols has long been recognized. In 1990 Clark and Tennenhouse proposed a new protocol model called Application Level Framing (ALF) which explicitly includes an application's semantics in the design of that application's protocol [6]. ALF was later elaborated with a light-weight rendezvous mechanism based on the IP multicast distribution model, and with a notion of receiver-based adaptation for unreliable, real-time applications such as audio and video conferencing. The result, known as Light-Weight Sessions (LWS) [19], has been very successful in the design of wide-area, large-scale, conferencing applications. This paper further evolves the principles of ALF and LWS to add a framework for Scalable Reliable Multicast (SRM).

ALF says that the best way to meet diverse application requirements is to leave as much functionality and flexibility as possible to the application. Therefore SRM is designed to meet only the minimal definition of reliable multicast, i.e., eventual delivery of all the data to all the group members, without enforcing any particular delivery order. We believe that if the need arises, machinery to enforce a particular delivery order can be easily added on top of this reliable delivery service.

It has been argued [36, 34] that a single dynamically configurable protocol should be used to accommodate different application requirements. The ALF argument is even stronger: not only do different applications require different types of error recovery, flow control, and rate control mechanisms, but further, these mechanisms must explicitly account for the structure of the underlying application data itself.

SRM is also heavily based on the group delivery model that is the centerpiece of the IP multicast protocol [8]. In IP multicast, data sources simply send to the group's multicast address (a normal IP address chosen from a reserved range of addresses) without needing any advance knowledge of the group membership. To receive any data sent to the group, receivers simply announce that they are interested (via a "join" message multicast on the local subnet) — no knowledge of the group membership or active senders is required. Each receiver joins and leaves the group individually, without affecting the data transmission

S. Floyd and V. Jacobson are both with the Network Research Group, Lawrence Berkeley Laboratory, Berkeley CA, and S. McCanne is with the University of California, Berkeley, CA (email: floyd, van@ee.lbl.gov, mccanne@eecs.berkeley.edu). S. Floyd, V. Jacobson, and S. McCanne were supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Ching-Gung Liu is with the University of Southern California, Los Angeles, CA (email: charley@carlsbad.usc.edu).

Lixia Zhang is with UCLA, Los Angeles, CA (email: lixia@cs.ucla.edu). Ching-Gung Liu and Lixia Zhang were supported in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contract DABT63-94-C-0073.

[1] An earlier version of this paper appeared in ACM SIGCOMM 95.

to any other member. SRM further enhances the multicast group concept by maximizing information and data sharing among all the members, and strengthens the individuality of membership by making each member responsible for its own correct reception of all the data.

Finally, SRM attempts to follow the core design principles of TCP/IP. First, SRM requires only the basic IP delivery model — best-effort with possible duplication and reordering of packets — and builds reliability on an end-to-end basis. No change or special support is required from the underlying IP network. Second, in a fashion similar to TCP adaptively setting timers or congestion control windows, the algorithms in SRM dynamically adjust their control parameters based on the observed performance within a session. This allows applications using the SRM framework to adapt to a wide range of group sizes, topologies and link bandwidths while maintaining robust and high performance.

Wb, the distributed whiteboard tool designed and implemented by McCanne and Jacobson [17, 23], is the first application based on the SRM framework. In this paper we discuss wb in some detail, to illustrate the use of SRM in a specific application.

The paper proceeds as follows: Section 2 discusses general issues for reliable multicast delivery. Section 3 describes the SRM framework, and discusses the wb instantiation of this framework. Section 4 discusses the performance of SRM in simple topologies such as chains, stars, and bounded-degree trees, and Section 5 presents simulation results from more complex topologies. Section 6 examines the behavior of the loss recovery algorithm in SRM as a function of the timer parameters. Section 7 discusses extensions to the basic reliable multicast framework, such as adaptive algorithms for adjusting the timer parameters and algorithms for local recovery. Section 8 discusses related work on reliable multicast. Section 9 discusses future work on SRM.

2 The design of reliable multicast

2.1 Reliable data delivery: adding the word "multicast"

The problem of reliable unicast data delivery is well understood and a variety of well-tested solutions are available. However, for the reliable transmission of data to a potentially large group of receivers, multicast transmission offers the most promising approach. If a sender were to open N separate unicast TCP connections to N different receivers, then N copies of each packet might have to be sent over links close to the sender, making poor use of the available bandwidth. In addition, the sender would have to keep track of the status of each of the N receivers. Multicast delivery permits a much more efficient use of the available bandwidth, with at most one copy of each packet sent over each link in the absence of dropped packets. In addition, IP multicast allows the sender to send to a group without having to have any knowledge of the group membership. At the same time, adding "multicast" to the data transport problem significantly changes the solution set for reliable delivery.

For example, in any reliable protocol some party must take responsibility for loss detection and recovery. Because of the "fate-sharing" implicit in unicast communication, i.e., the data transmission fails if either of the two ends fails, either the sender or receiver can take on this role. In TCP, the sender times transmissions and keeps retransmitting until an acknowledgment is received. NETBLT [7] uses the opposite model and makes the receiver responsible for all loss detection and recovery. Both approaches have been shown to work well for unicast.

However, if a TCP-style, sender-based approach is applied to multicast distribution, a number of problems occur. First, because data packets trigger acknowledgments (positive or negative) from all the receivers, the sender is subject to the well-known ACK implosion effect [10]. Also, if the sender is responsible for reliable delivery, it must continuously track the changing set of active receivers and the reception state of each. Since the IP multicast model deliberately imposes a level of indirection between senders and receivers (i.e., data is sent to the multicast group, not to the set of receivers), the receiver set may be expensive or impossible to obtain. Finally, the algorithms that are used to adapt to changing network conditions tend to lose their meaning in the case of multicast. E.g., how should the round-trip time estimate for a retransmit timer be computed when there may be several orders of magnitude difference in propagation time to different receivers? What is a congestion window if the delay-bandwidth product to different receivers varies by orders of magnitude? What self-clocking information exists in the ACK stream(s) if some receivers share one bottleneck link and some another?

These problems illustrate that single-point, sender-based control does not adapt or scale well for multicast delivery. Since members of a multicast group have different communication paths and may come and go at any time, the "fate-shared" coupling of sender and receiver in unicast transmissions does not generalize to multicast. Thus it is clear that receiver-based reliability is a far better building block for reliable multicast [33].

Another unicast convention that migrates poorly to multicast has to do with the vocabulary used by the sender and receiver(s) to describe the progress of their communication. A receiver can request a retransmission either in application data units ("sector 5 of file sigcomm-slides.ps") or in terms of the shared communication state ("sequence numbers 2560 to 3071 of this conversation"). Both models have been used successfully (e.g., NFS uses the former and TCP the latter) but, because the use of communication state for naming data allows the protocol to be entirely independent of any application's namespace, it is by far the most popular approach for unicast applications. However, since multicast transmission tends to have much weaker and more diverse state synchronization than does unicast, using shared communication state to name data does not

work well in the multicast case.

For example, if a receiver joins a conversation late and receives sequence numbers 2560 to 3071, it has no idea of what's been missed (since the sender's starting number is arbitrary) and so can neither do anything useful with the data nor make an intelligent request for retransmission. If receivers hear from a sender again after a lengthy network partition, they have no way of knowing whether "2560" is a retransmission of data they received before the partition or is completely new (due to sequence number wrapping during the partition). Thus the "naming in application data units (ADUs)" model works far better for multicast.

Use of this model also has two beneficial side effects. As Clark and Tennenhouse [6] point out, a separate protocol namespace can impose delays and inefficiencies on an application, e.g., TCP will only deliver data in sequence even though a file transfer application might be perfectly happy to receive sectors in any order. The ADU model eliminates this delay and puts the application back in control. Also, since ADU names can be made independent of the sending host, it is possible to use the anonymity of IP multicast to exploit the redundancy of multiple receivers. E.g., if some receiver asks for a retransmit of "sigcomm-slides.ps sector 5", any member who has a copy of the data, not just the original sender, can carry out the retransmission.

2.2 Reliable multicast requirements

While the ALF model says that applications should be actively involved in their communications and that communication should be done in terms of ADUs rather than some generic protocol namespace, we do not claim that every application's protocol must be completely different from every other's or that there can be no shared design or code. A great deal of design commonality is imposed simply because different applications are attempting to solve the same problem: scalable, reliable, multipoint communication over the Internet. As Section 2.1 pointed out, just going from unicast to multicast greatly limits the viable protocol design choices. In addition, experience with the Internet has shown that successful protocols must accommodate many orders of magnitude variation in every possible dimension. While several algorithms meet the constraints of Section 2.1, very few of them continue to work if the delay, bandwidth and user population are all varied by factors of 1000 or more.

In the end we believe the ALF model results in a framework that is then filled in with application specific details. Portions of the SRM framework are completely determined by network dynamics and scaling considerations and apply to any application. For example, the scalable request and repair algorithms described in Sections 3 through 7 are completely generic and apply to a wide variety of reliable multicast applications. Each different application supplies this reliability framework with a namespace to talk about what data has been sent and received; a policy and machinery to determine how much bandwidth is available to the group as a whole; a policy to determine how the available bandwidth should be apportioned between the participants in the group; and a local send policy that a participant uses to arbitrate the different demands on its bandwidth (e.g., locally originated data, requests and responses, etc.). It is the intent of this paper to describe the framework common to scalable, reliable multicast applications. In particular, this paper focuses on reliability rather than on congestion control. We believe that for multicast applications, the congestion control mechanisms will have to take into account application-specific needs and capabilities.

To make the SRM framework concrete, we first describe a widely used application — wb, the LBNL network whiteboard — that has been implemented according to the SRM framework. One component of wb is an application-level reliable multicast protocol that is the precursor to SRM. However, the goal of this paper is not to explore the specifics of wb, but to use wb to illustrate the underlying reliable multicast framework. After mentioning some details of wb's operation that are direct results of the design considerations outlined in Section 2.1, we then factor out the wb specifics to expose the generic SRM framework underneath. The remaining sections of this paper are an exploration of that framework.

2.3 Wb's assumptions about reliable multicast

This section briefly describes wb, a network conferencing tool that provides a distributed whiteboard, and explores some of the assumptions made in wb's use of reliable multicast.

