Scalable Reliable Multicast with Layered Recovery and Low-Overhead Network Delay Estimation

Injong Rhee[†], *Srinath R. Joshi*[†], *Minsuk Lee*[†], *S. Muthukrishinan*[‡] *and Volkan Ozdemir*[†],

[†]Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534
{rhee,srjoshi,mlee1,vozdemi}@csc.ncsu.edu

[‡]AT&T Labs-Research
180 Park Avenue
Florham Park, NJ 07932
muthu@research.att.com

Feb 1999

Abstract

We study two problems that arise in designing scalable reliable multicast protocols.

The first problem we study is that of localizing repair packets when packets are lost. When repair packets are multicasted, a highly lossy receiver may swamp the entire multicast "group" with duplicate repair packets thereby wasting bandwidth; thus, the protocols need repair locality. In this paper, we present a novel multicast layering protocol where the sender proactively distributes FEC repair packets among multiple multicast groups. Receivers can selectively tune in to a subset of these multicast groups to obtain only the number of repair packets they require. We present a highly efficient algorithm that dynamically determines the optimal distribution of FEC repair packets to a given (small constant) number of multicast groups. The running time of this algorithm is independent of the number of receivers in the multicast session, and it is hence scalable.

The second problem we address is to estimate the network delay between each pair of nodes in the multicast session. This key parameter is useful, among other things, in suppressing the implosion of request and repair packets, and in detecting congestion. Existing implementations use $O(n)$ multicasts with $O(n)$ message size each (total of $O(n^2)$ bits); here, n is the session size. This paper presents a new delay estimation protocol that requires $O(n)$ multicasts only with $O(1)$ message size each. This protocol is hence more scalable. Furthermore, it does not require synchronized clocks, or any knowledge of network topology or the session size.

Our solutions to these two problems can be integrated into many known reliable multicast protocols to enhance their scalability. For concreteness, we focus on singly scoped and hierarchically scoped SRM as well as tree-based protocols and present combined protocols incorporating our solutions into each of them. Our simulation experiments suggest that our solutions can substantially enhance the scalability of these reliable multicast protocols.

1 Introduction

Reliable multicast protocols need to address the issue of recovering from packet losses. The crux is to design a protocol that scales to tens of thousands, or even millions of receivers since the added control complexity of multicasting has a greater payoff with such large numbers of receivers. Scalability is thus an important problem in reliable multicasting, and it is a well-studied one. Many clever protocols have been proposed, e.g., SRM [1], RMTP[2], SHARQFEC [3], TMTP [4], to name a few; the resource web site [5] gives an extensive background. In this paper, we identify two bottlenecks in known scalable reliable multicast protocols; our main results are the efficient solutions we develop for these two problems. We also integrate our solutions into three well-known protocols. The resulting protocols are substantially more scalable than their original versions, as our experiments indicate.

In what follows, we develop the background of scalable reliable multicast to identify the bottlenecks. A natural approach to recover from packet losses is to make the sender retransmit the lost packets to individual receivers. However, such sender-centric retransmission does not scale well. In a large-scale multicast session, the probability that a given packet is lost by some receiver is high; thus, the sender may end up retransmitting every packet. Additionally, if a packet is lost near the sender, most receivers would lose that packet, and it leads to repair traffic that is proportional to the session size. We may avoid this problem by allowing the sender to multicast the retransmission. However, as recent MBone studies [6] indicate, many packet losses are not correlated, and different receivers may experience different loss rates. This leads to the well-known *repair-locality problem* whereby repair traffic is not localized to its desired receivers. Thus, when retransmission is multicasted, receivers may end up receiving many "unwanted" packets in the repair traffic. Sender-centric retransmission schemes also suffer from the well-known *implosion problem* in which the sender is potentially flooded by control traffic (request for packets, negative acknowledgment from receivers, status requests, etc).

The popular consensus now seems to be to effectively delegate the responsibility of recovery to the receivers. SRM [1], perhaps the most popular scheme for reliable multicast, allows receivers to multicast requests to the entire group. Any receiver with requested packets can multicast it. With clever use of randomized timers and suppression, SRM effectively solves the implosion problem. Unfortunately, SRM does not solve the repair locality problem. This problem is alleviated, but not entirely solved, by local and hierarchical scoping (grouping of receivers) [7].

There are tree-structured protocols such as TMTP [4], RMTP [2], and LBRM [8] that solve the implosion and repair locality problems by imposing a logical tree structure to the multicast session. Specialized receivers located at the root of the sub-trees of the logical tree receive requests and effect retransmission only to their own children in the tree. These protocols work without any router support, but they need specialized receivers. There are other protocols such as PGMP [9] and LSM [10] which propose to modify the routers in order to localize repair packets to the region where they can be most effective. For these protocols to be effectively used, "special" receivers (or routers) need to be widely deployed. Managing the logical tree involving a large number of specialized receivers under network partition or machine failure would create an enormous administrative burden.

Another approach to solving both the implosion and repair locality problems is to use forward error correction (FEC). This involves splitting the original packet stream into groups of B packets, called *blocks*. For each block, h FEC encoded packets are generated for suitably chosen h. Receivers can recover the original block by receiving any B packets out of the $B + h$ ones. When combined with an appropriate ARQ technique, the FEC technique incurs very low network overhead [11, 12, 13, 14, 15, 3].

The FEC approach can be employed in any of the protocols we have described so far such as SRM, tree-based protocols etc. A noteworthy example is the SHARQFEC protocol [3] that combines hierarchical scoping and hybrid FEC/ARQ. It breaks the entire multicast group into hierarchically nested scopes and designates a receiver within each scope as the "zone closest receiver" (ZCR). After receiving each data block, each ZCR proactively multicasts FEC packets to the receivers in its scope. Since proactive FEC packets and SRM-style suppression can reduce much of repair and request traffic, a scope can contain many more receivers than a sub-tree in tree-based protocols. Therefore, fewer ZCRs are needed than the number of designated receivers needed in tree-based protocols. In [3], it is shown that with scopes as large as 500 receivers, SHARQFEC can potentially scale to millions of receivers.

To summarize, scoping and employing FEC together with suppression techniques, leads to reliable multicast protocols that are suitably scalable. Now we can describe the two bottlenecks that inhibit further scalability and efficiency of these protocols.

The first bottleneck arises when the FEC technique is employed within a scope or a group. The main issue is to determine how many FEC packets to transmit, and the suitable protocol to schedule the transmission of these FEC packets. Most existing schemes force the sender to multicast as many FEC packets as the most lossy receiver in the session requires. This is effective only if all the receivers in the group have similar loss rates. Unfortunately, in most existing protocols, scopes are not defined by the loss rates of receivers, but by their physical locations. In [16, 7], scopes are defined to be receivers within a certain "hop count" or time-to-live (TTL). Administrative scoping incorporated into multicast addresses [17] is also based on physical locations. SHARQFEC [3] also defines scopes based on the physical regions, cities, suburbs, etc. Inevitably, a large scope defined only by physical locations comprises receivers with widely varying loss rates; thus, receivers with low loss rates receive far too many redundant repair packets, and the overall repair traffic is excessive.

The second bottleneck in designing reliable multicast protocols that scale to a large number of receivers is in network delay estimation. When using SRM-style suppression, these protocols must engage in frequently estimating the pairwise delay between receivers in a scope. The currently best known estimation process involves each receiver in a scope to multicast a *session* message involving delay information about all the other receivers in the scope [3, 7]. The delay information consists of at least two time stamps. Let us do a quick calculation to estimate the bandwidth requirements for this session traffic. Say we have a scope with 500 members; the size of a session message is as large as $500 \times 2 \times 4 = 4$ KB, assuming a micro-second resolution on the time scale (i.e., 4 bytes for each time stamp). As in SHARQFEC, if we set the interval between two consecutive session message transmissions to be one second, each receiver will be subject to the total session traffic of $4 \times 500 = 2$ MB/s which is prohibitive. On the other hand, if we allow only 5 KB/s bandwidth for session traffic, then the time interval between session messages has to be adjusted to about 7 minutes, which does not faithfully represent changing delays between receivers. Thus, with the existing delay estimation protocol, both SRM and SHARQFEC cannot be widely deployed for large-scale reliable multicast.

Contribution Our contribution is threefold. There are briefly as follows (further details below):

1. We present a novel protocol called *Multicast Layered Recovery* (MLR). A multicast session is allocated with multiple multicast group addresses. The sender multicasts different amounts of FEC packets to different multicast groups. Each receiver joins a subset of the groups that together provide the number of FEC packets it needs under its current loss rate. We present a highly efficient algorithm to find the optimal allocation of repair packets to different multicast groups in order to minimize redundant traffic; its running time is independent of the number of receivers in the session. Thus, this procedure scales effectively.

2. We present a protocol that for each receiver, estimates the delay from it to each of the others in its scope during a multicast transmission. Our protocol uses $O(n)$ multicasts with $O(1)$ message size each; here, n is the size of a scope (or the session size if singly scoped). Existing protocols, in contrast, use $O(n)$ multicasts with message size $O(n)$ each [1]. Our protocol thus has low overhead, and is hence more scalable than the existing ones.

3. Our two solutions can be employed in various existing reliable multicast protocols to enhance their repair locality, and scalability. In this paper, we integrate our solutions with SRM, a tree-based protocol, and a hierarchically scoped protocol. The resulting protocols are substantially enhanced versions of their original in their scalability.

In what follows, we give a more detailed overview of our first two contributions.

Multicast Layered Recovery. The outline of the protocol is as follows. Within each scope (session, if there is only one scope), K repair multicast groups (hereafter referred to as *repair groups*) are allocated. MLR uses a hybrid FEC/ARQ scheme. Given the maximum number of FEC repair packets, f_{max}, required by a receiver in the scope, a ZCR (or the sender) partitions f_{max} packets into F= $\{\phi_1, \phi_2, \ldots, \phi_K\}$ groups where K is given as a some small constant, and transmits ϕ_i repair packets to a different multicast group. While the receiver with the worst packet loss rate joins all the multicast groups to receive f_{max} repair packets, others can adjust the amount of repair traffic transmitted to them by selectively joining only a subset of the K multicast groups.

Given the number of FEC packets needed by the different receivers, a ZCR chooses optimal F to minimize the total redundant repair traffic to each receiver in its scope. For instance, if there are five receivers which require 2, 2, 1, 4 and 4 repair packets respectively, and $K = 2$, then ϕ_1 and ϕ_2 are 2 each in order to minimize the total number of redundant repair packets that all receivers would get. The first three receivers join the first multicast group while the last two receivers join both of the repair groups. The total number of redundant repair packets is equal to one because only the third receiver gets more than it requires when joining the first repair group.

In this paper, we develop an algorithm determining the optimal F with running time $O(U^2K)$ where U is the maximum number of FEC repair packets to add per block. While an unbounded number of repair packets can be proactively injected, we restrict U to be less than the block size B which is typically less than 20 (in our simulation, we chose 16). This also limits the total amount of FEC repair packets per block that is received by a member to B. The running time of the algorithm is independent of the number of receivers in the scope, making it scalable for large-scale multicast.

Low overhead network delay estimation. Our protocol uses at most $2n$ multicasts after an initial bootstrap phase. Thus, with $n = 500$, it consumes approximately 10 KB/s session bandwidth (not counting the header size) if the session interval is set to one second; this is quite moderate compared to the $2MB/s$ estimate for the existing protocols. Our protocol has additional features. It does not require synchronized clocks at receivers, or any knowledge of network topology and the identities of other members. We assume that the network delay on a path is symmetric. This assumption is common in most protocols that rely on delay estimation, such as TCP[18], SRM[1], NTP[19], and SHARQFEC[3], to name a few.

The network delay is a key parameter, and as such, our estimation protocol may be useful in other scenarios such as congestion control protocols etc. We do not explore these aspects here, and we focus only

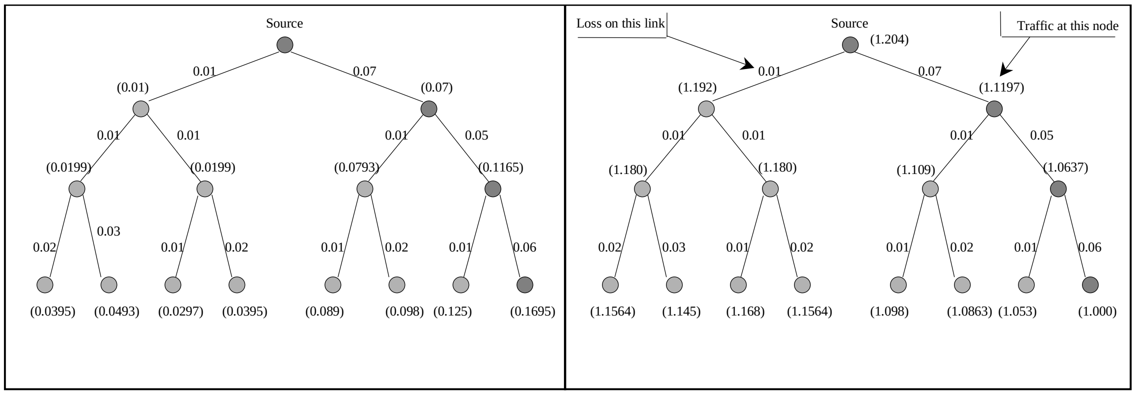


Figure 1: (a) A multicast tree (the number in parentheses near a node represents the loss rate of that node), (b) Normalized traffic volume for non-scoped and non-layered FEC.

on scalable reliable multicast protocols.

Organization We describe our MLR protocol in Section 2, and our network delay estimation protocol in Section 3. We present our simulation results in Section 4. There is a rather large body of work related to scalable, reliable multicasting; we discuss some relevant ones in Section 5. We present concluding remarks as well as potential limitations of our work in Section 6.

2 Multicast with layered recovery

We first illustrate the repair locality problem in more detail with some examples. Then we present our layered protocol to solve this problem. Finally, we show how it can be integrated into two different protocols: SRM and a tree-based protocol. In the Appendix, we show in detail how MLR can be combined with a hierarchically scoped protocol.

2.1 The repair locality problem

The repair locality problem can be best explained using examples. We modified several examples from [3].

Figure 1(a) shows that a single sender at the root of the multicast routing tree multicasts data to the other nodes which represent the receivers. Branches represent multicast routing paths. Different branches are subject to different loss rates, from low to high rates. The total loss at each node can be calculated by compounding the loss rate of every link between the sender and that receiver. In Figure 1(a), the worst case receiver loses about 17% of the packets.

Figure 1(b) illustrates the scenario when the sender multicasts the *maximum* number of repair packets requested by any receiver. It shows the number of redundant FEC repair packets received by a receiver under this scenario. The numbers in parentheses indicate the expected (normalized) amount of FEC repair packets that each node would receive when *both* data and FEC repair packets are subject to loss. While the

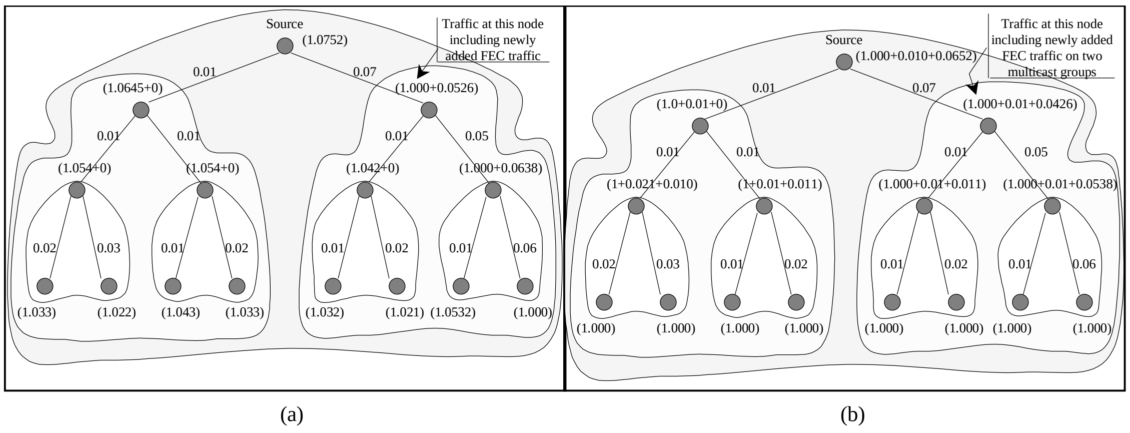


Figure 2: (a) SHARQFEC with normalized traffic volume, (b) MLR with normalized traffic volume 1.

worst receiver recovers data blocks with no redundant repair packets, other receivers are subject to redundant traffic up to 16.8%. Note that if retransmission were used instead of FEC repair packets, this number would be much larger.

Figure 2(a) shows a scoped version (SHARQFEC) of the same scenario as the above. The ZCRs (i.e., internal nodes and the sender in the figure) of nesting scopes inject sufficient repair packets to satisfy the receiver with the highest loss rate among its children. The amount of redundant packets is greatly reduced because of scoping. However some receivers (especially the ones in the left side of the tree) are still subject to a significant amount of redundant traffic (about 6.4%). This scenario is quite real. Since a single scope can expand to include many receivers (e.g., five hundred receivers as proposed in [3]), it is highly unlikely that the receivers in the same scope have similar loss rates. Therefore, scoping alone is not adequate for solving the repair locality problem.

Figure 2(b) shows the result when MLR is applied to scoping assuming each ZCR can have two repair groups. The label $(x + y + z)$ at each internal node denotes the normalized amount of traffic x that it gets from its parent, the normalized amount of the FEC traffic y that it adds to the first repair group, and the normalized amount z of the FEC traffic it adds to the second repair group. Each receiver has a choice of joining one or both groups. Under the current scenario, every node receives no more than what it requires to recover from its own losses. This example suggests that with a few additional multicast groups and appropriate assignment of the number of repair packets sent to each repair group, we can substantially reduce the redundant repair traffic. In the following sections, we develop and formalize the MLR protocol in greater depth.

2.2 A single scoped MLR protocol

We first sketch the outline of MLR for a single scope where the sender is the only node that can proactively multicast FEC repair packets. Then we show how it can be integrated with SRM [1] and a tree-based protocol such as RMTP [2].

2.2.1 The outline of the protocol

The sender has a fixed number of multicast groups $g_0, g_1, \ldots, g_K$, where K is a small constant. K is typically less than B, where B is the number of packets in a block. The sender multicasts data blocks to multicast group g_0, which we call the *base group*. The other multicast groups are *repair groups*.

Depending on the number of multicast groups allocated to the sender, the protocol runs in one of two modes: *static* or *dynamic*. Typically, if K is larger than B/c where c is a small constant and a protocol parameter, it chooses the static mode; otherwise, it is in the dynamic mode. The mode of MLR is determined at the beginning of the session.

In the static mode, the sender always transmits $\phi_i = \lceil B/K \rceil$ FEC repair packets to group g_i. If there are sufficiently many repair groups, the sender can transmit the groups of repair packets in a fine granularity; hence, the redundant traffic is minimal. For example, when $K \geq B/2$, each receiver gets at most one more repair packet per block than it requires since at most two repair packets can be sent to each repair group. The added benefit of dynamically allocating different amounts of repair packets to the repair groups is minimal. Note that in the static mode, the sender does not need to know how many repair packets receivers require, so no feedback is necessary.

In the dynamic mode, each receiver i periodically sends feedback to the sender containing information on the number of repair packets it needs, denoted f_i. f_i is computed based on i's loss rate. The loss rate is estimated through simple exponential weighted moving average; this estimation is based on the assumption that persistent loss rates tend to vary slowly. The interval between two consecutive transmission of feedback is adjusted so that the amount of traffic to the sender is no more than its back traffic in a point-to-point transmission such as TCP. The discussion on how feedback is sent is given in Section 2.2.4. The sender keeps the statistics on the number of repair packets requested by the receivers. This is used to determine the optimal distribution of repair packets among repair groups. Let ϕ_j be the number of repair packets to be sent to repair group g_j. ϕ_j's are chosen so as to minimize total (equivalently, the average) number of redundant repair packets sent to the receivers. We discuss the algorithm to compute ϕ_j in Section 2.2.2. Initially, when the sender does not have sufficient knowledges of the receivers' requirements, ϕ_j is set to $\lceil B/K \rceil$.

The sender includes the information about the distribution of repair packets in each packet it sends out. We call this the *repair group information* (RGI). In the static mode, RGI needs to occupy only one octet to store B/K because all repair groups carry the same number of repair packets. In the dynamic mode, a data packet needs to include K numbers, each indicating the number of repair packets to be sent to a repair group (i.e., ϕ_j's). Typically, in the dynamic mode, K is less than $B/2$. Thus, these numbers may occupy up to $\lceil \log_2(B/2) \rceil \times B/2$ bits. For example, for $B = 16$, this requires only three octets to store the 8 numbers.

We assume that the transmission rate of data and repair packets is governed by a flow and congestion control mechanism which is not described in this paper.[1] The overall transmission sequence works as follows. The sender multicasts the data to the base group g_0. Immediately after that, the sender multicasts a group of ϕ_j FEC encoded repair packets to g_j, in increasing order of j's. Two consecutive repair packets are delayed by Δ_f units of time in order to reduce the effect of burst losses (in our simulation, we set it to three packet intervals). The rationale behind this scheme is that the transmission rate of the sender must be about the same as that in non-layered hybrid ARQ protocols.

Based on the latest RGI, a receiver i decides to be in a subset of repair groups as described below. It always belongs to the base group g_0. Then it finds the minimum r, $r \leq K$, such that the sum of ϕ_1 to ϕ_r is

[1]For more information on the subject, see [20].

at least as large as f_i, and joins multicast groups $g_1, g_2, \ldots, g_r$. Thus if a receiver joins g_j, then it has to join groups from g_1 to g_{j-1}. Since repair packets in $g_k, 1 \leq k \leq K-1$ are always transmitted before those in g_{k+1}, this rule allows receivers to recover from losses as soon as possible. This is the *layering* aspect of our protocol.

If the number of packets lost per block (including data and repair) by a receiver is larger than the incoming repair packets from its current repair groups, then the packet losses are irrecoverable. In that case, the receiver initiates a request for additional transmission of FEC repair or data packets from the sender or other receivers who recover the same block. The specifics of how this request is handled depends on the type of reliable multicast protocols being integrated with MLR. We discuss this issue in detail when we explain how the MLR protocol can be combined with SRM and a tree-based protocol in Sections 2.2.4 and 2.2.5 respectively.

2.2.2 Optimal Layering

Formally our problem is as follows. We have n receivers; the ith receiver has a demand $f_i \geq 1$ (of FEC packets). Say $\max_i f_i \leq U$. We are given a parameter K which is the number of repair groups. Our goal is to choose $\phi_1, \phi_2, \ldots, \phi_K$, $\phi_i \geq 1$. Here, ϕ_i is the number of FEC packets sent to the multicast group i by the sender. All ϕ_i's and f_i's are integers. Each receiver i joins the multicast groups $1, \cdots, j$ such that $\sum_{\ell=1}^{\ell=j} \phi_\ell \geq f_i$ and j is the smallest integer with this property. The *cost* for the ith receiver is $(\sum_{\ell=1}^{\ell=j} \phi_\ell) - f_i$, where j is as above, and the *total cost* is the sum of the cost for each receiver. The problem is to find the solution (ϕ_i's and their ordering $\phi_1, \ldots, \phi_K$) of minimum total cost.

We observe that although the session size n is large, U is substantially small; typically $U \leq 20$. K is also small as described earlier.

To some extent, this problem is related to the k-median problem and, more generally, to the facility location problems in the literature, both of which are mildly hard, to notoriously hard not only to solve exactly in polynomial time, but even to approximate. They have been a subject of intense study even recently in the Theoretical Computer Science community [21]. The focus there is on the case when U is much larger than n and the distance function between f_i's is fairly sophisticated; in our problem, n is very large compared to U and the distance function is simple. Here, we are able to show that our problem can be solved optimally in running time which is a small polynomial in U and K, independent of n.

It suffices to consider the problem of finding the cost C in the optimal solution – recall the definition of the cost from earlier. A particular solution of ϕ_i's with this cost C can be easily retrieved from our description below.

For now, let us consider the subproblem in which our goal is to find the optimal solution using ℓ repair groups for receivers which request at most i FEC packets, that is, f_j's, $1 \leq f_j \leq i \leq U$. Say this solution is $\phi_1, \cdots, \phi_\ell$. We denote its cost by $S(i, \ell)$ and denote $\sum_{j=1}^{j=\ell} \phi_j$ by $SS(i, \ell)$. Note that $SS(i, \ell)$ is the total number of FEC packets sent out by the sender since it sends ϕ_j packets over the jth multicast group if we solve this subproblem alone. The following observation, although simple, proves to be the key. Informally, it says that the number of FEC repair packets sent by the sender over all the multicast groups $1, \ldots, \ell$, while solving this suproblem optimally equals the maximum number of FEC repair packets requested by any receiver which requested no more than i packets. Formally,

Lemma 2.1 *For any ℓ, $SS(i, \ell) = \max_h f_h$ where $f_h \leq i$.*

Proof Sketch. We fix a value of ℓ and show $SS(i,\ell) = \max_h f_h$ where $f_h \leq i$; that suffices. Trivially, $SS(i,\ell) \geq \max_h f_h,\ f_h \leq i$. Therefore, it suffices to prove $SS(i,\ell) \leq \max_h f_h,\ f_h \leq i$. Assume otherwise; thus, the optimal solution $\phi_1, \phi_2, \ldots, \phi_\ell$, $\phi_k \leq i$ for $1 \leq k \leq \ell$, has $SS(i,\ell) > \max_h f_h$ where $f_h \leq i$. Consider the solution $\phi_1, \phi_2, \ldots, \phi_\ell - 1$, $\phi_k \leq i$ for $1 \leq k \leq \ell$. This too is clearly a valid solution, but we can quite easily argue that this has strictly smaller total cost than our optimal solution. Thus we derive contradiction to our assumption. ∎

Henceforth, we let $SS(i)$ denote $SS(i,\ell)$ for any ℓ. From our observation above, $SS(i) = \max_h f_h$ where $f_h \leq i$. We have the following recurrence:

$$S(i,\ell) = \min_{j \mid 1 \leq j < i} S(j, \ell - 1) + C(j + 1 \cdots i - 1),$$

where $C(a \cdots b)$ is the total cost of all u's such that $a \leq f_u \leq b$. It is easy to see that $C(a \cdots b) = \sum_{a \leq f_u \leq b}(SS(b) - f_u)$. In order to solve our problem, it suffices to compute $S(U,K)$.