Wb separates the drawing into pages, where a new page can correspond to a new viewgraph in a talk or the clearing of the screen by a member of a meeting. Any member can create a page and any member can draw on any page. There are floor control mechanisms, largely external to wb, that can be used if necessary to control who can create or draw on pages. These can be combined with normal Internet privacy mechanisms (e.g., symmetric-key encryption of all the wb data) to limit participation to a particular group and/or with normal authentication mechanisms (e.g., participants signing their drawing operations via public-key encryption of a cryptographic hash over the data) [24, 18].

Each member is identified by a globally unique identifier, the Source-ID, and each page is identified by a Page-ID consisting of the Source-ID of the initiator of the page and a page number locally unique to that initiator. Each member drawing on the whiteboard produces a stream of drawing operations, or "drawops", that are timestamped and assigned sequence numbers, relative to the sender. Each sequence of drawops is sent with the Page-ID of the relevant page. An example would be a drawop to draw a blue line at a particular set of coordinates on a page.

Wb has no requirement for ordered delivery because most drawing operations are idempotent and are rendered immediately upon receipt; out of order drawops are sorted upon arrival according to their timestamps. Each member's graphics stream is thus independent from that of

other sites. Operations that are not strictly idempotent, such as a "delete" that references an earlier drawop, can be patched after the fact, when the missing data arrives.

The following assumptions are made in wb's reliable multicast design:

- All data has a unique, persistent name.
  This global name consists of the end host's Source-ID and a locally-unique sequence number.
- The name always refers to the same data.
  It is impossible to achieve consistency among different receivers in the face of late arrivals and network partitions if, say, drawop "floyd:5" initially means to draw a blue line and later means to draw a red circle. This does not mean that the drawing can't change, only that drawops must effect the change. E.g., to change a blue line to a red circle, a "delete" drawop for "floyd:5" is sent, then a drawop for the circle is sent.
- Source-ID's are persistent.
  A user will often quit a session and later re-join, obtaining the session's history from the network. By ensuring that Source-ID's are persistent across invocations of the application, the user maintains ownership of any data created before quitting.
- IP multicast datagram delivery is available.
- All participants join the same multicast group; there is no distinction between senders and receivers.

3 The SRM framework

SRM is the reliable multicast framework intended for a range of applications that share wb's assumptions above, including that of IP multicast datagram delivery. One assumption central to SRM is that the data has unique, persistent names. An open research challenge is to design a data naming scheme that reflects the flexibility of ALF yet allows the SRM framework to manipulate names in a generic fashion. A second assumption is that the application naming conventions allow us to impose a hierarchy over the name space. For the rest of this paper, we assume that the data space is subdivided into groups or containers that we call "pages", and that the locally unique name is a simple sequence number with sufficient precision to never wrap. (The term "page" refers to a general concept even though it reflects our whiteboard-biased design.)

Whenever a member generates new data, the data is multicast to the group. Each member of the group is individually responsible for detecting loss, generally by detecting a gap in the sequence space, and requesting retransmission. However, since it is possible that the last object of a sequence is dropped, each member multicasts low-rate, periodic, session messages that announce the highest sequence number received from every member for the current page. In addition to the reception state, the session messages contain timestamps that are used to estimate the distance (in time) from each member to every other (described in Section 3.1).

To prevent the implosion of control packets sent from receivers in a multicast group, receivers in the Xpress Transport Protocol (XTP) design [36] multicast control packets to the entire group. Using the slotting and damping mechanisms in the XTP design, receivers wait for a random time before sending a control packet, and refrain from sending a control packet if they see a control packet from another receiver with the same information. SRM uses similar mechanisms to control the sending of request and repair packets, with the addition that in the SRM design, the random delay before sending a request or repair packet is a function of that member's distance in seconds from the node that triggered the request or repair. The timer calculations are described in detail in Section 3.2.

As with the original data, repair requests and retransmissions are always multicast to the whole group. Thus, although a number of hosts may all miss the same packet, a host close to the point of failure is likely to timeout first and multicast the request. Other hosts that are also missing the data hear that request and suppress their own request. Any host that has a copy of the requested data can answer a request. It will set a repair timer, and multicast the repair when the timer goes off. Other hosts that had the data and scheduled repairs will cancel their repair timers when they hear the multicast from the first host. This does not require that all session members keep all of the data all of the time; reliable data delivery is ensured as long as each data item is available from at least one member. Ideally, a lost packet triggers only a single request from a host just downstream of the point of failure and a single repair from a host just upstream of the point of failure. Section 5 explores in more detail the number of requests and repairs in different topologies.

3.1 Session messages

In SRM, each member multicasts periodic session messages that report the sequence number state for active sources. Session messages for reliable multicast [10] have been previously proposed to enable receivers to detect the loss of the last packet in a burst, and to enable the sender to monitor the status of receivers. Members can also use session messages in SRM to determine the current participants of the session. The average bandwidth consumed by session messages is limited to a small fraction (e.g., 5%) of the aggregate data bandwidth, whether pre-allocated by a reservation protocol or measured adaptively by a congestion control algorithm. SRM members use the algorithm developed for vat and described in [32] for dynamically adjusting the generation rate of session messages in proportion to the multicast group size.

In a large, long-lived session, the state would become unmanageable if each receiver had to report the sequence numbers of everyone who had ever sent data to the group. To prevent this explosion, we impose hierarchy on the data by partitioning the state space into 'pages'. Each member only reports the state of the page it is currently viewing. A receiver browsing over previous pages may issue *page*

requests to learn the sequence number state for that page. If a receiver joins late, it may issue page requests to learn the existence of previous pages. We omit the details of the page state recovery protocol as it is almost identical to the repair request/response protocol for data.

In addition to state exchange, receivers use the session messages to estimate the one-way distance between nodes. All packets for that group, including session packets, include a Source-ID and a timestamp. The session packet timestamps are used to estimate the host-to-host distances needed by the repair algorithm.

The timestamps are used in a highly simplified version of the NTP time synchronization algorithm [26]. Assume that host A sends a session packet P_1 at time t_1 and host B receives P_1 at time t_2. At some later time, t_3, host B generates a session packet P_2, marked with (t_1, Δ) where $\Delta = t_3 - t_2$ (time t_1 is included in P_2 to make the algorithm robust to lost session packets). Upon receiving P_2 at time t_4, host A can estimate the latency from host B to host A as $(t_4 - t_1 - \Delta)/2$, or equivalently, as $((t_4 - t_3) + (t_2 - t_1))/2$. Note that while this estimate does not assume synchronized clocks, it does assume that paths are roughly symmetric. We have not yet explored the performance of these algorithms in topologies with strong asymmetry in the one-way delays of forward and reverse paths.

3.2 Loss recovery

This section describes SRM's loss recovery algorithm, which provides the foundation for reliable delivery. Section 7.1 describes a modified version of this algorithm with an adaptive adjustment of the timer parameters. Section 7.2 discusses the local recovery algorithms that would be a critical component of SRM for efficient operation in large multicast groups in a congested environment.

In SRM, members who detect a loss wait a random time and then multicast their repair request, to suppress requests from other members sharing that loss. These *repair requests* differ from traditional negative acknowledgements (NACKs) in two respects: they are not addressed to a specific sender, and they request data by its unique, persistent name. When a host A detects a loss, it schedules a repair request for a random time in the future. When the request timer expires, host A multicasts a request for the missing data, and doubles the request timer to wait for the repair.

In SRM, the interval over which the request timer is set is a function of the member's estimated distance to the source of the packet. The request timer is chosen from the uniform distribution on $[C_1 d_{S,A}, (C_1 + C_2) d_{S,A}]$ seconds, where $d_{S,A}$ is host A's estimate of the one-way delay to the original source S of the missing data. The numbers C_1 and C_2 are parameters of the request algorithm that are discussed at length later in the paper.

If host A receives a request for the missing data before its own request timer for that data expires, then host A does a (random) exponential backoff, and resets its request timer.[2] That is, if the current timer had been chosen from the uniform distribution on

$$2^i [C_1 d_{S,A}, (C_1 + C_2) d_{S,A}],$$

then the backed-off timer is randomly chosen from the uniform distribution on

$$2^{i+1} [C_1 d_{S,A}, (C_1 + C_2) d_{S,A}].$$

When some other host B (where B may be the original source S) receives a request from A that host B is capable of answering, host B sets a repair timer to a value from the uniform distribution on

$$[D_1 d_{A,B}, (D_1 + D_2) d_{A,B}]$$

seconds, where $d_{A,B}$ is host B's estimate of the one-way delay to host A, and the numbers D_1 and D_2 are parameters of the repair algorithm discussed later in the paper. If host B receives a repair for the missing data before its repair timer expires, then host B cancels its repair timer. Otherwise, when host B's repair timer expires host B multicasts the repair. In keeping with the philosophy that the receiver is responsible for ensuring its own correct reception of the data, host B does not verify whether host A actually receives the repair.

Due to the probabilistic nature of these algorithms, it is not unusual for a dropped packet to be followed by more than one request. When two or more hosts generate a request for the same data at roughly the same time, we have redundant control traffic (i.e., wasted bandwidth) and the colliding participants should increase the spread in their retransmission distribution to avoid similar collisions in the future.

Because there can be more than one request, a host could receive a duplicate request immediately after sending a repair, or immediately after receiving a repair in response to its own earlier request. In order to prevent duplicate requests from triggering a responding set of duplicate repairs, host B ignores requests for data D for $3\, d_{S,B}$ seconds after sending or receiving a repair for that data, where host S is either the original source of data D or the source of the first request.

3.3 Congestion control

The simplest congestion control mechanism for SRM would be for all members of the multicast group to assume a fixed bandwidth constraint over the aggregate session. This would be appropriate, for example, if members of the multicast group used an out-of-band mechanism (e.g., explicit bandwidth reservations, or the informal, consensus-based

[2] Some care is required in deciding when to back-off an already backed-off timer. In our simulator, we use a heuristic to detect requests that belong to the same iteration of loss recovery. When member A backs-off the request timer, then member A sets an *ignore-backoff* variable to a time halfway between the current time and the expiration time, and ignores additional duplicate requests until *ignore-backoff* time. Requests received before the ignore-backoff time are assumed to belong to the same iteration of the loss recovery as the request that resulted in the most recent backoff. A request received after the ignore-backoff time is assumed to belong to the next iteration, and causes member A to again back-off its request timer.

procedures of the current Mbone) to verify bandwidth availability. However, different congestion control mechanisms are likely to be required for different applications and different contexts. Congestion control mechanisms for SRM are discussed further in Section 9.3.