We compute $S(U,K)$ using dynamic programming. We initialize an array $S[1 \cdots U, 0 \cdots K]$ such that $S[i,0] = \infty$ and $S[i,j] = 0$ for $i \leq j$ and $1 \leq j \leq K$. Also, $S[i,1] = C[1, i-1]$ for all i. Now for each $2 \leq \ell \leq K$, we consider each choice of i and we calculate $S[i,\ell]$ using the equation above.

Say the time taken to determine $C(a \cdots b)$ for any a, b is at most t_Q. There are $O(UK)$ terms of the form $S(i,\ell)$ each of which takes time $O(Ut_Q)$ to compute. The total running time is thus $O(U^2Kt_Q)$. We can naively upper bound t_Q to be $O(U)$ by computing $C(a \cdots b)$ whenever needed. A more efficient solution is as follows:

Lemma 2.2 *We can compute $C(a \cdots b)$ for all a, b in time $O(U^2)$ after which we can determine any $C(a \cdots b)$ in time $t_Q = O(1)$.*

Proof. We compute the table $Z(a,b) = C(a \cdots b)$ for all a, b which can be looked up when needed in $O(1)$ time. In order to compute $Z(a,b)$, we determine tables $X(a,b) = SS(a \cdots b)$ and $Y(a,b) = \sum_{a \leq f_u \leq b} f_u$ for each a, b. Both the tables X and Y can be easily computed in $O(U^2)$ time. We then use the following formula:

$$Z(a,b) = C(a \cdots b) = \sum_{a \leq f_u \leq b}(SS(b) - f_u)$$

$$= \sum_{a \leq f_u \leq b} SS(b) - \sum_{a \leq f_u \leq b} f_u = X(a,b) - Y(a,b)$$

Thus it takes $O(U^2)$ time to compute the Z table. ∎

Based on the arguments above, we have our main result:

Theorem 2.3 *The optimal solution $S(U,K)$ can be determined in time $O(U^2K)$.*

We make several remarks about our result. (1) In order to determine the set of ϕ_i's in the right order with the optimal cost $S(U,K)$, we store index j where the minimum occurs in the formula for $S[i,\ell]$. Using this, we can determine the set ϕ_i correctly and efficiently (as is standard in dynamic programming solutions). (2) We can provide a more sophisticated algorithm that is theoretically more efficient than one we have stated above. But for our implementations, the above solution suffices. (3) Our solution above applies to many other cost functions such as minimizing the maximum of $(\sum_{\ell=1}^{\ell=j} \phi_\ell) - f_i$ over all receivers.

2.2.3 Determining the number of FEC repairs required by a receiver

The number of FEC packets that a receiver needs depends on the pattern and rate of its losses.

We adopt the standard model for packet losses found in the literature [22, 14, 11]. The data packet losses are assumed to undergo burst losses. Hence, they are described by a two-state model: one state (referred to as state 1) represents a packet loss, and the other state (referred to as state 0) represents the successful receipt of the packet. If the system is in state 0, the probability of staying in state 0 is α, and the probability of switching to state 1 is $1-\alpha$. If the system is in state 1, the probability of staying in state 1 is β and the probability of moving to state 0 is $1-\beta$. Both parameters α and β can be obtained from the measured mean loss rate and burst length. Each FEC packet belonging to the same block is separated by Δ_f time distance. Δ_f is set large enough to force the losses of FEC packets to be close to independent. It was reported in [22] that periodic UDP packets separated by as little as 40 msec tend to undergo near-independent losses. Thus, we assume that FEC repair packets undergo independent random losses; hence, they are described using a binomial distribution; again, the independent loss probability is known from measurements. The losses of data packets (in multicast group 0) and those of FEC packets are assumed to be independent.

We can calculate the following two quantities efficiently. Let $P(f,i)$ denote the probability of receiving j packets out of f FEC repair packets; it can be computed in a straightforward way from the definition of the binomial distribution. Let $D(B,i)$ be the probability of receiving i data packets out of a block of B data packets under the two-state model. $D(B,i)$ can be computed using dynamic programming in time $O(B^2)$ using the recursive definition of D in [14].

Two parameters are relevant. The first is ρ, the probability of recovering a block when the total number of repair packet in the multicast groups the receiver belongs to is f. That is, if i packets are received from the base group, at least $(B-i)$ FEC packets are needed from the repair groups in order to recover the block. Thus, $\rho = \sum_{i=\max(B-f,0)}^{B} \left(D(B,i) \times \sum_{j=B-i}^{f} P(f,j) \right)$. The second parameter of relevance is the expected wasted bandwidth due to redundant FEC repair packets. Let $EX(B,f)$ be the expected number of packets received when f FEC packets are used to protect B data packets. It is easy to see that $EX(B,f) = \sum_{i=0}^{B} iD(B,i) + \sum_{j=0}^{f} jP(f,j)$. Then, the normalized expected bandwidth wastage is $EW(B,f) = \frac{EX(B,f)-B}{B+f}$.

One approach to estimate the FEC repair packets for a receiver would be to set ρ very close to 1; that would give a value for f. This approach requires FEC repair packets to protect transmission even from very rarely occurring events such as a large number of data packets in the block being lost and/or FEC packets being continually lost. Thus, it tends to ask for too many packets and waste bandwidth. A more appropriate approach is the one we adopt here, namely, find the number of FEC repair packets that would maximize the probability of recovery given the expected wasted bandwidth is bounded by an acceptable amount for a receiver. Since different receivers may be able to tolerate different amounts of wasted bandwidth, this approach would allow receivers to wage their own "risk" in getting FEC repair packets. In our simulations, we set the tolerable wasted bandwidth to be 5% of the total forward bandwidth.

The effect of the number of FEC packets on the wasted bandwidth and probability of recovery is shown in Figure 3. We calculated ρ and $EW(B,f)$ with $B=16$ while varying f and loss rates. The plot shows that when the normalized expected bandwidth wastage is around 5%-6%, the chances of recovery are significant. Note that the chances of recovery can be up to 20% higher than the case when the bandwidth wastage is minimum.

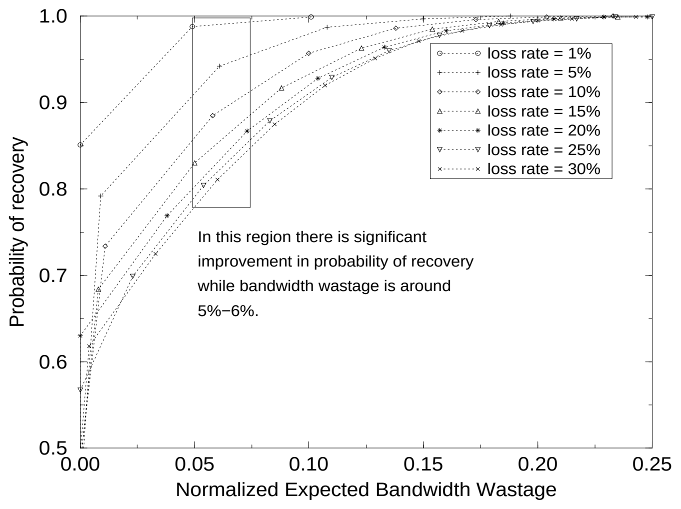


Figure 3: The relation of wasted bandwidth and the probability of recovery when $B = 16$.

2.2.4 Integrating with SRM

We now show how to integrate MLR into SRM. When MLR fails to recover a block, we employ SRM style receiver-centric error recovery. When a receiver fails to recover a block, it multicasts the request for additional transmission as in the SRM protocol. We need to address an important issue before providing further details, namely, does the receiver request retransmission of its missing *original* data packets, or request additional FEC repair packets for the lost block? Retransmitted FEC repair packets for a receiver may additionally help some of the other receivers recover the same block; hence, it has the potential to avoid proliferation of repair packets and utilize bandwidth efficiently. However, this recovery process has some delay. For applications such as video transmissions for which the delay is critical, retransmission of original data packets may help recover the block with partial losses which may suffice. In this paper, we only describe our protocol when retransmissions are FEC repair packets; the modification to deal with retransmission of original data packets is straightforward. We now describe the SRM-style protocol in detail.

Request. When a receiver r fails to recover a block, it sets its *request timer* in the same manner as specified by the SRM protocol. That is, the delay is chosen uniformly ¿from the interval $2^i[C_1 d_{S,r}, (C_1 + C_2) d_{S,r}]$, where C_1 are C_2 are protocol parameters typically set to 2; here, $d_{S,r}$ is the receiver r's estimate of the one-way network delay time from the sender (S) to r during a multicast, and i is the backoff factor, which is described below. When the timer expires, it multicasts a request for FEC repair packets for its incomplete block. The number of repair packets requested is a tunable parameter that depends on the receiver's past experience on the number of duplicate repair packets, denoted d, it gets for each request. If a receiver requires y packets, then it may request $\lceil y/x \rceil$ repair packets. If the receiver receives a request from another receiver for the same block before the timer expires, it increments its backoff factor i by one, and resets the request timer.

Repair If receiver r receives a request from receiver q for n_q packets from a block r has successfully recovered, it sets its *repair timer* to $[D_1 d_{r,q}, (D_1 + D_2) d_{r,q}]$ where D_1 and D_2 are set to 1, and $d_{r,q}$ is the estimated network delay from r to q. We associate with the repair timer for r, a number N_r which is the number of repair packets it would send when the timer expires. N_r is initialized to n_q which is the requested number of repair packets. If it receives further requests for repair packets, N_r is updated to the new requested value if that is larger. If it receives a repair packet ¿from another receiver while the timer is on, N_r is decremented by one. This is the technique used in SHARQFEC [3] for suppressing request and repairs.

When the timer expires and $N_r > 0$, r multicasts N_r repair packets selected as follows. First, the node generates $(F + 1)N_r$ unique FEC-encoded repair packets where F is a tunable parameter chosen based on the size of a block and the computational overhead in generating F FEC repair packets (in our simulations, we set it to B, the block size). Then, it randomly chooses N_r packets from packets with sequence number between F and $(F + 1)N_r$. This scheme increases the chance that receivers will get unique FEC-encoded repair packets. Note that the sender would not transmit more than B repair packets, and every node uses the same FEC-encoding scheme.

Feedback In the dynamic mode, the sender needs feedback on the number of FEC repair packets that each receiver requires. In SRM, the feedback has to be directly sent to the sender since there is no mechanism to fuse the feedback on the path to the sender in order to decrease the feedback traffic at the sender. Thus, we need some scheme to reduce this implosion at the sender. Intuitively, we define *implosion* to be the sender's computational load to process a larger amount of back traffic than the sender would get in a point-to-point "high rate" transmission such as a TCP connection over a high speed Ethernet.

To reduce implosion, the time interval between two consecutive transmissions of feedback by a receiver has to be properly adjusted so that the feedback received by the sender in a given interval is no more than it would receive if engaged in a TCP unicast of a high transmission rate. Modern workstations can handle more than a 1 MB/s transmission rate. This implies that these workstations handle over 1000 acknowledgments per second, assuming that one acknowledgment is sent per packet of 1 KB.[2] We assume that the sender is capable of handling this feedback rate. In a 500 node session, this interval translates into 500 ms feedback interval for each receiver. A receiver randomizes its transmission time within the interval to reduce unwanted synchronization.

2.2.5 Integrating with a tree-based protocol

MLR can be combined with a tree-based protocol such as RMTP[2] or LBRM[8]. As discussed in the introduction, a tree-based protocol solves implosion and repair locality by imposing a logical tree structure, and designating a receiver — called the *designated receiver* (DR) in RMTP — as the root in the sub-trees of the tree.

A tree-based protocol can benefit from MLR. Because MLR multicasts FEC-repair packets, FEC repair packets can proactively suppress many repair requests. In addition, since MLR allows receivers to choose the amount of FEC repair packets to receive proactively, it reduces much of the repair locality problem. Since the implosion and repair locality problems become less limiting by the use of MLR, the tree-based protocol can scale to incorporate a larger fanout, resulting in a reduced number of DRs.

[2]Some TCP implementations allow one ACK per two packets.

As in the SRM case, the recovery technique of the tree-based protocol being combined with MLR is used when a receiver cannot recover a block from FEC repair packets. The sender multicasts data and repair packets, and receivers select repair groups as described in Section 2.2.1. When a receiver detects the failure to recover a block, it transmits a request (or NACK) to its DR. The DR immediately retransmits the requested packet. In tree-based protocols, we allow DRs to retransmit original data packets. In some tree-based protocols, DRs actually unicast repair packets to their child receivers [2] when there are not many requests for the same packet pending. When packets are unicasted, the repair locality problem is not the key concern.

The tree structure of the protocol actually simplifies aspects of MLR e.g., in gathering feedback, and by decreasing the sender's load in processing feedback from receivers. A DR can gather feedback, compile the statistics on the number of receivers in its subtree and the required number of repair packets, and report that to its parent DR. Thus the sender receives only a small number of feedback messages from its children.

3 Low-overhead network delay estimation

In this section, we focus on the problem of estimating the network delay between receivers. Network delay is an important parameter that is used in request and repair suppression, congestion control etc. Our motivation is its role in reliable multicasting. A crucial point in what follows is that the clocks at the receivers (or the sender) are not synchronized; this is realistic.

3.1 Estimating network delays in a single scope

The problem we address in this section is to estimate the delay incurred by a transmission from any receiver (or the sender) to any other receiver or the sender, all in a single scope.

Clearly the delay values are dynamic, changing with the traffic pattern and congestion in the network. In fact, these values may be significantly affected by the overhead of the traffic generated by the protocol that performs the estimation by actively injecting packets into the network; hence, any such protocol must minimize this traffic overhead. Another motivation for reducing the traffic overhead is to increase the scalability of the estimation process, since it has to be run frequently to be up-to-date.

The existing protocols for estimating these network delays require $\Theta(n^2)$ messages of $O(1)$ size each, or $\Theta(n)$ messages of size $\Theta(n)$ each [23, 7]; here, n is the number of receivers within a scope (or within the entire multicast group as in the case of SRM). In this section, we present a protocol that requires only $\Theta(n)$ messages of size $\Theta(1)$ each. In Section 4, we demonstrate the accuracy of this estimation protocol.

We use the same accounting scheme as is currently employed in the literature: any unicast or multicast transmission is counted as one message. We also assume that the network delays are symmetric, that is, the delay from receiver R to Q is the same (or nearly the same) as that from Q to R; this is also standard in the literature. As needed, our protocol does not require synchronized clocks on the network, not the knowledge of the network topology.

The protocol consists of two phases: *setup* and *estimation* phase. We describe each phase in detail now.

Setup phase. In this phase, each receiver R determines the network delay $d(S, R)$ from the sender S to itself. In order to do this, each receiver sends its current time in message to the sender. The sender merely

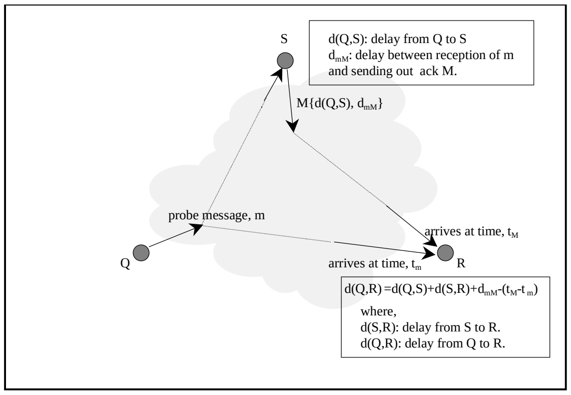

S
d(Q,S): delay from Q to S
d_{mM}: delay between reception of m and sending out ack M.
M{d(Q,S), d_{mM}}
probe message, m
arrives at time, t_M
Q
arrives at time, t_m
R
d(Q,R) =d(Q,S)+d(S,R)+d_{mM}-(t_M-t_m)
where,
d(S,R): delay from S to R.
d(Q,R): delay from Q to R.

Figure 4: The delay estimation phase

sends the same message back to each of the receivers. By comparing the time this message is received with the time stored in the message, each receiver can compute the delay $d(S, R)$. As stated, this phase uses at most $2n$ messages each of $O(1)$ size (assuming the granularity of time scale to be micro-seconds, the size can be as large as four bytes). This phase can be simplified by letting the sender multicast a message containing the concatenation of all their message contents back to the receivers. This requires n messages of size $O(1)$, and one message of $O(n)$.

Estimation phase In this phase, each receiver R determines the delay from every other receiver Q to itself; here, $d(S, R)$ is used crucially.

Figure 4 illustrates this part of the protocol. *Fix* a receiver Q. We focus on the problem of each receiver R estimating the delay of transmissions ¿from Q. Receiver Q multicasts a *probe* message m containing $d(A, S)$ and the (local) time it sends m. Say the time this message is received by a receiver R is t_m. The sender, upon receiving m, multicasts a message M comprising d_{mM}, the delay between receiving m and sending out M. Say M is received by R at time t_M. The following holds.

Lemma 3.1 $d(Q, R) = d(Q, S) + d(S, R) + d_{mM} - (t_M - tm)$.

Proof. Let t be the time (in R's clock) that Q sent m. We can estimate the time t in two different ways. Since m was received at R at time t_m, $t = t_m - d(Q, R)$. Also since M was received at R at time t_M, $t = t_M - d(S, R) - d_{mM} - d(Q, S)$. Both estimates of t must be identical, hence $t_m - d(Q, R) = t_M - d(S, R) - d_{mM} - d(Q, S)$ ¿from which the lemma follows. ∎

Thus every receiver R can compute $d(Q, R)$ for a fixed Q. This takes 2 multicast messages each of size $O(1)$. Now, we repeat this process with each receiver R playing the role of Q; at the end of this, every receiver R has $d(R, Q)$ for each Q. The whole process requires $2n$ multicast messages each of size $O(1)$. We make two remarks.

Remark 1. One can optimize the number of messages by allowing the sender to collect the probe messages (m's) from *all* the receivers, and to multicast a cumulative acknowledgment M to all the receivers that

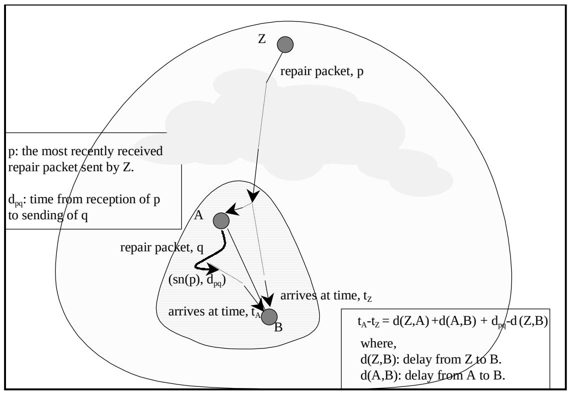

Z
repair packet, p
p: the most recently received repair packet sent by Z.
d_{pq}: time from reception of p to sending of q
A
repair packet, q
(sn(p), d_{pq})
arrives at time, t_Z
arrives at time, t_A
B
$t_A - t_Z = d(Z,A) + d(A,B) + d_{pq} - d(Z,B)$
where,
d(Z,B): delay from Z to B.
d(A,B): delay from A to B.

Figure 5: Hierarchical delay estimation

contains d_{mM} for all m's. This requires n probe messages each of size $O(1)$ and a single multicast message from the sender of size $O(n)$.

Remark 2. The overall protocol is to run the setup and estimation phases alternately. By slightly modifying the estimation protocol, we can ensure that following a *single run* of the setup phase, repeatedly running the estimation protocol itself suffices. The receiver Q now sends the message m comprising (a) the time the sender sent the last packet received by Q from the sender (this information can be obtained from the packet) and (b) the delay between when the last packet from the sender was received by Q and the current time when Q sends message m. The sender can determine $d(Q, S)$ from this information. Sender now broadcasts $d(Q, S)$ and d_{mM} as before, and the rest follows easily. The message complexity remains unchanged. The pseudo-code for this protocol appears in the Appendix.

3.2 Estimating network delay in a hierarchical scope

Our discussion on network delay estimation in Section 3 focuses only on how to estimate delays between receivers within a single scope. The protocol there relies on the fact that the feedback from the receivers is multicasted to the entire scope. However, in presence of hierarchical scopes, a receiver does not send feedback to the ZCRs of its outer scopes. SHARQFEC [3] deals with this difficulty by assuming that ZCRs are at the multicast routes from the sender, and successively adding delays between a parent ZCR and its child ZCR. However, this assumption may not hold true if ZCRs are elected based on a challenge protocol. This is because the closest receiver to the sender in a scope may not be always located at the routing path from the sender to all the receivers in the scope.

In this section, we present a protocol that estimates the network delay from a sender Z to its receivers without any topological assumption or synchronized clocks. No feedback from the descendent receivers other than ¿from the immediate children of Z is necessary. As before, we assume that network delays are symmetric.

The protocol applies the principle discussed in Section 3 recursively in the hierarchical setting. Figure 5 illustrates the intuition behind the protocol. The objective is to estimate delays from a ZCR Z to a receiver B ($d(Z, B)$). Suppose that receiver A is the immediate parent of B, and A is a descendent of Z. Note that Z

multicasts repairs (or data if the sender) to both A and B through the base group of Z. This happens when Z is the ZCR of a nesting scope of the scope of A and B. If it is not in a nesting scope, we do not need to measure the network delay between Z and B. The delay between A and B ($d(A, B)$) can be computed as described earlier (i.e., B includes in the feedback the time stamp in the last message received ¿from A).

Suppose that receiver B knows the delay from Z to A ($d(Z, A)$). Then $d(Z, B)$ can be estimated by B as follows. First Z multicasts a repair packet p to its base group to which both A and B must belong. Let t_Z be the time that B receives this packet according to B's clock. When A receives p, it multicasts packet q containing the sequence number of the last packet it received ¿from Z and the delay from the reception of that packet to the sending of q which is denoted d_{pq}. Let t_A be the time that B receives packet iq from A. We have the following: $t_A - T_Z = d(Z, A) + d(A, B) + d_{pq} - d(Z, B)$. It follows that, $d(Z, B) = d(Z, A) + d(A, B) + d_{pq} - (t_A - T_Z)$.

Now the remaining issue is ensure that receiver B knows the time delay between Z and A. Note that all the ancestor ZCRs of B (including A) can estimate the delay from Z by applying the same argument recursively while the delay between Z and its immediate child receivers (which is a base case in the recursion) can be estimated in the usual way. Thus, A knows the delays from all of its ancestor ZCRs of its nesting scopes. This information can be piggy-backed on the repair packets it multicasts to B. This information would occupy no more than $O(h)$ space where h is the height of the tree of scopes. This does not add much overhead.

4 Simulation

We implemented our MLR protocol using the UCB/LBNL/VINT network simulator `ns`. We incorporated it into three well-known protocols, namely, the basic SRM, a hierarchical SRM, and a tree-based protocol; we compared their performance to that of SRM [1], SHARQFEC [3], and ECSRM [23] respectively. We also implemented our delay estimation protocol, and studied the performance of SRM with this new protocol. Our overall experimental setup is very similar to the one in [3].

Topology. Our simulation experiments were run using variants of the hybrid mesh tree topology used in [3]. The two topologies used in our experiments are shown in Figure 6. their configuration is identical to the ones in [3] except for the loss rates assigned to each link. The sender at node 0 feeds data to a three level hierarchy of 112 receivers arranged as a mesh of 7 receivers each of which feeds balanced trees. Each of seven trees in the topology is an exact copy of each other, and three subtrees within each tree are also a copy of each other. The links connecting the source to the top 7 nodes in each tree are 45Mbits/s with all other links set to 10 Mbits/s. The latency between any two receivers located within each tree was set to 20 ms for each link while the latencies used for the backbone links are shown in the figures. The loss rates for the links are varied over different parts of the networks.

The loss model. To simulate a realistic loss behavior, we conducted transmission experiments over a transpacific link every 45 minutes between Oct. 10 and Oct. 13, 1998, and recorded all the packets being received and lost. We gathered over 100 traces each 15 minutes long, and extracted the profile information of each trace which comprises the loss characteristics of every non-overlapping 300 ms segment. The loss characteristics include the number of instances of loss bursts of lengths from 1 to over 200. For each link

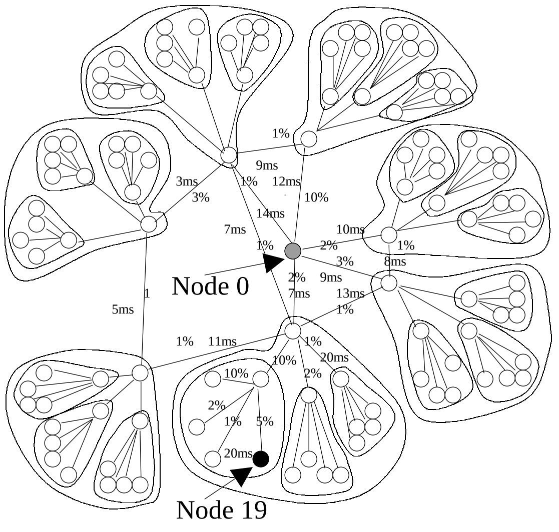


Figure 6: Network simulation topology with 113 nodes

in the networks shown in Figure 6, we find a trace that undergoes the same average loss rate as that of the link, and use the trace to pick packets to drop during each 300 ms period. The loss rates for links are shown in Figure 6. The loss rates that receivers experience can be obtained by compounding the loss rates on the links from the sender to the receivers. In our topologies, they vary from 1% to 27.5%. Every packet passing through a link — data, repair, request, and session — is subject to the same loss rate indicated on that link.