Because data represents idempotent operations, loss recovery can proceed independently from the transmission of new data. Similarly, recovery for losses from two different sources can also proceed independently. Since transmission bandwidth is often limited, a single transmission rate is allocated to control the throughput across all these different modes of operation, while the application determines the order of packet transmission according to their relative importance.

3.4 Network partitioning and other concerns

Because SRM relies on the underlying concept of an IP multicast group, where members can arrive and depart independently, SRM does not distinguish a network partition from a normal departure of members from the multicast session. During a partition, members can continue to send data in the connected components of the partitions. After recovery all data will still have unique names and the repair mechanism will distribute any new state throughout the entire group.

For applications that may require partial or total data ordering, the SRM framework could be used to reliably deliver the data to all group members, and a partial or total ordering protocol could be built on top that is specifically tailored to the ordering needs of that application. Ordering is further complicated by disagreements about how the ordering itself should be defined: Cheriton and Skeen [5] have argued (and Birman [1] has rebutted) that for applications with ordering requirements, preserving the ordering of messages as they appear in the network can be an expensive and inadequate substitute for preserving the "semantic ordering" of the messages appropriate for the application.

Potential applications for SRM other than wb, including routing protocol updates, Usenet news, and adaptive web caches, are discussed briefly in [12, 13].

3.5 Wb's instantiation of SRM

This section describes both the design and the current state of the implementation of reliable multicast for wb. As discussed below, the rate-contol mechanism and the estimates of one-way delay are key aspects of the design that are not yet included in the current implementation of wb.

In the present implementation of wb (version 1.59), members set a request timer to a random value from the interval $[d, 7d]$, where d is set to a fixed value of 30 msec. The estimation of the distance to other members has not yet been included in the current implementation. Similarly, after receiving a request members set a repair timer to a random value from the interval $[d_1, 2d_1]$. For the original source of the data, d_1 is set to a fixed value of 100 msec., and for other members d_1 is set to 200 msec. These fixed values for d and d_1 were chosen after examinations of traces taken over several typical wide-area wb sessions. The current values for d and d_1 are sufficiently large to ensure that there is generally only one request and one repair. When the original source of the data is still on-line, the repair generally comes from that original source.

The current implementation of wb relies on the informal, consensus-based "admissions-control procedure" of the current Mbone. The congestion control mechanism in the design for wb assumes a fixed maximum bandwidth allocation for each session. In this design, each wb session would have a sender bandwidth limit advertised as part of the session announcement, and individual members would use a token bucket rate limiter to enforce this peak rate on transmissions. As of the writing of this paper, this rate control mechanism has not yet been added to the wb implementation. In practice, wb sessions generally use considerably less average bandwidth than their accompanying audio sessions. However, the need for this rate control can at times be made painfully obvious, for example, when new members join a session and ask for back history.

One application-specific issue concerns the relative priorities between sending new data, requests, and repairs. When a member of a wb session is able to send a packet, the highest priority goes to requests or repairs for the current page, middle priority to new data, and lowest priority to requests or repairs for previous pages.

One issue that has been made obvious from implementation experience has been the persistence of the data. Wb does not necessarily store all of the data on backup storage on a disk; data for current pages is kept only in memory. If data somehow becomes corrupt — either due to internal application bugs or because of external system failures — it can spread like a virus throughout the wb session. When the corrupted data is used to answer repair requests, the corrupted data is distributed throughout the multicast group, and persists for the life of the session. To avoid this, each piece of data can be accompanied by a tag that not only authenticates the source of the data but also verifies its integrity.

4 Request/repair algorithms for simple topologies

We now turn to a more detailed investigation of the loss recovery algorithms in SRM. Because multiple hosts may detect the same losses, and multiple hosts may attempt to handle the same repair request, the goal of the request/repair timer algorithms is to de-synchronize host actions to keep the number of duplicates low. Among hosts that have diverse delays to other hosts in the same group, this difference in delay can be used to differentiate hosts; for hosts that have similar delays to reach others, we can only rely on randomization to de-synchronize their actions.

This section discusses a few simple, yet representative, topologies, namely chain, star, and tree topologies, to pro-

vide a foundation for understanding the loss recovery algorithms in more complex environments. For a chain the essential feature of a loss recovery algorithm is that the timer value is a function of distance. For a star topology the essential feature of the loss recovery algorithm is the randomization used to reduce implosion. Request/repair algorithms in a tree combine both the randomization and the setting of the timer as a function of distance. This section shows that the performance of the loss recovery algorithms depends on the underlying network topology.

4.1 Chains

Figure 1 shows a chain topology where all nodes in the chain are members of the multicast session. Each node in the underlying multicast tree has degree at most two. The chain is an extreme topology where a simple deterministic loss recovery algorithm suffices. In this section we assume that the timer parameters C_1 and D_1 are set to 1, and that C_2 and D_2 are set to 0. This results in request timers set deterministically to $d_{S,A}$, and repair timers set to $d_{A,B}$.

For the chain, as in most of the other scenarios in this paper, link distance and delay are both normalized. We assume that packets take one unit of time to travel each link, i.e., all links have distance of 1.

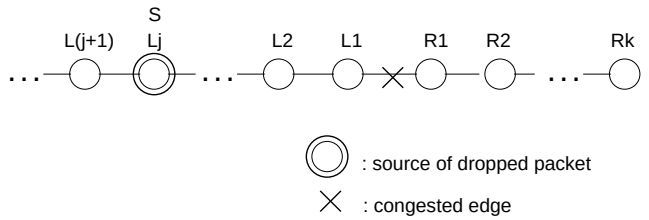

Figure 1: A chain topology.

In Figure 1 the nodes in the chain are labeled as either to the right or to the left of the congested link. Assume that source L_j multicasts a packet that is subsequently dropped on link $(L_1,\ R_1)$, and that the second packet sent from source L_j is not dropped. We call the edge that dropped the packet, whether due to congestion or to other problems, the *congested* link. Let the right-hand nodes each detect the failure when they receive the second packet from L_j.

Let node R_1 first detect the loss at time t, and let each link have distance 1. Then node R_1 multicasts a request at time $t + j$. Node L_1 receives the request at time $t + j + 1$ and multicasts a repair at time $t + j + 2$. Node R_k receives the repair at time $t + k + j + 2$.

Note that all nodes to the right of node R_1 receive the request from R_1 before their own request timers expire. We call this *deterministic suppression*. The reader can verify that, due to deterministic suppression, there will be only one request and one repair. For example, node R_k detects the loss at time $t + k - 1$, sets its request timer for time $(t + k - 1) + (j + k - 1) = t + 2k + j - 2$, and receives the request from node R_1 at time $(t + j) + (k - 1)$, well before its own request timer expires.

Had the loss repair been done by unicast, i.e. node R_k sent a unicast request to the source L_j as soon as it detected the failure and L_j sent a unicast repair to R_k as soon as it received the request, node R_k would not receive the repair until time $t + 2j + 3k$. Thus, with a chain and with a simple deterministic loss recovery algorithm, the furthest node receives the repair sooner than it would if it had to rely on its own unicast communication with the original source, because both the request and the repair come from nodes immediately adjacent to the congested link.

4.2 Stars

For the star topology in Figure 2 we assume that all links are identical and that the center node is not a member of the multicast group. For a star topology, setting the request timer as a function of the distance from the source is not an essential feature, as all nodes detect a loss at exactly the same time. Instead, the essential feature of the loss recovery algorithm in a star is the randomization used to reduce implosion; we call this *probabilistic suppression*.

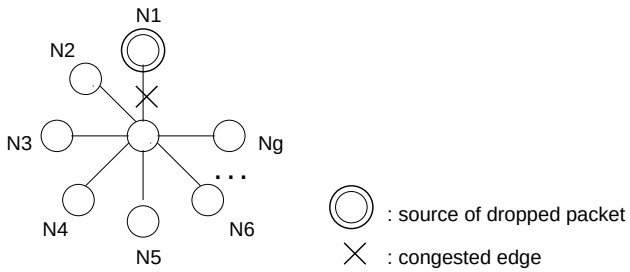

Figure 2: A star topology.

For the star topology in Figure 2 assume that the first packet from node N_1 is dropped on the adjacent link. There are G members of the multicast session, and the other members detect the loss at exactly the same time. For the discussion of this topology we assume that the timer parameters C_1 and D_1 are set to 0; because all nodes detect losses and receive requests at the same time, C_1 and D_1 are not needed to amplify differences in delay. For a star topology, the only benefits in setting C_1 greater than 0 are to avoid unnecessary requests from out-of-order packets and to ensure a minimum delay when a request timer is backed-off.

If C_2 is at most 1, then there will always be $G - 1$ requests. Increasing C_2 reduces the expected number of requests but increases the expected time until the first request is sent. For $C_2 > 1$, the expected number of requests is roughly $1 + (G - 2)/C_2$, and the expected delay until the first timer expires is $2C_2/G$ seconds (where one unit of time is one second).[3] For example, if C_2 is set to $\sqrt{G}$, then the expected number of requests is roughly $\sqrt{G}$, and the expected delay until the first timer expires is $2/\sqrt{G}$ seconds.

[3]The $G - 1$ nodes all detect the failure at the same time, and all set their timers to a uniform value in an interval of width $2C_2$. If the first timer expires at time t, then the other $G - 2$ receivers receive that first request at time $t + 2$. So the expected number of duplicate requests is equal to the expected number of timers that expire in the interval $[t,\ t+2]$.

Note that if N_2 was the source of the dropped packet, then N_1 would be the only node to send a request, and the other session members would receive the request at the same time. The same remarks as above would then apply to D_2 with respect to repairs.

4.3 Bounded-degree trees

The loss recovery performance in a tree topology uses both the deterministic suppression described for chain topologies and the probabilistic suppression described for star topologies. Consider a network topology of a bounded-degree tree with N nodes where interior nodes have degree p. A tree topology combines aspects of both chains and stars. The timer value should be a function of distance, to enable requests and repairs to suppress request and repair timers at nodes further down in the tree. In addition, randomization is needed to reduce request/repair implosion from nodes that are at an equal distance from the source (of the dropped packet, or of the first request). In this section, we show that the behavior of the request algorithms in a tree topology depends principally on the distance of the sender from the congested link, and on the ratio between the timer parameters C_2 and C_1.

We assume that node S in the tree is the source of the dropped packet, and that link (B,A) drops a packet from source S. We call nodes on the source's side of the congested link (including node B) *good* nodes, and nodes on the other side of the congested link (including node A) *bad* nodes. Node A detects the dropped packet at time t, when it receives the next packet from node S. We designate node A as a *level-0* node, and we call a bad node a *level-i* node if it is at distance i from node A.

Assume that the source of the dropped packet is at distance j from node A. Node A's request timer expires at time

$$t + C_1 j + U_1[C_2]j,$$

where $U[C_2]$ denotes a uniform random variable between 0 and C_2. Assuming that node A's request is not suppressed, a level-i node receives node A's request at time

$$t + i + C_1 j + U_1[C_2]j.$$

Node B receives node A's repair request at time

$$t + 1 + C_1 j + U_1[C_2]j.$$

A bad level-i node detects the loss at time $t+i$, and such a node's request timer expires at some time

$$t + i + C_1(i + j) + U_2[C_2](i + j).$$

Note that regardless of the values of $U_1[C_2]$ and $U_2[C_2]$, a level-i node receives node A's request by time $t+i+C_1 j+C_2 j$, and a level-i node's request timer expires no sooner than $t + i + C_1(i + j)$. If

$$t + i + C_1 j + C_2 j \leq t + i + C_1(i + j),$$

that is, if

$$\frac{C_2}{C_1} j \leq i,$$

then the level-i node's request timer will always be suppressed by the request from the level-0 node. Thus, the smaller the ratio C_2/C_1, the fewer the number of levels that could be involved in duplicate requests. This relation also demonstrates why the number of duplicate requests or repairs is smaller when the source (of the dropped packet, or of the request) is close to the congested link.

Note that the parameter C_1 serves two different functions. A smaller value for C_1 gives a smaller delay for node B to receive the first request. At the same time, for nodes further away from the congested link, a larger value for C_1 contributes to suppressing additional levels of request timers. A similar tradeoff occurs with the parameter C_2. A smaller value for C_2 gives a smaller delay for node B to receive the first repair request. At the same time, for topologies such as star topologies, a larger value for C_2 helps to prevent duplicate requests from session members at the same distance from the congested link. Similar remarks apply to the functions of D_1 and D_2 in the repair timer algorithm.

5 Simulations of the request and repair algorithms

For a given underlying network, set of session members, session sources, and congested link, it should be feasible to analyze the behavior of the repair and request algorithms with fixed timer parameters C_1, C_2, D_1, and D_2. However, we are interested in the repair and request algorithms across a wide range of topologies and scenarios. We use simulations to examine the performance of the loss recovery algorithms for individual packet drops in random and bounded-degree trees. We do not claim to be presenting realistic topologies or typical patterns of packet loss.

We define the *density* of a session as the fraction of nodes in the underlying network that are members of the multicast session. The simulations in this section show that the loss recovery algorithms with fixed timer parameters perform well in a random or bounded-degree tree for *dense* sessions, where many of the nodes in the underlying tree are members of the multicast session. The loss recovery algorithms perform somewhat less well for a *sparse* session, where the session size is small relative to the size of the underlying network, and the members might be scattered throughout the net. This motivates the development on the adaptive loss recovery algorithm in Section 7.1, where the timer parameters C_1, C_2, D_1, and D_2 are adjusted in response to past performance.

In these simulations the fixed timer parameters are set as follows: $C_1, C_2 = 2$, and $D_1, D_2 = \log_{10} G$, where G is the number of members in the same multicast session. The choice of $\log_{10} G$ for D_1 and D_2 is not critical, but gives slightly better performance than $D_1, D_2 = 1$ for large G.

Each simulation constructs either a random tree or a bounded degree tree with N nodes as the network topology. Next, G of the N nodes are randomly chosen to be session members; these session members are not necessarily leaf nodes in the network topology. Finally, a source S is randomly chosen from the G session members.

We assume that messages are multicast to members of the multicast group along a shortest-path tree from the source of the message. In each simulation we randomly choose a link L on the shortest-path tree from source S to the G members of the multicast group. We assume that the first packet from source S is dropped by link L, and that receivers detect this loss when they receive the subsequent packet from source S.

[13] discusses the tools that we used to verify that our simulator is correctly implementing the loss recovery algorithms. The simulator that we used for the simulations in this paper is not publically available. However, much of the same functionality has been implemented in the ns-2 simulator [28]. Further progress will be reported on the SRM web page [38].

5.1 Simulations on random trees

In this section we consider networks of random labeled trees, where all nodes in the networks are session members. The next section considers large networks with nodes of degree four, where only a fraction of the nodes are members of the multicast group.

For the simulations on random labeled trees of N nodes, the random labeled trees are constructed according to the labeling algorithm in [30, p.99]. These trees have unbounded degree, but for large N, the probability that a particular vertex in a random labeled tree has degree at most four approaches (approximately) 0.98 [30, p.114]. Figure 3 shows simulations of the loss recovery algorithm for this case, where all N nodes in the tree are members of the multicast session (that is, $G = N$). For each graph the x-axis shows the session size G; twenty simulations were run for each value of G. For each simulation, a new random tree was constructed, and session members, a source, and a congested link were randomly chosen. Each simulation is represented by a *jittered* dot[4], and the median from the twenty simulations is shown by a solid line. The two dotted lines mark the upper and lower quartiles; thus, the results from half of the simulations lie between the two dotted lines. While there are not enough simulations to make accurate predictions of the behavior of the loss recovery algorithms, the simulations do illustrate the loss recovery algorithms under a range of circumstances.

The top two graphs in Figure 3 show the number of requests and repairs to recover from a single loss. In these graphs the median, lower quartile, and upper quartile lines are the same; the y-axis was chosen for an easy visual comparison with other simulations later in the paper.

[4]A jittered dot is a dot for which some small random jitter has been added to the x and y coordinates. In this way, the reader can differentiate between a single dot, and multiple dots all with the same coordinates.

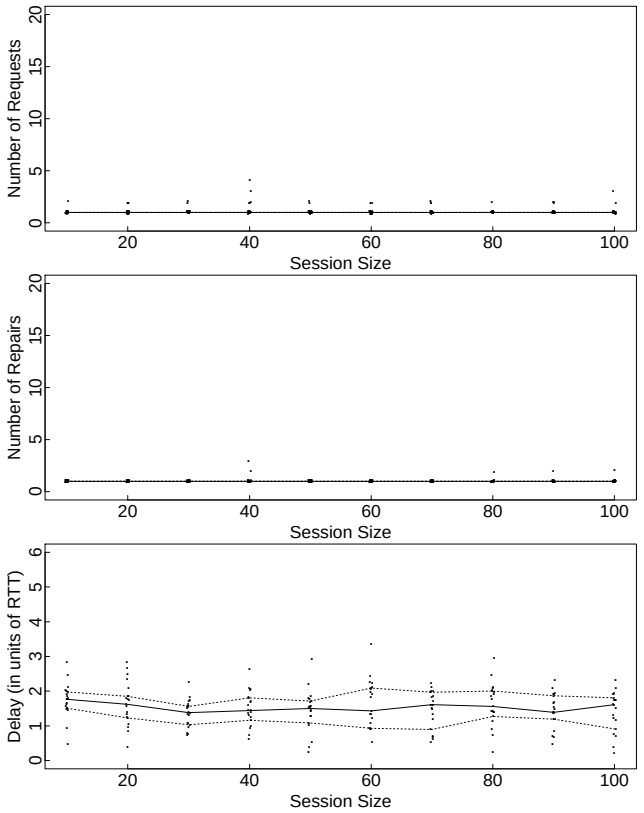

Figure 3: Random trees with a random congested link and a single packet loss, where all nodes are members of the multicast session.

For each member affected by the loss, we define the *loss recovery delay* as the time from when the member first detects the loss until the member first receives a repair. For each simulation, there is a dot in the bottom graph in Figure 3 showing the loss recovery delay for the last member of the multicast session to receive the repair. This loss recovery delay is given as a multiple of the RTT, the roundtrip time from that member to the original source of the dropped packet. While this member has the largest loss recovery delay in absolute terms, this member generally does not have the largest delay when expressed in units of its own RTT.

Note that with unicast communications the ratio of loss recovery delay to RTT is at least one. For a unicast receiver that detects a packet loss by waiting for a retransmit timer to time out, the typical ratio of delay to RTT is closer to 2. With multicast loss recovery algorithms the ratio of delay to RTT can be less than one, because the request and repair could each come from a node close to the point of failure.

Figure 3 shows that the repair/request algorithm with fixed timer parameters works well for a tree topology where all nodes of the tree are members of the multicast session. There is usually only one request and one repair. (Some lack of symmetry results from the fact that the original source of the dropped packet might be far from the point of failure, while the first request comes from a node close to the point of failure.) The average recovery delay for

the farthest node is less than 2 RTT, competitive with the average delay available from a unicast algorithm such as TCP. The results are similar in simulations where the congested link is chosen adjacent to the source of the dropped packet, and for simulations on a bounded-degree tree of size $N = G$ where interior nodes have degree four. (We do not claim that this is the average degree for a router in the Internet, in the current Mbone, or in the likely multicast backbone of the foreseeable future. From looking at a map of the current Mbone topology, choosing a degree of four seemed as reasonable a choice as any other that we might have made.)

5.2 Simulations on large bounded-degree trees

The loss recovery algorithms with fixed timer parameters perform less well for a sparse session in a large bounded-degree tree. The underlying topology for the simulations in this section is a balanced bounded-degree tree of $N = 1000$ nodes, with interior nodes of degree four. In these simulations the session size G is significantly less than N. For a session that is sparse relative to the underlying network, the nodes close to the congested link might not be members of the session.

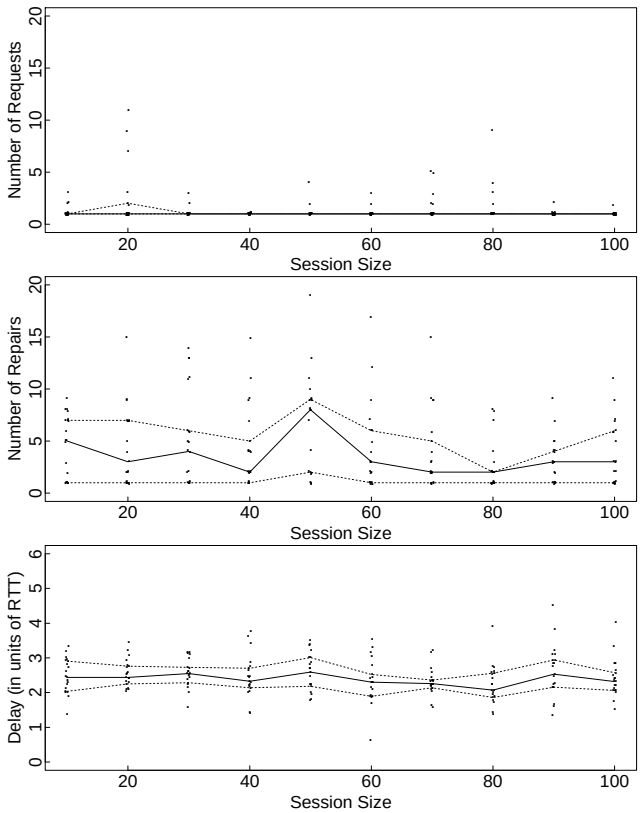

Figure 4: Bounded-degree tree, degree 4, 1000 nodes, with a random congested link.