Transmission. Each simulation experiment starts the session at time 1 second, at which time nodes begin sending session messages, and after the initial bootstrap phase of 6 seconds, node 0 starts sending traffic at a constant bit rate of 800 Kbits/s. Each data packet is 1024 bytes. The sender stops transmitting data packets at time 16 seconds. The block size B is 16, and for all MLR experiments, unless specified, K is set to 5, and MLR uses the dynamic mode. Note that at this transmission rate, each receiver will get approximately 10 packets over a 100 ms period. In MLR, every receiver determines the required number of FEC packets based on 5% bandwidth wastage threshold as described in Section 2.2.3.

Parameters of interest. We focus on three categories of traffic: *data* (this comprises the original data packets), *proactive* (this is the traffic transmitted by the sender over and above the data packets without an explicit request from the receivers), and *reactive* (this is the traffic introduced by the sender or other receivers in response to the retransmission requests). The total redundant proactive (reactive) traffic is the total number of proactive (reactive respectively) packets that reach the receivers in excess of their requirements; total redundant traffic includes both. All ratios and percentages are with respect to the total data traffic received by all the receivers.

Experimental results. We present a selection of experimental results to support key observations on the performance of MLR as well as on the impact of our delay estimation protocol.

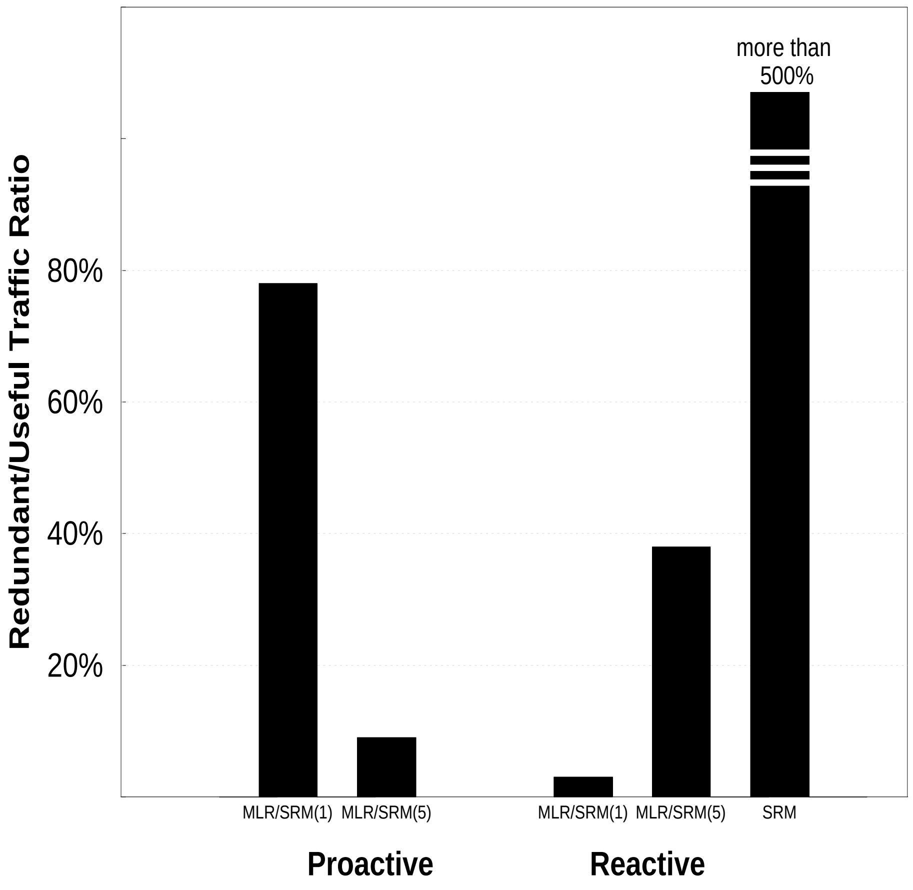


Figure 7: Impact of layering on proactive repair transmission

4.1 The performance of MLR

Impact of layering on proactive repair transmission. We measure the impact of MLR in proactive FEC transmission schemes. We tested single-scoped SRM, MLR/SRM with one repair group (denoted MLR/SRM(1)), and MLR/SRM with five repair groups (denoted MLR/SRM(5)). The result of the simulation runs are shown in Figure 7. All transmission tests finished approximately at the same time.

First, since SRM does not add any FEC packets, its redundant proactive traffic is zero. However, its reactive traffic soars to at least 500%. This is because each repair packet is multicasted to the entire session ([3] also reports a similar performance for SRM). Second, while MLR/SRM(1) shows about 78% proactive redundant traffic ratio, MLR/SRM(5) shows only 9%. This shows that MLR is effective in allowing receivers to reduce their incoming proactive traffic by selectively joining repair groups. However, MLR/SRM(1) has 3% redundant reactive repair traffic ratio while that of MLR/SRM(5) is 38%. This is because MLR/SRM(1)

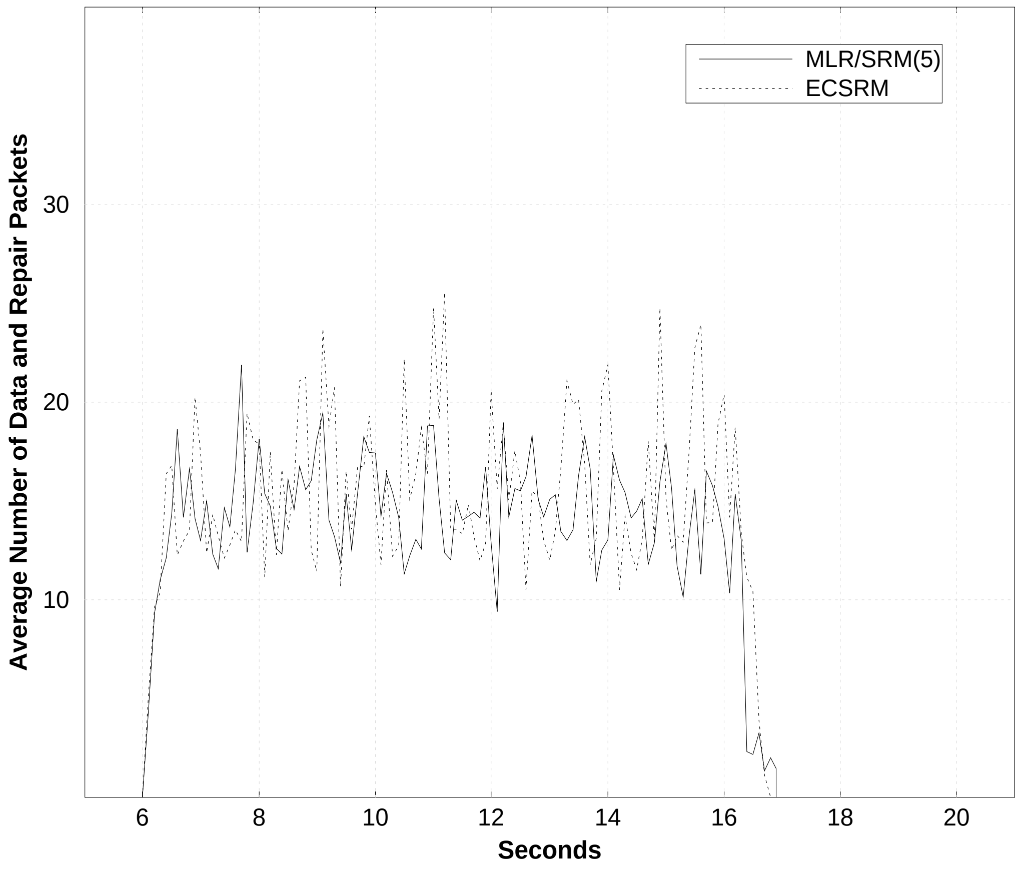


Figure 8: The performance comparison of MLR/SRM, and ECSRM

aggressively subjects receivers to a large amount of proactive traffic and successfully reduces the chance that further repair packets are needed. However, the total redundant traffic ratio for MLR/SRM(1) is 81% which is 30% more than that for MLR/SRM(5). Thus multi-layering reduces the overall redundant traffic substantially.

Comparison of FEC recovery techniques. We now compare the performance of MLR with that of other existing FEC recovery schemes. Figures 8, 9, and 10 show this comparison in terms of the average number of packets (of all categories stated above) per receiver received during each 100 ms period. In all simulation tests, the transmission finished at approximately the same time for all the protocols; thus, their throughput is similar. However, they differ in how efficiently they use the bandwidth as described below.

We first compare the performance of MLR/SRM with that of ECSRM [23]. ECSRM is a tree-based protocol that is a version of SHARQFEC where scoping and proactive repair are turned off, and only the

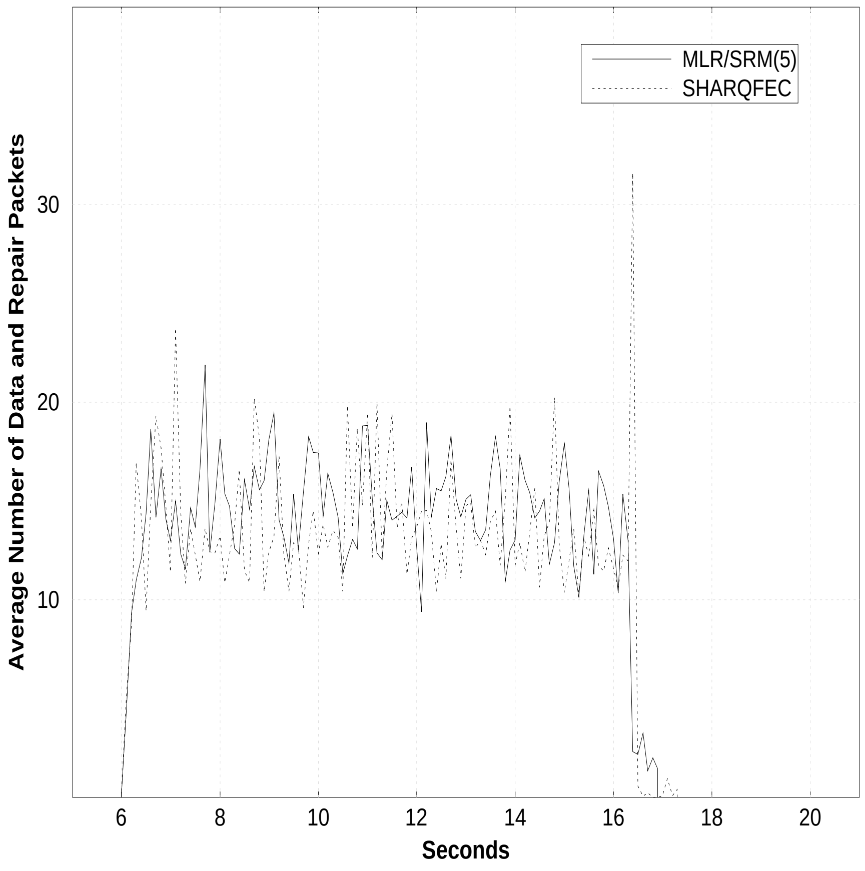


Figure 9: The performance comparison of MLR/SRM, and SHARQFEC

sender participates in reactive repair. From Figure 8, one can do a calculation to conclude that ECSRM has about 30% more redundant traffic than MLR/SRM. This is because ECSRM multicasts reactive repair packets, and has poor repair locality. However, the performance of ECSRM is much better than that of the basic SRM we quoted earlier because ECSRM uses reactive FEC repair and also does not allow receivers to participate in the repair. Thus, duplicates in repair transmission are substantially reduced.

We then compare the performance of MLR/SRM to SHARQFEC in Figure 9. SHARQFEC forms three-level hierarchy with 29 nesting scopes from the simulation topology in Figure 6 where each bounding circle represents a separate scope, and the roots of 7 trees forms another scope. The root of all the trees in the topology and the sender are ZCRs. SHARQFEC generates only 15% less redundant traffic than MLR/SRM. This result is very encouraging for MLR/SRM because it is only a single scoped protocol while SHARQFEC is extensively scoped. Since SHARQFEC uses 29 scopes of less than five members, the simulation experiment shows excellent repair locality. However, these scopes come with additional cost of

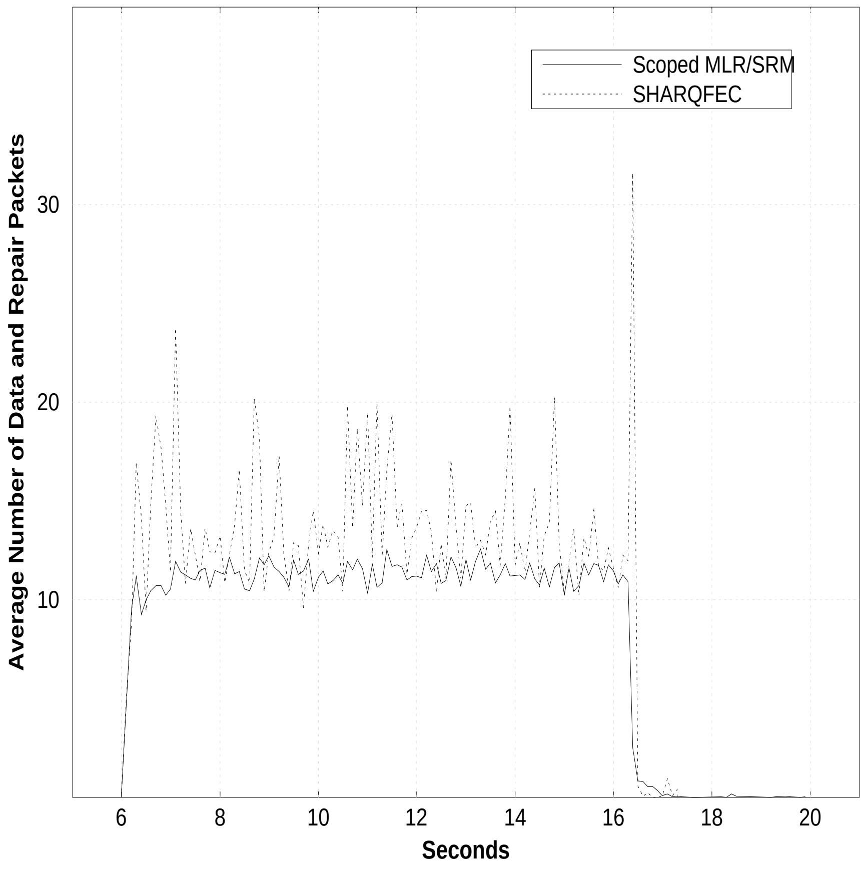


Figure 10: The performance comparison of Scope MLR/SRM and SHARQFEC

maintaining the scopes and ZCRs. This result suggests that MLR can enhance the repair locality of SRM up to the level comparable to that of SHARQFEC.

Figure 10 compares the performance of SHARQFEC with scoped MLR/SRM. We allow MLR/SRM to have the same scopes as SHARQFEC in the same topology. The total traffic in scoped MLR is far less than that of SHARQFEC. Overall, scoped MLR has only 19% redundant traffic while SHARQFEC has about 40% redundant traffic. This result strongly suggests that when combined with hierarchical scoping, MLR can achieve excellent repair locality.

Dynamic vs. static protocols. The optimal layering protocol determines the number of FEC packets to be sent to different multicast groups in a dynamic manner. We also discussed the static protocol in Section 2 where the maximum number of FEC packets to be sent is divided equally amongst the multicast groups. Static protocol has simple control while the optimal protocol needs more sophisticated control. Here we

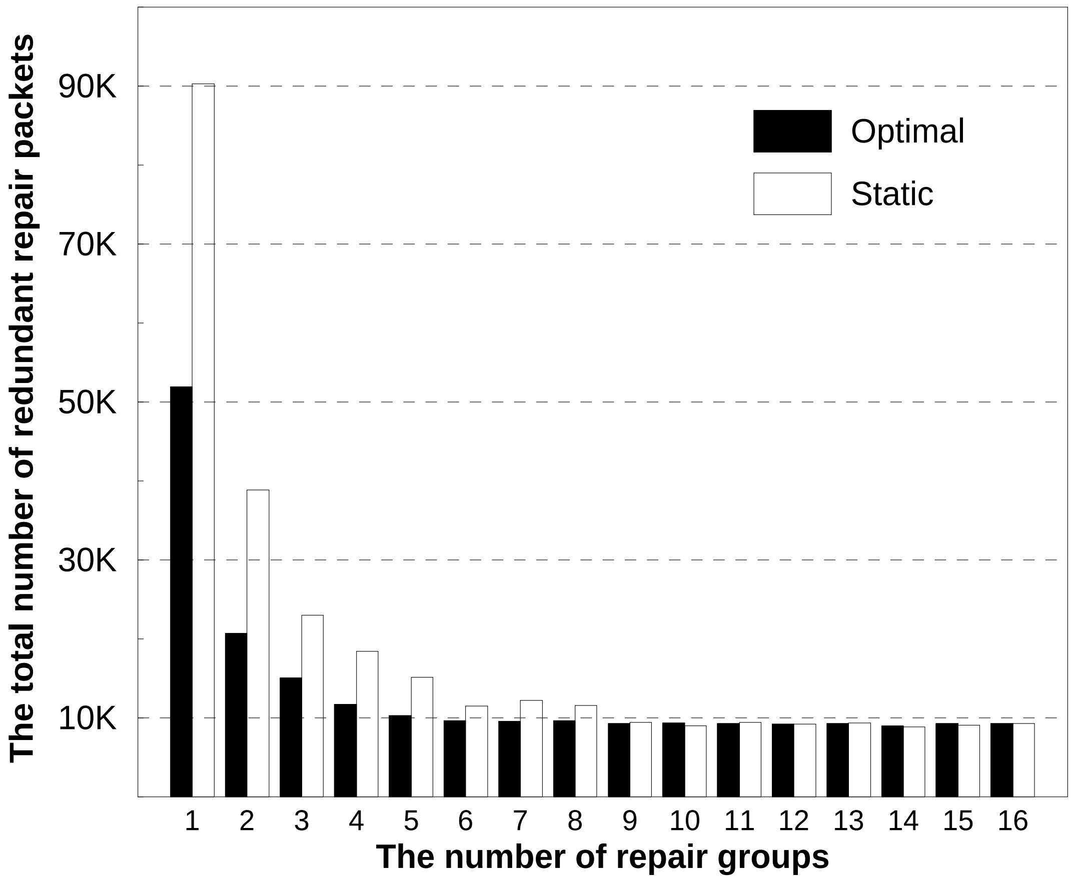


Figure 11: The performance comparison of static and optimal allocations of repair packets

compare their performance in Figure 11. The performance figures therein have to be interpreted carefully because many obvious improvement of the static protocl are possible.

We ran MLR/TREE over various numbers of repair groups on the same topology to see the effect of the dynamic allocations of repair packets using our optimal algorithm on the redundant repair traffic. We chose MLR/TREE because it does not generate any global reactive repair traffic, and thus it is suitable for studying the effect on proactive repair traffic. Figure 11 shows the total number of redundant repair packets received by all receivers during a simulation experiment. Recall that the static allocation (i.e., in the static mode) assigns $\lceil B/K \rceil$ FEC repair packets to each repair group regardless of the current loss rates of receivers. Under the loss rates assigned to the topology, approximately 8 to 10 repair packets per block are sufficient for the worst case receiver to recover its losses. That is, $f_{max} \leq 10$.

The adverse effect of the static allocation is evident when only a small number of repair groups are available. For example, when only one repair group is available, the static protocol multicasts all B repair

Time(sec)	Our Protocol	SRM
1	9056	753792
2	25824	1710528
3	24448	1652544
4	23840	1594560
5	24576	1623552
6	24960	1710528
7	24160	1609056
8	23648	1638048

Table 1: Bandwidth (bits/sec) used for delay estimation protocols measured at each second of simulation

packets per block to that repair group while the dynamic protocol multicasts less than 10 repair packets per block. Thus, in the static protocol, many receivers with low loss rates are subject to many redundant repair packets. However, when the number of repair groups gets larger than $B/2$ (currently 8), then the performance advantage of the optimal protocol quickly diminishes, favoring the static protocol because of the overhead of the optimal protocol involved in collecting feedback from all receivers.

4.2 Delay estimation simulation

We implemented our delay estimation protocol and compared its overhead and performance with that of SRM's session protocol. We use the topology shown in Figure 6 for simulation experiments. Table 1 compares the bandwidth overhead of our protocol with that of the SRM session protocol. Here the time interval between two consecutive transmissions of delay estimation probe (or session message) by the same receiver is set to one second. Our protocol uses only about 20 Kbits/s while the SRM session protocol uses over 1.6 Mbits/s. This is because the size of session messages in SRM is proportional to the number of receivers in the session, and each receiver sends at least one message per second. In contrast, our protocol uses only a constant size message.

To see the the accuracy of our delay estimation, we measure the ratio of actual network delays experienced by each packet to our estimated delays. For this purpose, we randomly picked node 19 in the topology and measured the delays that node 19 experiences and compared its estimated delays. The actual delays can be computed using time stamps embeded in each packet. Since the simulation, uses synchronized clocks, actual delays can be easily computed by taking a weighted average of each sample. Figure 12 shows that our estimated delays are well within a few percents of the actual delays.

We also incorporated our protocol into SRM: we modified SRM to use our delay estimation to set its suppression timers. Figures 13 and 14 compare the numbers of request and repair packets per receiver for the original SRM and that for the modified SRM. As the graphs indicate, using our scheme does not change the request and repair message traffic by a significant amount.

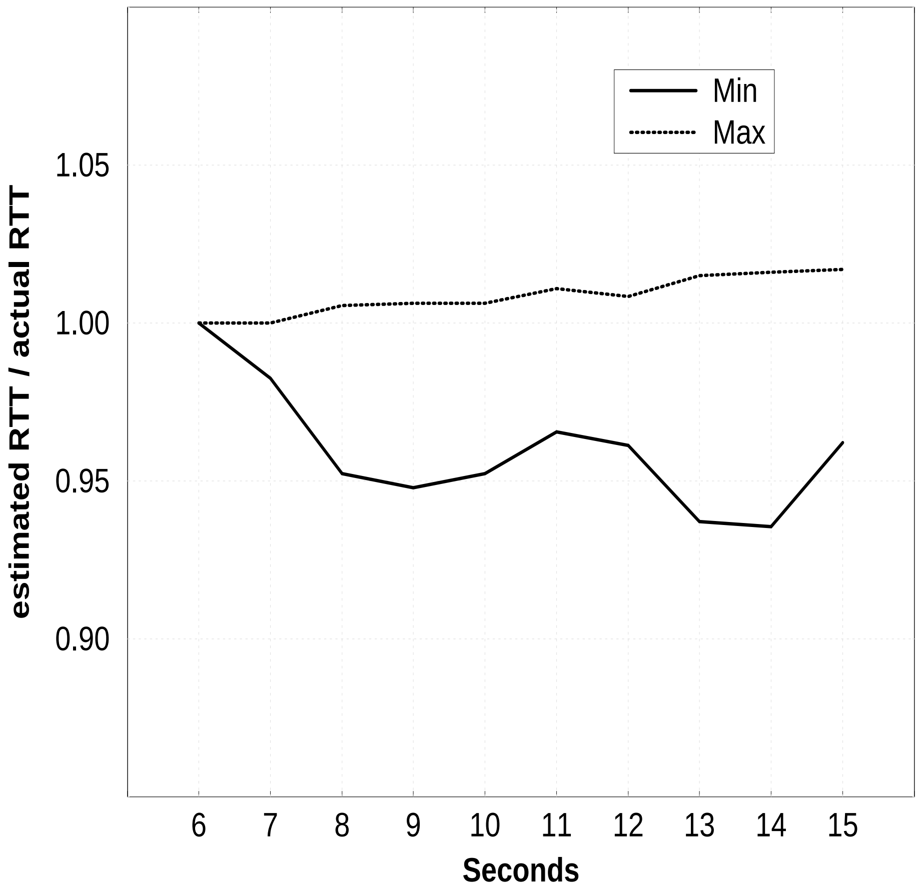


Figure 12: Ratio of estimated RTTs over actual RTTs

5 Related Work

In this section, we review related work in each of the three categories of relevance, namely, multicasting using multiple groups, use of FEC in reliable multicasting, and estimating network delays.