As Figure 4 shows, the average number of repairs for each loss is somewhat high. In simulations shown in [13] where the congested link is always adjacent to the source, the number of repairs is low but the average number of requests is high.

The performance of the loss recovery algorithm on a range of topologies is shown in [13]. These include topologies where each of the N nodes in the underlying network is a router with an adjacent Ethernet with 5 workstations, point-to-point topologies where the edges have a range of propagation delays, and topologies where the underlying network is more dense that a tree. None of these variations that we have explored have significantly affected the performance of the loss recovery algorithms with fixed timer parameters.

6 Exploring the parameter space

As the previous section showed, a particular set of values for the timer parameters C_1, C_2, D_1, and D_2 that performs well in one scenario might not perform well in another scenario. In this section we choose a few simple topologies, and explore the behavior of the request/repair algorithms as a function of the request timer parameter C_2. In the following section we discuss adaptive algorithms where the timer parameters are adjusted as a function of the past performance of the loss recovery algorithms.

The results in this section can be briefly summarized as follows. The only simulations in this section that give unacceptably large numbers of requests are those with small values for C_2 on stars or for sparse sessions on trees. For these scenarios, increasing C_2 reduces the number of duplicate requests, accompanied by moderate increases in the loss recovery delay. For a star topology, there is a clear tradeoff between the delay and the number of duplicates. In contrast, with a chain topology, setting C_2 to zero gives the optimal behavior both in terms of delay and in the number of duplicates. For a dense session in a tree topology, a small value for C_2 gives good performance in terms of both delay and duplicates.

For the simulations in this section, C_1 is set to 2. As Section 4.1 showed, for a chain with a deterministic loss recovery algorithm, it is sufficient to set C_1 to 1. However, for a chain with a randomized loss recovery algorithm, a higher value of C_1 is needed to ensure that members further from the congested link receive a request before their own request timer expires.

Figure 5 shows the tradeoffs between delay and duplicates in a star topology of size 100, where the congested link is adjacent to the source of the dropped packet. We define the *request delay* for a session member as the delay from when the request timer is set until a request was either sent by that member or received from another member. The top graph in Figure 5 contains a dot for each integer value of C_2 from 0 to 100, for the star topology described in Section 4.2. For each dot, the x-coordinate is the expected request delay for that value of C_2, and the y-coordinate is the expected number of requests.

More precisely, the x-coordinate is given by the expected request delay for the bad member closest to the source of

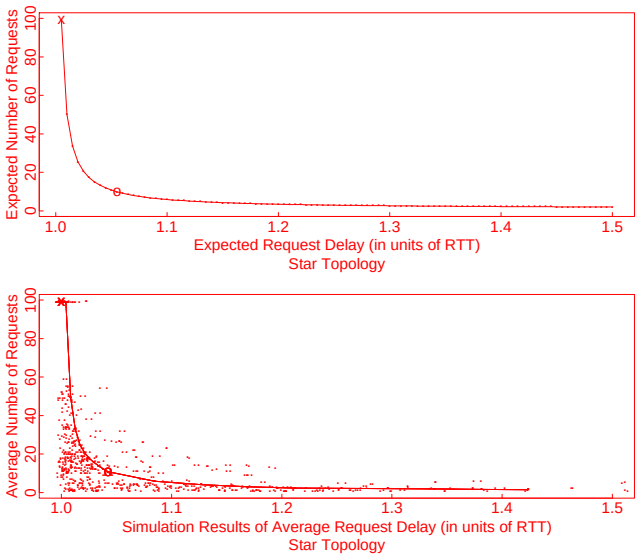

Figure 5: Tradeoff between delay and duplicates in a star topology.

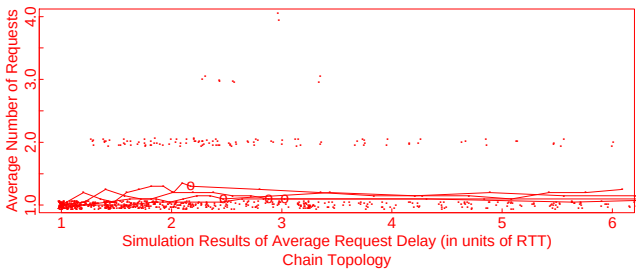

Figure 6: Tradeoff between delay and duplicates in a chain topology.

the dropped packet, expressed as a multiple of the roundtrip time from that member to the source of the dropped packet. When there is not a unique bad member at the minimum distance from the source, as in a star topology, then the x-axis shows the expected smallest request delay from those members at the minimum distance from the source. For a star topology this is the request delay for that member whose request timer expires first.

From the heuristic analysis in Section 4.2, the expected request delay (in units of the RTT of $2D$) is as follows:

$$\frac{C_1 D + C_2 D/G}{2D}$$
$$= C_1/2 + C_2/(2G),$$

where D is the distance in seconds from the source to a session member. From Section 4.2, the expected number of requests is estimated as $1 + (G-2)/C_2$. The "x" in Figure 5 shows the results for $C_2 = 0$, and the circle shows the results for $C_2 = 10$. Thus the top graph of Figure 5 shows that increasing C_2 in a star topology increases the expected request delay slightly while significantly decreasing the expected number of requests.

The bottom graph in Figure 5 shows the results from simulations, which concur with the analytical results in the top graph. For each integer value of C_2 from 0 to 100, twenty simulations are run, and the request delay and total number of requests is calculated for each simulation. Each simulation is represented by a jittered dot, and the line shows the average for each value of C_2. For example, for C_2 set to one hundred the average number of requests is 1.5 and the average request delay, as a multiple of the RTT, is 1.42. The minimum request delay of 1 comes from the fixed value of 2 for request parameter C_1.

These results generally concur with those of [31], which investigates the relative benefits of using unicast or multicast NACKs. La Porta and Schwartz [31] conclude that for a scenario similar to our star topology, where a message sent by any member is received by all other members exactly r seconds later, and for a multicast group with ten members, the random interval over which NACK timers were set would have to be at least 10 times r for the multicasting of NACKs to result in bandwidth savings over a scheme of unicasting NACKs to the source. La Porta and Schwartz [31] conclude that unicasting NACKs would be desirable in some scenarios, but for multicast groups that could have hundreds of members, and for multicast groups where the receivers were somewhat tolerant of delay, multicasting NACKs would be quite effective in reducing the unnecessary use of bandwidth.

Figure 6 shows the results from the chain topology discussed in Section 4.1. For a chain, with C_2 set to zero there will be exactly one request, with request delay $C_1/(2D)$. Increasing C_2 can increase both the expected request delay and the expected number of duplicates. The four lines in Figure 6 show the results for a chain topology with a failed edge 1, 2, 5, or 10 hops, respectively, from the source of the dropped packet. For the simulations with a failed edge one hop from the source, the individual simulations are shown by a dot. For each scenario C_2 ranges from 0 to 10 in increments of 1, and then from 10 to 100 in increments of 10. While increasing C_2 can increase the number of duplicates, the magnitude of the increase is quite small.

Figures 7 and 8 show the results for a range of tree topologies. Each line shows the results for a particular fixed scenario, as C_2 varies from 0 to 100. In all of the scenarios the session size is at least 100. In each graph, the lines represent scenarios that differ only in the number of hops between the source and the failed edge. The four lines represent scenarios with failed edges that are one, two, three, or four hops, respectively, from the source of the dropped packet. For all of the topologies, the failed edge closest to the source gives the line with the worst-case number of duplicate requests. For this line, the individual simulations are each shown by a jittered dot. The graphs are sized for easy comparisons, and do not necessarily show all of the dots.

As an example, the top graph in Figure 7 shows the results for trees of density 1. For each of the lines the average number of duplicates is minimized for $C_2 = 0$, and maximized for an intermediate value of C_2.

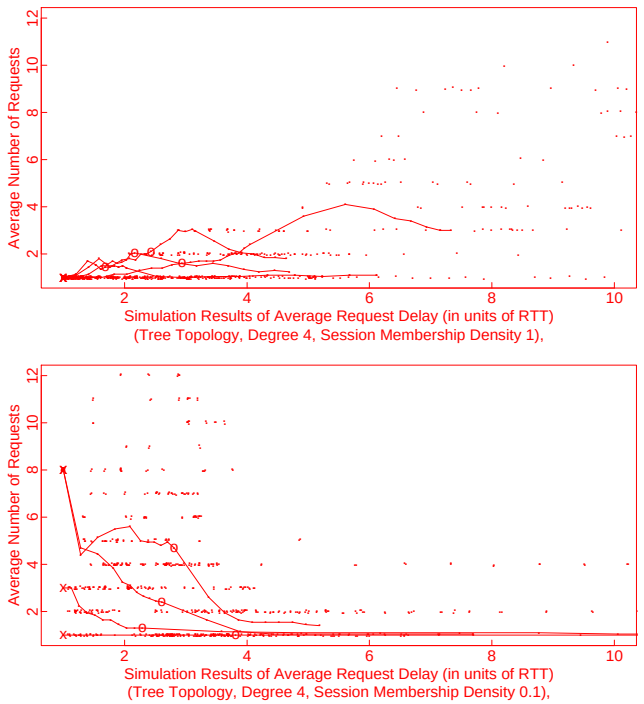

Figure 7: Tradeoff between delay and duplicates for dense sessions in tree topologies.