Use of multiple multicast groups. It is a natural idea to consider using multiple multicast groups for reliable multicasting, and it has appeared before. Previously, it has been applied to congestion control [24, 25, 26, 27] and error recovery [28]. Layering has the potential to work better for loss recovery rather than congestion control because loss recovery does not need any synchronization with other receivers on the common path. The known uses of multiple multicast groups differ from our MLR in the particular layering technique and in their specific applications. In particular, none of the prior work considers the dynamic allocation of repair packets to different multicast groups to optimize repair locality, and do so in a provably

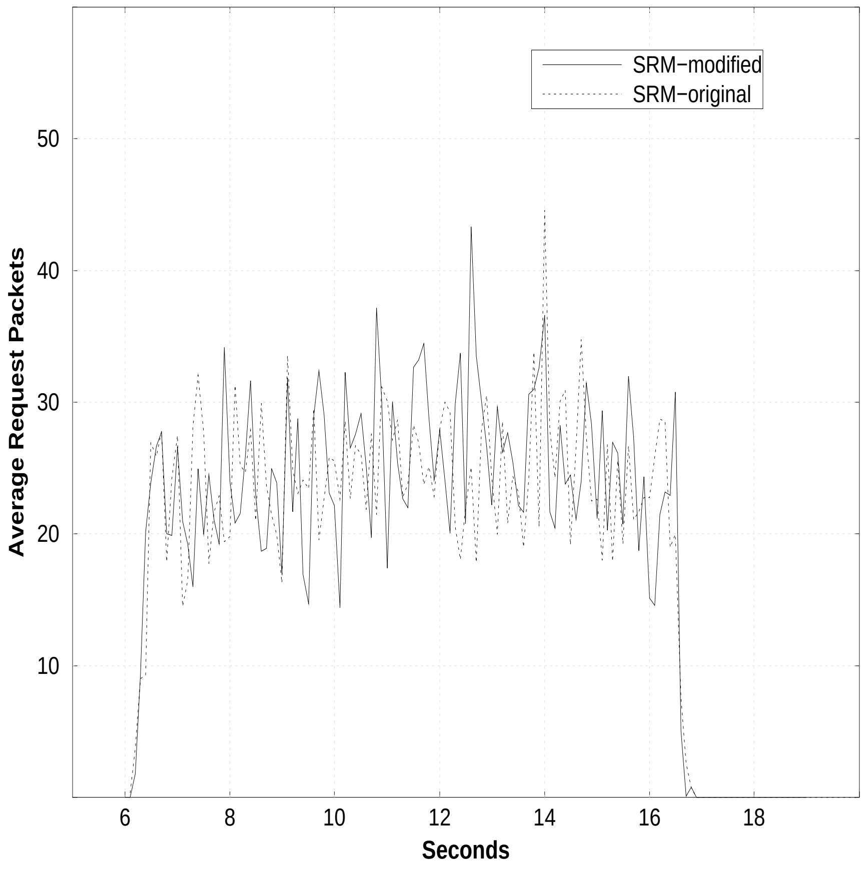


Figure 13: The impact of our protocol on the request suppression

optimal manner as we do.

In what follows, we review previous applications of multiple multicast groups. Ammar and Wu [24] first applied the idea of *destination set grouping* for improving fairness among receivers with different capabilities. Their scheme divides receivers into groups with similar capabilities; in each group, the sender transmits data at a suitable rate. Later, Cheung *et al*. [25] extended the work for multicasting real-time video. In both cases, the receivers do not belong to more than one group. McCanne *et al*. [26] applied a technique called *Receiver-driven Layered Multicast* (RLM) to control congestion in real-time video transmission. The sender multicasts different layers of video signals to different multicast groups. Each receiver chooses a subset of multicast groups and controls the amount of its incoming traffic. RLM and MLR are similar since they both allow receivers to adjust the amount of incoming traffic based on receivers' capability (be it loss rate or its power), but they differ in crucial ways. First, MLR is applied to error recovery whereas RLM is applied to congestion control; also, MLR layers FEC repair packets while RLM layers video data. Hence,

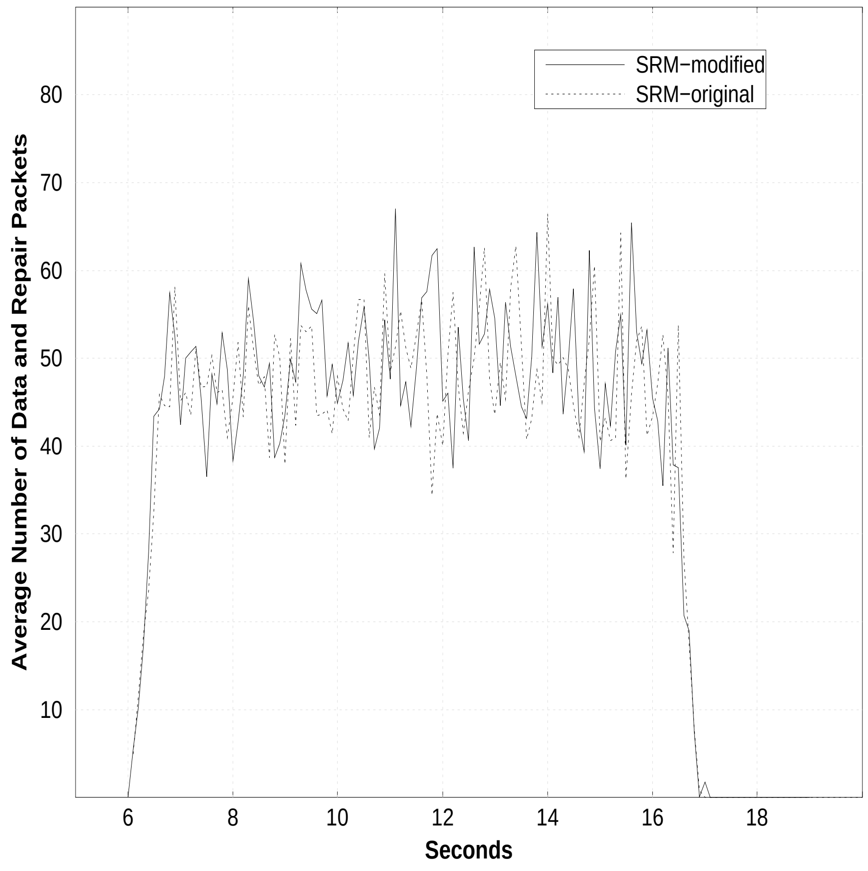


Figure 14: The impact of our protocol on the repair suppression

the optimization concerns are very different. Second, MLR has a provably optimal way to layer the number of FEC packets while RLM, as it stands, does not have a provably optimal layering strategy. Nevertheless, MLR may be considered as an example of a technique similar to RLM applied successfully to error recovery.

Vicisano *et al*. [27, 29] also developed a technique to layer bulk data using linear block coding, and applied it to reliable multicast for error recovery and congestion control. The technique is applicable primarily when a large portion of the data is available for encoding prior to transmission. The amount of redundant data in each multicast channel is statically allocated, and it is exponentially spaced amongst channels. Their layering technique replicates data periodically over a fixed time interval, called the *window*, while keeping every packet within a unique window. Thus, a packet lost in a window can only be recovered from the subsequent windows in the same multicast channel. Therefore, it seems best suited for delay-insensitive applications. In contrast, MLR uses a very different layering technique where the amount of data transmitted to each group is dynamically allocated to minimize the redundant repair traffic. Combined with an ARQ technique, MLR can easily accommodate delay-sensitive applications.

A work closely related to MLR is also in [28]. Like us, the authors in [28] use multiple multicast groups to solve the repair locality problem in sender-centric retransmission protocols. When detecting a loss of a packet p_i, receivers multicast NACK, and join a retransmission group g_i. The sender retransmits p_i to g_i. After receiving p_i, the receivers leave g_i. Since receivers join only the retransmission groups which carry their missing packets, this technique achieves excellent repair locality. However, it requires a receiver to join and leave a group for every data packet loss, thus potentially creating unreasonably large membership control (IGMP) traffic. Therefore, without proper router support, this technique may not scale well.

Application of FEC to reliable multicast Nonnenmacher *et al*. [11] studied a hybrid FEC/ARQ techniques for loss recovery in reliable multicast. They showed that a hybrid FEC/ARQ can significantly reduce bandwidth overhead of a large-scale reliable multicast. They also compared a layered implementation of the technique where FEC and ARQ are supported at different system layers with non-layered, combined implementation, and analytically showed that the combined approach yields more efficiency in the use of bandwidth. Recent studies [12, 13] also show that a hybrid technique can yield high performance when combined with local distributed recovery as in tree-based protocols. However, they report that FEC-based recovery diminishes the performance advantage of local recovery over sender-oriented recovery. Their work analytically shows that FEC can significantly improve the scalalability of reliable multicast. However, they have not considered the effect of multicasting FEC packets to repair locality.

Rubenstein *et al.* [14] proposed several protocols using proactive FEC transmission for real-time reliable multicast. Their protocols, based on rounds of transmission and feedback, rely on feedback from receivers to determine the number of FEC packets to add at each round. In each round, the sender transmits the maximum number of packets requested by the receivers as in [11]. Clearly this layering differs from ours substantially. Gemmel[23] applied a protocol similar to the one in [11] to SRM. In the resulting protocol, receivers multicasts NACKs which are used to suppress other requests. Unlike SRM, the protocol allows only the sender to respond to requests and multicasts a FEC encoded packet. This technique was later incorporated into SHARQFEC [3]; we already discussed the relation between our work and SHARQFEC.

To summarize, FEC is very effective in handling uncorrelated losses. Thus combined with retransmission ARQ techniques which are more effective in handling correlated losses, FEC can be a viable solution for error recovery in reliable multicast. Earlier study in hybrid ARQ techniques provides strong support for this. However, although the study indicates that FEC can achieve high performance in large-scale global multicast, no protocol has been developed to localize repair traffic under global multicast. Our MLR shows strong potential in this direction.

Delay estimation Sharma *et al.* [7] propose to impose a self-configurable hierarchy on SRM to solve the scalability problem induced by session traffic involved in estimating the pairwise network delay. The session members are divided into local scopes each of which contains a local representative. Local members within a scope send session messages to each other and conduct delay estimation among themselves while representatives perform their own delay estimation among each other. Delays between two members in different scopes are approximated via delays to their representatives. Under a small (local and global) scope comprising 20 to 40 members this technique is shown to reduce the bandwidth utilization of the delay estimation protocol.

Although hierarchical scoping helps increase the scalability of the delay estimation protocol in SRM, it still does not remove the $O(n^2)$ performance bound (hence, the limit on the scalability) where n can be the size of a scope. Thus, only small scopes can be accommodated. Our delay estimation protocol has a better performance bound, can be incorporated into many protocols that require scalable, non-intrusive delay estimation.

6 Conclusion

We study the scalability of reliable multicast protocols. Some of the fundamental known techniques for devising scalable reliable multicast protocols include receiver-centric repair [1], scoping [3], suppressing unwanted traffic by using timers [1], use of FEC packets [] etc. In this paper, we offer two additional tools, namely, an efficient algorithm to transmit the optimal number of FEC repair packets in layers so as to minimize the total excess repair traffic and a low-overhead protocol for estimating network delay between the receivers. We employ both these tools on existing protocols and present simulation results that show the combined protocols to be substantially more scalable.

Many aspects of our work remain to be refined further and studied experimentally, or by building a testbed. For example, in a hierarchical scope, a receiver may receive repairs from all of its nesting scopes. Thus, the repair group addresses of overlapping scopes have to be unique. Non-overlapping scopes do not need unique addresses since administrative scoping of multicast addresses [17] allows multicast addresses to be safely reused in non-overlapping scopes. Thus, as administrative scoping becomes widely deployed, the number of multicast groups required for hierarchically scoped protocols grows linearly with the nesting level of the most deeply nested scope; it is desirable to decrease this number. As SHARQFEC [3] has shown, five level nesting scopes are sufficient for millions of receivers dispersed over the nation. Assuming that there are at most 10 multicast group in each scope, we need at most 50 multicast groups. By employing non-nested scopes for some strategic locations (where it is certain that ZCRs are placed at the routing paths), the nesting level of hierarchical scopes can be further reduced. The overall effect of the nesting level remains to be understood. Another example is to determine the appropriate interval between feedbacks. This depends on the ability to predict the loss patterns, and insights into this process will greatly help our protocols. Yet another example is to understand the effect of loss of RGI (repair group information), and the delays associated with joining and leaving multicast groups. Some of these issues are the subject our current and future study.

References

[1] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of the ACM SIGCOMM Conference*, pages 342–356, October 1995.

[2] S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya. Reliable multicast transport protocol (RMTP). In *Proceedings of the IEEE INFOCOM*, San Francisco, CA, March 1996.

[3] R. G. Kermode. Scoped hybrid automatic repeat request with forward error correction (sharqfec). In *Proceedings of the ACM SIGCOMM Conference*, pages 278–289, October 1998.

[4] R. Yavatkar, J. Griffioen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of ACM Multimedia*, 1996.

[5] Reliable multicast links: http://research.ivv.nasa.gov/rmp/links.html.

[6] M. Yajnik, J. Kurose, and D. Towsley. Packet loss correlation in the mbone multicast network. In *Proceedings of IEEE Globecom'96*, November 1996.

[7] P. Sharma, D. Estrin, S. Floyd, and L. Zhang. Scalable session messages in srm using self-configuration. In *USC Technical Report*, July 1998.

[8] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. In *Proceedings of the ACM SIGCOMM*, pages 328–341, August 1995.

[9] T. Speakman, D. Farinacci, S. Lin, and A. Tweedly. Pgm reliable transport protocol. In *Internet Draft*, August 1998.

[10] C. Papadopoulos, G. Parulkar, and G. Varghese. An error control scheme for large-scale multicast applications. In *Proceedings of the IEEE INFOCOM*, 1998.