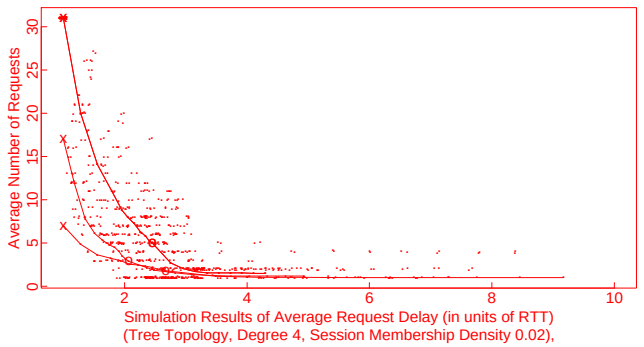

Figure 8: Tradeoff between delay and duplicates for sparse sessions in a tree topology.

7 Extending the basic approach

7.1 Adaptive adjustment of random timer algorithms

The results in the previous section suggest that the SRM loss recovery algorithms with fixed timer parameters give acceptable performance for sessions willing to tolerate a small number of duplicate requests and repairs and willing to accept a moderate request and repair delay (in terms of the roundtrip times of the underlying multicast group). However, there is not a single setting for the timer parameters that gives optimal performance for all topologies, session memberships, and loss patterns. For applications where it is desirable to optimize the tradeoff between delay and the number of duplicate requests and repairs, an adaptive algorithm can be used that adjusts the timer parameters C_1, C_2, D_1, and D_2 in response to the past behavior of the loss recovery algorithms.

In this section we describe an adaptive algorithm that adjusts the timer parameters as a function of both the delay and of the number of duplicate requests and repairs in recent loss recovery exchanges. A related strategy to minimize the number of duplicate requests is to rely on deterministic suppression, with members closest to the point of failure sending requests first. The rest of Section VII-A describes the adaptive algorithm for adjusting the timer parameters in some detail. Section VII-B continues with a discussion of local recovery mechanisms.

One mechanism for encouraging deterministic suppression is for members to reduce C_1 after they send a request. Because members who frequently send requests are likely to also be members who are close to the point of failure, reducing C_1 for those members aids the deterministic suppression. In a star topology, where otherwise the loss recovery mechanisms rely on probabilistic suppression, reducing C_1 in this fashion helps to break symmetry, encouraging certain members to continue sending requests early.

A second mechanism for encouraging deterministic suppression is for members who have sent requests to reduce C_2 if they have received duplicate requests from members significantly further from the source of the failed packet. This mechanism for requests requires that requests include the requestor's estimated distance from the original source of the requested packet. The corresponding mechanism for replies requires that replies include the replier's estimated distance from the source of the request.

```
After sending a request:
   decrease start of req. timer interval
Before each new request timer is set:
   if requests sent in prev. rounds, and any
   dup. requests were from further away:
      decrease request timer interval
   else if ave. dup. requests high:
      increase request timer interval
   else if ave. dup. requests low
   and ave. req. delay too high:
      decrease request timer interval
```

Figure 9: Dynamic adjustment algorithm for request timer interval.

Figure 9 gives the outline of the dynamic adjustment algorithm for adjusting the request timer parameters. A corresponding algorithm applies for adjusting the reply timer parameters. This adaptive algorithm combines the general adaptation performed by all members when they set a request timer with more specific adaptations performed only by members who have recently sent requests. A member determines if the average number of duplicate requests is "too high" by comparing the observed average to a predefined threshold; in this paper the predefined threshold is one duplicate request. If the average number of duplicate requests is too high, then the adaptive algorithm increases the request timer interval. Alternately, if the average num-

ber of duplicates is okay but the average delay in sending a request is too high, then the adaptive algorithm decreases the request timer interval. In this fashion the algorithm can adapt the timer parameters not only to fit the generally-fixed underlying topology, but also to fit a changing session membership and pattern of congestion.

First we describe how a session member measures the average delay and number of duplicate requests in previous loss recovery rounds in which that member has been a participant. A *request period* begins when a member first detects a loss and sets a request timer, and ends when that member detects a subsequent loss and begins a new request period. The variable *dup_req* keeps count of the number of duplicate requests received during one request period; these could be duplicates of the most recent request or of some previous request, but do not include requests for data for which that member never set a request timer. At the end of each request period, the member updates *ave_dup_req*, the average number of duplicate requests per request period, before resetting *dup_req* to zero. The average is computed as an exponential-weighted moving average,

$$ave_dup_req \leftarrow (1-\alpha)\ ave_dup_req + \alpha\ dup_req,$$

with $\alpha = 1/4$ in our simulations. Thus, *ave_dup_req* gives the average number of duplicate requests for those request events for which that member has actually set a request timer.

When a request timer either expires or is reset for the first time, indicating that either this member or some other member has sent a request for that data, the member computes *req_delay*, the delay from the time the request timer was first set (following the detection of a loss) until a request was sent (as indicated by the time that the request timer either expired or was reset). The variable *req_delay* expresses this delay as a multiple of the roundtrip time to the source of the missing data. The member computes the average request delay, *ave_req_delay*.

In a similar fashion, a *repair period* begins when a member receives a request and sets a repair timer, and ends when a member receives a request and sets a repair timer for a different data item. In computing *dup_rep*, the number of duplicate repairs, the member considers only those repairs for which that member at some point set a repair timer. At the end of a repair period the member updates *ave_dup_rep*, the average number of duplicate repairs.

When a repair timer either expires or is cleared, indicating that this member or some other member sent a repair for that data, the member computes *rep_delay*, the delay from the time the repair timer was set (following the receipt of a request) until a repair was sent (as indicated by the time that the repair timer either expired or was cleared). As above, the variable *rep_delay* expresses this delay as a multiple of the roundtrip time to the source of the missing data. The member computes the average repair delay, *ave_rep_delay*.

Figure 10 gives the adaptive adjustment algorithm used in our simulator to adjust the request timer parameters C_1 and C_2. The adaptive algorithm is based on comparing the

```
After a request timer expires or is first
        reset:
   update ave_req_delay
After sending a request:
   C1 − = 0.1
Before each new request timer is set:
   update ave_dup_req
   if closest_requestor on past requests:
      C2 − = 0.1
   else if (ave_dup_req ≥ AveDups)):
      C1 + = 0.1
      C2 + = 0.5
   else if (ave_dup_req < AveDups−ε):
      if (ave_req_delay > AveDelay):
         C2 − = 0.1
      if (ave_dup_req < 1/4):
         C1 − = 0.05
   else C1 + = 0.05
```

Figure 10: Dynamic adjustment algorithm for request timer parameters. In our simulations $\epsilon = 0.1$

measurements *ave_dup_req* and *ave_req_delay* with AveDups and AveDelay, the target bounds for the average number of duplicates and the average delay. An identical adjustment algorithm is used to adapt the repair timer parameters D_1 and D_2, based on the measurements *ave_dup_rep* and *ave_rep_delay*. Figure 11 gives the initial values used in our simulations for the timer parameters. All four timer parameters are constrained to stay within the minimum and maximum values in Figure 11.

```
Initial values:
   C1 = 2; C2 = 2
   D1 = log10G; D2 = log10G
Fixed parameters:
   MinC1 = 0.5; MaxC1 = 2
   MinC2 = 1; MaxC2 = G
   MinD1 = 0.5; MaxD1 = log10G
   MinD2 = 1; MaxD2 = G
   AveDups = 1; AveDelay = 1
```

Figure 11: Parameters for adaptive algorithms

The numerical parameters in Figure 10 of 0.05, 0.1, and 0.5 were chosen somewhat arbitrarily. While this might look like a multitude of constants, the exact value of these constants is not important - all that matters is that they represent small adjustments to the timer parameters C_1 and C_2 as a function of the past observed behavior of the loss recovery algorithms. The adjustments of ± 0.05 and $+0.1$ for C_1 are small, as the adjustment of C_1 is not the primary mechanism for controlling the number of duplicates. The adjustments of -0.1 and $+0.5$ for C_2 are sufficiently small to minimize oscillations in the setting of the timer parameters. Sample trajectories of the loss recovery algorithms confirm that the variations from the random component of the timer algorithms dominate the behavior of the algorithms, minimizing the effect of oscillations.

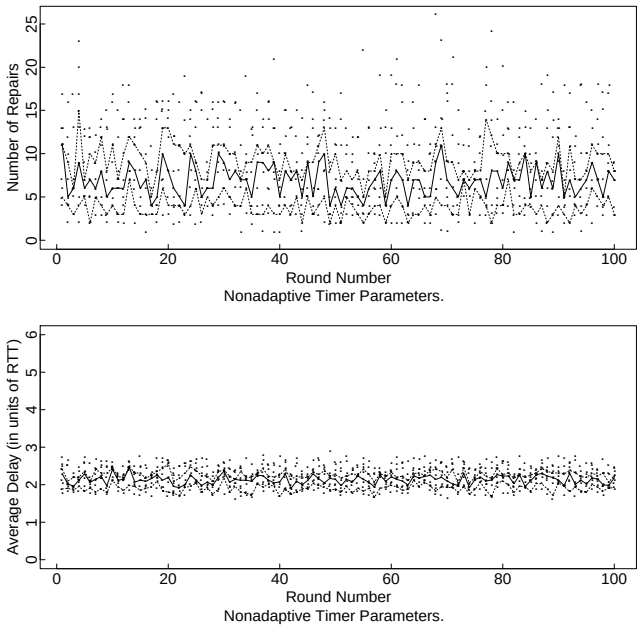

Figure 12: The non-adaptive algorithm.

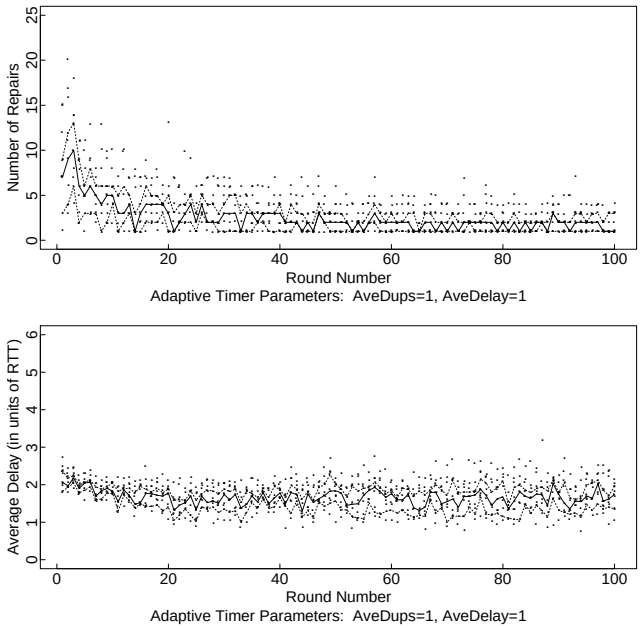

Figure 13: The adaptive algorithm.

In our simulations we use a multiplicative factor of 3 rather than 2 for the request timer backoff described in Section 3.2. With a multiplicative factor of 2, and with an adaptive algorithm with small minimum values for C_1, a single node that experiences a packet loss could have its backed-off request timer expire before receiving the repair packet, resulting in an unnecessary duplicate request.

We have not attempted to devise an optimal adaptive algorithm for reducing some function of both delay and of the number of duplicates; such an optimal algorithm could involve rather complex decisions about whether to adjust mainly C_1 or C_2, possibly depending on such factors as that member's relative distance to the source of the lost packet. For a sparse session in a tree topology, increasing C_2 reduces the number of duplicate requests; our adaptive algorithm relies largely on increases of C_2 to reduce duplicates. Our adaptive algorithm also decreases C_2 for members who have sent requests, if duplicate requests have come from members further from the source of the requested packet. (In our simulations "further from the source" is defined as "at a reported distance greater than 1.5 times the distance of the current member".) Our adaptive algorithm only decreases C_1 for members who have sent requests, or when the average number of duplicates is already small.

Figures 12 and 13 show simulations comparing adaptive and non-adaptive algorithms. The simulation set in Figure 12 uses fixed values for the timer parameters, and the one in Figure 13 uses the adaptive algorithm. From the simulation set in Figure 4, we chose a network topology, session membership, and drop scenario that resulted in a large number of duplicate requests with the non-adaptive algorithm. The network topology is a bounded-degree tree of 1000 nodes with degree 4 for interior nodes, and the multicast session consists of 50 members.

Each of the two figures shows ten runs of the simulation, with 100 loss recovery rounds in each run. The same topology and loss scenario is used for each of the ten runs, but each run uses a new seed for the pseudo-random number generator to control the timer choices for the requests and repairs. In each *loss recovery round* a packet from the source is dropped on the congested link, a second packet from the source is not dropped, and the loss recovery algorithms are run until all members have received the dropped packet. The x-axis of each graph shows the round number. For each figure, the top graph shows the number of requests in that round, and the bottom graph shows the loss recovery delay. Each round of each simulation is marked with a jittered dot, and a solid line shows the median from the ten simulations. The dotted lines show the upper and lower quartiles.

For the simulations in Figure 12 with fixed timer parameters, one round differs from another only in that each round uses a different set of random numbers for choosing the timers.

For the simulations with the adaptive algorithm in Figure 13, after each round of the simulation each session member uses the adaptive algorithms to adjust the timer parameters, based on the results from previous rounds. Figure 13 shows that for this scenario, the adaptive algorithms quickly reduce the average number of repairs, reaching steady state after about forty iterations. Figure 13 also shows a small reduction in delay.

To explore the adaptive algorithms in a range of scenarios, Figure 14 shows the results of the adaptive algorithm on the same set of scenarios as that in Figure 4. For each scenario (i.e., network topology, session membership, source member, and congested link) in Figure 14, the adaptive algorithm is run repeatedly for 40 loss recovery rounds, and Figure 14 shows the results from the 40th loss recov-

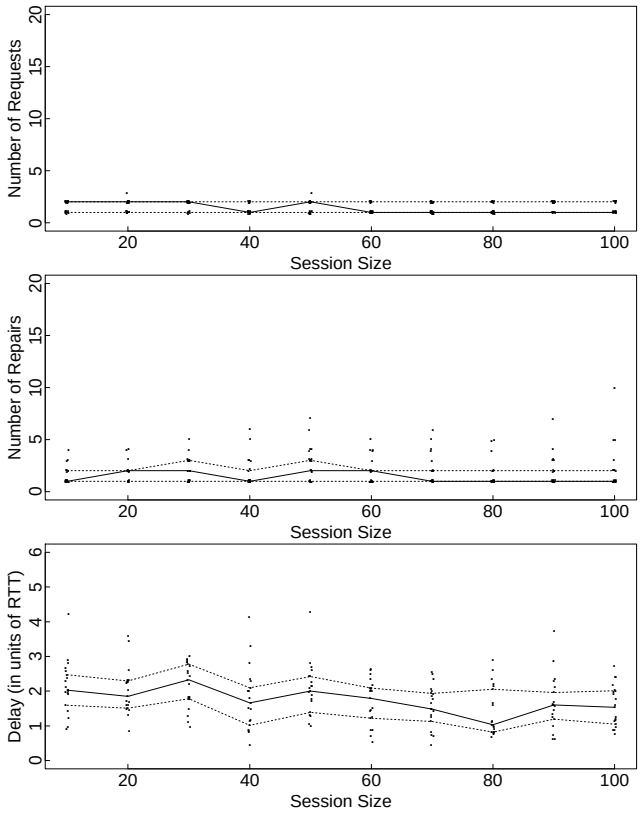

Figure 14: Adaptive algorithm on round 40, for a bounded-degree tree of 1000 nodes with degree 4 and a randomly picked congested link.

ery round. Comparing Figures 4 and 14 shows that the adaptive algorithm is effective in controlling the number of duplicates over a range of scenarios.

Simulations in [13] show that the adaptive algorithm works well in a wide range of conditions. These include scenarios where only one session member experiences the packet loss; where the congested link is chosen adjacent to the source of the packet to be dropped; and for a range of underlying topologies, including 5000-node trees, trees with interior nodes of degree 10; and connected graphs that are more dense that trees, with 1000 nodes and 1500 edges.

In actual multicast sessions, successive packet losses are not necessarily from the same source or on the same network link. Simulations in [13] show that in this case, the adaptive timer algorithms tune themselves to give good average performance for the range of packet drops encountered. Simulations in [13] show that, by choosing different values for AveDelay and AveDups, tradeoffs can be made between the relative importance of low delay and a low number of duplicates.

In the simulations in this section, there is only one congested link, and each packet that is dropped is dropped on only that one link. More realistic simulations would include scenarios with multiple locations for drops of a single packet, and would use an extended SRM that incorporates local recovery mechanisms into the loss recovery algorithms.

Similarly, in the simulations in this section, none of the requests or repairs are themselves dropped. In more realistic scenarios where not only data messages but requests and repairs can be dropped at congested links as well, members have to rely on retransmit timer algorithms to retransmit requests and repairs as needed. Obviously, this will increase not only the delay, but also the number of duplicate requests and repairs in different parts of the network. The use of local recovery, described in the following section, would help to reduce the unnecessary use of bandwidth in the loss recovery algorithms.

7.2 Local recovery

With SRM's global loss recovery algorithm described above, even if a packet is dropped on a link to a single member, both the request and the repair are multicast to the entire group. In cases where the neighborhood affected by the loss is small, the bandwidth costs of the loss recovery algorithm can be reduced if requests and repairs are multicast to a limited area. In this section we sugest that local recovery can be quite effective in reducing the unnecessary use of bandwidth.

Scenarios that could benefit from local recovery include sessions with persistent losses to a small neighborhood of members and isolated late arrivals to a multicast session asking for back history. Studies of packet loss patterns in the current Mbone [39] suggest that packet loss in multicast traffic is most likely to occur not in the "backbone" but in the "edges" of the multicast network. In addition, the larger the multicast group, the more likely it is that a packet will be dropped somewhere along the multicast tree, even in the absence of a particular persistent point of congestion. Forward Error Correction (FEC) [29] and Explicit Congestion Notification (ECN) [11] both have great potential for reducing the negative impacts of transient or mild congestion for reliable multicast applications. However, links with persistent congestion and persistent packet drops are likely to remain. In this case, local recovery is needed to ensure that the fraction of bandwidth used for request and repair messages scales well as the multicast group grows.

We are not at this stage proposing a complete set of algorithms for implementing local recovery. We explore in this section a set of mechanisms that can be used to limit the scope of a request and of an answering repair. The question of how a member decides the scope to use for a particular request is an area for future research.

Local recovery assumes that the member sending the request has some information about the neighborhood of members sharing recent losses. We define a *loss neighborhood* as a set of members who are all experiencing the same set of losses. End nodes should not know about network topology, but end nodes can learn about "loss neighborhoods" from information in session messages, without learning about the network topology.

For each member, we call a loss a *local loss* if the number of members experiencing the loss is much smaller than the total number of members in the session. To help identify

loss neighborhoods, session messages could report a member's loss rate, that is, the fraction of data for which a request timer was set. In addition, session messages could report a "loss fingerprint", i.e., the names of the last few local losses.

A member should send a request with local scope when recent losses have been confined to a single loss neighborhood, and when this local request seems likely to reach some member capable of answering it. If no repair is received before a backed-off request timer expires, then the next request can be sent with a wider scope.

7.2.1 Administrative scoping

One simple and now widely available mechanism for local recovery is the use of administrative scope in IP multicast. If a member believes that both the loss neighborhood and a potential source of repairs are contained in the local administratively-scoped neighborhood, then both the request and the repair can be sent with administrative scoping, so that both messages are restricted to that neighborhood. This is most likely to be of use for larger administratively-scoped neighborhoods.

7.2.2 Separate multicast groups

Another potential mechanism under investigation is the use of separate multicast groups for local recovery [22]. In this scheme, the initial requestor creates a separate multicast group for local recovery and invites other nearby members to join that multicast group. The multicast group must include some member capable of sending repairs. This mechanism is appropriate when there is a stable loss neighborhood that results from a particular lossy link, or when an isolated member joins a group late and asks for past history. Kasera, Kurose, and Towsley [20] explore a somewhat-different use of multiple multicast groups for recovery aimed primarily at reducing the costs of processing unwanted packets at receivers.

7.2.3 TTL-based scoping

A third possible mechanism for local recovery is for members to use *time-to-live-* or TTL-based scope to limit the reach of request and repair messages. In the current Mbone, each link (more precisely, each interface or tunnel) is assigned a *threshold*, with a default threshold of one. The threshold is the minimum TTL required for an IP multicast packet to be forwarded on that link, and is used to control the scope of multicast packets. Every multicast router decrements the TTL of a forwarded packet by one. In order to limit the scope of a request or repair message, the sender simply sets each packet's TTL field to an appropriate value. By including the initial TTL in a separate packet field, members receiving the request (or reply) message explicitly learn the original TTL as well as the hop count for the path from the source.

The simplest version of TTL-based local recovery is a one-step repair algorithm. In this approach, a request sent with TTL h might be answered with a repair sent with TTL $h + k$, where k is the number of hops to the original requestor. In this way, the repair would be guaranteed to reach all of the members reached by the original request (if we optimistically assume that multicast routes and thresholds are symmetric). However, simulations suggest that one-step repair is not very effective — there is significant unnecessary use of bandwidth by the repair packets.

A two-step repair message is considerably more effective in limiting the unnecessary use of bandwidth. In the first step of the repair, a local repair is sent with the same TTL used in the request. This TTL should be sufficiently large to reach the original requestor, given sufficient symmetry, but not necessarily sufficiently large to reach all of the members reached by the original request. The local repair includes the name of the member whose request triggered the repair. In the second step of the repair, the requestor, upon receiving the first local repair naming itself as the original requestor, resends the repair using the same TTL as in the original request. In this way the repair is received by all of the members who saw the original request.

We use simulations to explore the optimal behavior that could be achieved from two-step local recovery. First we examine networks where all links have a link threshold of one, and next we examine networks with a range of values for the link thresholds.

To explore the optimal possible performance, we assume that the loss neighborhood is stable, and that members have some method for estimating h_1 and h_2, where h_1 is the minimum TTL needed to reach all members in the loss neighborhood, and h_2 is the minimum TTL needed to reach some member not in the loss neighborhood. Further, we assume that for each loss recovery event, the request/repair algorithms exhibit their optimal behavior. That is, we assume that there is a single request and a single repair, and that both come from the members closest to the point of failure. We restrict attention to scenarios where the loss neighborhood contains at most 1/10-th of the session members.

Figure 15 shows the results of such an optimal execution of the two-step local recovery algorithms in a large bounded-degree network of degree four, with link thresholds of one. The x-axis in each graph shows the session size. For each session size, twenty simulations are run, each with a different session membership, source, and randomly-chosen congested link for the dropped packet. The results of each simulation are represented by a jittered dot. The three lines indicate the first, second, and third quartiles.

In the top graph of Figure 15, the y-axis shows the fraction of session members reached by the repair. In the bottom graph of Figure 15, the y-axis shows the number of session members in the *repair neighborhood*, that is, the number of session members reached by the repair, as a multiple of the number of members in the loss neighborhood. Additional simulations not reported here show that local recovery with two-step repairs can work well in networks with a range of topologies and link thresholds. Simulations

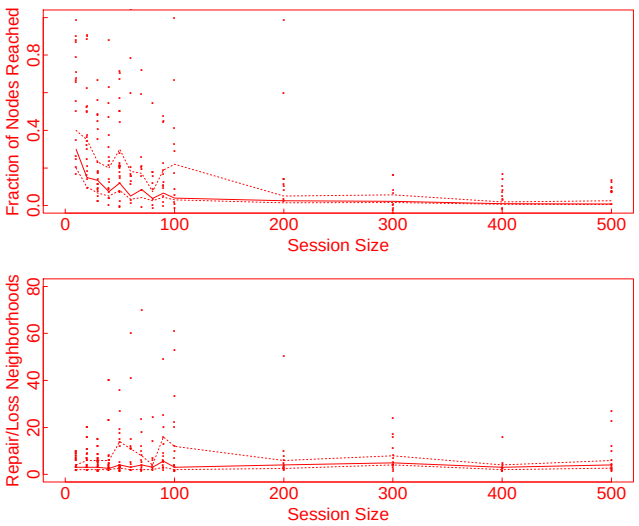

Figure 15: Local recovery with two-step repairs in bounded-degree trees with 1000 nodes, thresholds of one.

in [13] show that, in contrast to two-step repairs, one-step repairs are fairly inefficient in their use of bandwidth, even given an optimal setting of the the TTL of the original request.

8 Related research on reliable multicast

The literature is rich with architectures for reliable multicast [27, 9]. Several of the centralized approaches to reliable multicast are discussed briefly in [12, 13]. In this section we focus on those approaches to reliable multicast that are more closely related to SRM.

The Xpress Transport Protocol (XTP) [36, 37] is designed for either unicast or one-to-many multicast communication. Reliable communication is based on negative acknowledgments. The sender may also initiate a synchronizing handshake, to determine the status of the receivers. In this case, receivers each use a "slotting" technique to wait a random delay before sending their control packet, to reduce a control packet implosion. The combined slotting and damping techniques proposed in [36] to reduce NACK suppression have been described earlier in this paper. In XTP receivers or routers can impose a maximum data rate and maximum burst size on the sender.

Several proposals for reliable multicast use *secondary servers* (also called *Designated Routers* or *Group Controllers* in different proposals), to handle retransmissions within a subgroup of the multicast group. One such protocol, Log-based Receiver-reliable Multicast (LBRM) [15], was designed to support Distributed Interactive Simulation (DIS). The receiver-based reliability is provided by primary and secondary logging servers. Receivers request retransmissions from the secondary logging servers, which requests retransmissions from the primary logging server. Both the source and the secondary logging servers use either deterministic or probabilistic requests to select between unicast and multicast retransmissions.

LBRM uses a variable heartbeat scheme that sends heartbeat messages (e.g., session messages) more frequently immediately after a data transmission. In an environment when the basic transmission rate is low, this variable heartbeat enables receivers to detect losses sooner, with no penalty in terms of the total number of heartbeat messages transmitted. While the variable heartbeat scheme would not be appropriate for an application such as wb, where the original congestion could itself result from many senders sending data at the same time, the variable heartbeat scheme could be quite useful for an application with a natural limit on the worst-case number of concurrent senders, and would be easily implementable in SRM.

Like LBRM and SRM, the Reliable Multicast Transport Protocol (RMTP) [21] also includes among its goals scalability and receiver-based reliability. RMTP accomplishes this by using Designated Routers (DRs) in each region of the multicast group, where the DRs receive incoming acknowledgements and perform retransmissions as needed. RMTP uses windowed flow control tuned to the requirements of the worst-case receiver. The problem of dynamically choosing DRs for a given multicast tree is left for continued research.

A Local Group Concept is proposed in [14], where the multicast group is divided into Local Groups, each represented by a Group Controller that handles retransmissions for members in the Local Group. The Group Controller is not a router or a separate server, but simply one of the members of the multicast group. Hofmann in [14] aims at the dynamic generation of Local Groups and of Group Controllers, but does not explore in detail the algorithms for finding the nearby Local Group, responding to the failure of a local Group Controller, or choosing a new Group Controller.

Perhaps the most well-known work on reliable multicast is the ISIS distributed programming system developed at Cornell University [2, 16]. ISIS provides causal ordering and, if desired, total ordering of messages *on top of* a reliable multicast delivery protocol. Therefore the ISIS work is to some extent orthogonal to the work described in this paper, and further confirms our notion that a partial or total ordering, when desired, can always be added on top of a reliable multicast delivery system.

There is also a growing literature on the analysis of reliable multicast schemes. As one example, Bhagwat, Mishra, and Tripathi [4] consider the performance of one-to-many reliable multicast with a block-based ACK scheme. The paper investigates the regime where transfer sizes are large, receivers have limited buffering, and all retransmissions come from the original sender.

Pejhan, Schwartz, and Anastassiou [31] compare several retransmission schemes for multicast protocols for real-time media. The retransmission schemes are intended for real-time media with playback times, so that packets received after the playback time are dropped. They assume that

receivers unicast NACKs to the sender, and retransmissions are done by the sender. Note that these assumptions differ from those of SRM, which is intended for applications without fixed deadlines by which packets have to be received, and which allows retransmissions from members other than the original source.

9 Future work

9.1 Future work on scalable session messages

The SRM framework outlined in this paper assumes that members of the multicast group send session messages and estimate the distance to each of the other group members. For larger groups, we are investigating a hierarchical approach for scalable session messages [35], where members in a local area dynamically select one of the local members to be the *representative*, as far as session messages are concerned. The representatives would each send global session messages, and maintain an estimate of their distance in seconds from each of the other representatives. All other members would send local session messages with limited scope sufficient to reach their representative.

9.2 Future work on local recovery

Section 7.2 has shown that local recovery can be effective in limiting the unnecessary use of bandwidth in loss recovery events, if members can estimate the scope to use in sending local requests. While we discuss in [13] some of the issues in implementing TTL-based local recovery, there are many open questions about which mechanisms should be used to define local-recovery neighborhoods, how individual members should determine whether to send requests with local or global scope, etc. For local recovery based on separate multicast groups, there is ongoing research on algorithms for initiating, joining, and leaving such multicast groups, and for soliciting additional members to join such groups.

In many topologies, the effectiveness of local recovery could be improved by adding members to the multicast group in strategic locations. For example, consider the known stable topologies discussed in [15], where losses are expected to occur mainly on the tail circuits, rather than in the backbone or in the LANs, and the design priority is to keep unnecessary traffic off of the tail circuits. The addition of a session member (i.e., cache) on a node near the local end of the tail circuit, coupled with a local-recovery neighborhood defined to include all members on that end of the tail circuit, would allow local recovery to continue for losses on the local area without adding any unnecessary traffic to the tail circuit itself. For losses on the tail circuit itself, a larger local recovery area that spanned the tail circuit just into the backbone would isolate individual local recovery to independent tail circuits.

9.3 Future work on congestion control

SRM's basic framework for congestion control assumes that the members of the multicast session have an estimate of the *available bandwidth* for the session, and constrain the data transmitted to be within this estimated bandwidth. This framework raises several somewhat separate issues, such as how members determine this available bandwidth; how to detect congestion or avoid potential congestion; and given available bandwidth, which piece of data a member should send first.

Multicast congestion control is a relatively new area for research. For unicast traffic, there is a single path from source to receiver, with a feedback loop provided by returning packets sent by the receiver. In contrast, in a multicast group there could be several sources, and the various communication paths from an active source to the members of the multicast group can have a range of bandwidth, propagation delay, and competing congestion. In this case, how does one define and detect congestion?

With multicast traffic, there are application-specific policy decisions about whether or not to tune the congestion control procedures to the needs of the worst-case receiver; these questions do not arise with unicast transmissions. Tuning the sending rate to the worst-case receiver is only viable for a multicast group with a controlled membership; otherwise, the multicast group would be vulnerable to denial-of-service attacks by members joining the group from an extremely-low-bandwidth path. Given an uncontrolled membership, and a group where the bandwidth along different paths in the multicast group differs substantially, the sender could tune the sending rate to the needs of the majority of receivers, requiring that receivers on more congested paths unsubscribe from the multicast group.

A receiver-based approach under investigation for the video tool vic [24] is to divide the total data transmission into several substreams, with each being sent