[11] J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. In *Proceedings of ACM SIGCOMM*, September 1997.

[12] J. Nonnenmacher, M. Lacher, M. Jung, E. Biersack, and G. Carle. How bad is reliable multicast without local recovery. In *Proceedings of IEEE INFOCOM*, March 1998.

[13] J. Nonnenmacher M. Lacher and E. Biersack. Performance comparison of centralized versus distributed error recovery for reliable multicast. In *Submitted to Transactions on Networking*, August 1998.

[14] Dan Rubenstein, Jim Kurose, and Don Towsley. Real-time reliable multicast using proactive forward error correction. *NOSSDAV 98 (to appear)*.

[15] D. Rubenstein, S. Kasera, J. Kurose, and D. Towsley. Improving reliable multicast using active parity encoding services(apes). In *To appear in IEEE Infocom '99*.

[16] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing, extended report. September 1995.

[17] V. Jacobson. Administratively scoped ip multicast. July 1994.

[18] V. Jacobson. Congestion avoidance and control. pages 314–329, August 1988.

[19] D. Mills. Network time protocol(v3). April 1992.

[20] M. Handley. A congestion control architecture for bulk data transfer, the reliable multicast research group meeting in cannes. September 1997.

[21] D. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for the facility location problems. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[22] J. Bolot. End-to-end packet delay and loss behavior in the internet. In *Proceedings of the ACM SIGCOMM*, pages 289–298, San Francisco, CA, September 93.

[23] J. Gemmell. Scalable reliable multicast using erasure-correcting re-sends. In *Microsoft Research Technical Report, MSR-TR-97-20*, June 1997.

[24] M.H. Ammar and L. Wu. Improving the throughput of point-to-multipoint arq protocols through destination set splitting. In *Proceedings of IEEE Infocom*, pages 262–269, June 1992.

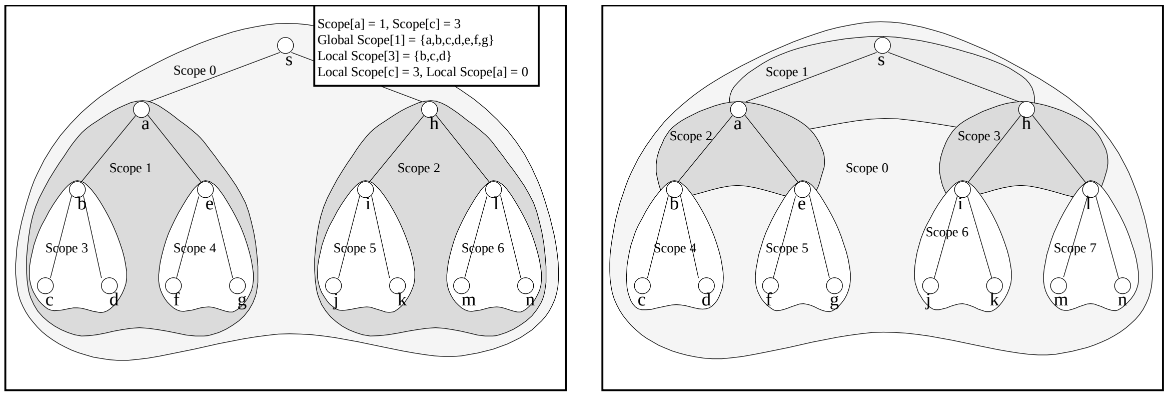
[25] S. Y. Cheung and M. H. Ammar. Using destination set grouping to improve the performance of window-controlled multipoint connections. Technical Report CC-94-32, Georgia Institute of Technology, Atlanta, August 1994.

[26] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proceedings of the ACM SIGCOMM*, pages 117– 130, Stanford, CA, August 1996.

[27] L. Vicisano, L. Rizzo, and J. Crowcroft. Tcp-like congestion control for layered multicast data transfer. In *Proceedings of the IEEE INFOCOM*, August 1997.

[28] S. Kasera, J. Kurose, and D. Towsley. Scalable reliable multicast using multiple multicast channels. In *CMPSCI Technical Report TR 96-73*, October 1996.

[29] Lorenzo Vicisano. Notes on a cumulative layered organization of data packets across multiple streams with different rates. January 1997.



(a) (b)

Figure 15: (a) Nested Scopes, (b) Non-nested scopes

A The hierarchically scoped MLR protocol

The preceding discussion of MLR protocol focused on the single scope case where the sender is the only node proactively injecting FEC repair packets. MLR can also be applied to hierarchically scoped protocols such as scoped-SRM [1] or SHARQFEC [3]. Hierarchical scoping is in general effective in localizing repair, thereby scaling reliable multicast to large number of receivers. The combination of MLR and hierarchical scoping can substantially increase this scalability.

In this section, we describe how MLR can be used in presence of scoping. Informally, this can be achieved by electing scope leaders in each local scope, and letting scope leaders proactively distribute FEC repair packets to different local multicast groups based on the feedback received from their own scopes. Receivers can listen to a set of local multicast groups to receive repair packets. There are many details and we describe them now.

A.1 Hierarchical Scopes

The entire multicast group is divided in to scopes as shown in Figure 15. Each scope is given a unique scope ID. Every scope except scope 0 has one *parent* scope, and may have one or more *child* scopes. Each scope designates the node closest to the sender to be the representative of the receivers in that scope; this representative is called the *Zone closest receiver* (ZCR). The sender becomes the ZCR of the scope 0. A receiver can be a member of one or more scopes. For instance, the ZCR of a scope is also a member of its parent scope. The dynamic election of a ZCR within a scope is described in detail in [3]; we adopt that procedure and do not describe it here any further.

Definitions. The following definitions are helpful in describing our protocol. The *scope of a receiver* r is i if r is a member of i and there does not exist a scope $j \neq i$ of which r is a member, such that j is a child scope of i. Thus, in Figure 15, a's scope is 1 while c's scope is 3. The *global scope* of a scope i includes every receiver that is a member of scope i. For example, in Figure 15, the global scope of 1 includes a, b, c, and d. The *local scope* of a scope i includes only those receivers whose scope is i (i.e., those belonging only to scope i). For example, in Figure 15, the local scope of 1 contains only a and b which are the ZCRs of the child scopes of scope 1. The *local scope of a receiver* is the scope of that node if that receiver is not the ZCR of the scope. Otherwise, it is its parent scope. Thus, the local scope of c is 3 while the local scope of a is 1.

Scope may be *nested* or *non-nested*. A nested scope is one that completely contains all of its child scopes. A non-nested scope is one whose child scopes do not include any receivers (with the exception of their ZCRs) that are members of their parent. Scope 0 is always a nested scope because it contains every descendent scope. All the non-ZCR receivers in a non-nested scope i belongs to only scopes i and 0. Thus, in a non-nested scope i, its global and local scopes are the same. Figure 15 shows examples of non-nested and nested scopes respectively.

A.2 The allocation of multicast groups

Each ZCR is allocated $K + 1$ unique multicast groups. The number of multicast groups allocated to each ZCR is determined at the beginning of data transmission based on the expected sizes of multicast groups and scopes. In each scope x, there are $g_{x,0}, g_{x,2}, \ldots, g_{x,K}$ multicast groups. We call $g_{x,0}$ the *base group* of scope x. The sender multicasts data packets to $g_{0,0}$ and all receivers join that group. Each receiver is a member of the base group of its own scope. ZCRs are members of the base group of the parent scope of their scopes. A local scope i is allocated one multicast group, called *local group* l_i, which comprises all receivers whose local scope is i, and the ZCR of scope i. The local groups are used for the dissemination of the feedback, and for retransmission request and repair packets.

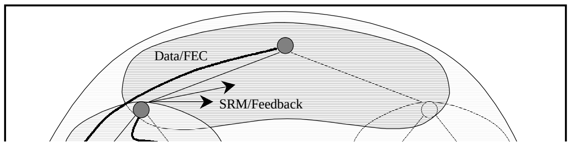
Each ZCR is responsible for multicasting FEC repair packets to its scope. Each ZCR at scope x multicasts FEC repair packets to multicast groups $g_{x,1}, \ldots, g_{x,K}$. Whether a scope is nested or not affects the MLR protocol (especially, that of receivers and ZCRs). We discuss the MLR protocols for non-nested scopes and nested scopes separately.

A.3 Non-nested scopes

Each receiver multicasts its feedback only to its local group and the feedback is identical to that in the single scoped protocol.

The ZCR of a scope responds to the feedback from its local group much the same way as the sender does in the single scoped protocol. More details are as follows. The ZCR computes the distribution of repair packets using the feedback received from the local group, and obtains $\phi_1, \ldots, \phi_K$. After the ZCR successfully receives each block of data, it multicasts $\phi_{i,j}$, $1 \leq j \leq K$, of uniquely FEC encoded repair packets to $g_{i,j}$. As before, there is a delay between successive FEC pockets in order to reduce short burst losses of FEC repair packets. The ZCR multicasts its session information (i.e., the information on the distribution of repair packets) to its base group and includes it in every repair packet it multicasts. The ZCR participates in the network delay estimation protocol for the receivers of its local scope by multicasting acknowledgment to probes (included in feedback) received from its local group.

Receivers behave much the same way as they do in the single scoped protocol. A receiver in scope x always obtains data packets from $g_{0,0}$ and the session information from $g_{x,0}$. Using the session information,



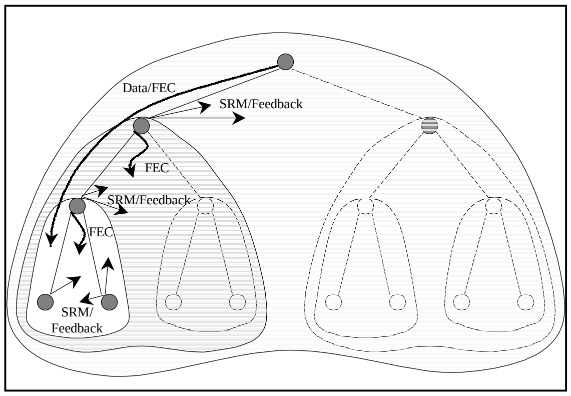


Figure 17: The extent of the multicasted information travels in nested scopes

a receiver finds the minimum r, $r \leq k$, such that the sum of $\phi_{x,1}$ to $\phi_{x,r}$ is at least as large as its currently required number of repair packets. It then joins multicast groups $g_{x,1}, g_{x,2}, \ldots, g_{x,r}$ for FEC repair packets.

Figure 16 depicts the extent of the multicasted information travels in non-nested scopes. Arrows crossing over two scopes indicate that information is multicasted to both scopes, and arrows contained in a scope indicate that information is multicasted only to that scope. Data packets are multicasted to every scope while FEC repair packets, retransmission requests and repairs are confined to local scopes.

A.4 Nested scopes

A scope can be nested in multi-levels. The *nesting level* of a nested scope i is the number of scopes containing scope i. We assume that if any two scopes j and k nest scope i, either scope j nests scope k or vice versa. Note that the global scope of the parent of a nested scope is different from the local scope of the parent because a receiver.

As in non-nested scopes, receivers multicast feedback only to their local repair group, and based on the feedback, the ZCR of that scope determines the number of FEC repairs that it proactively transmits to its global scope. This number also depends on how many repairs it receives from its nesting-ancestor scopes. Figure 17 depicts the extent that messages travel to in nested scopes. While FEC repair packets multicasted to global scopes, retransmission requests and repairs, and feedback multicasted only to local scopes.

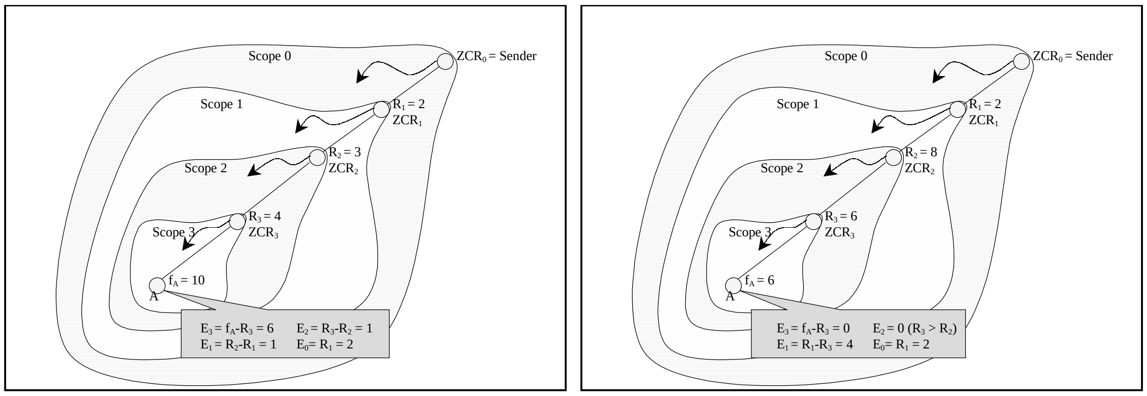
Let f_A be the number of FEC repairs needed by a receiver A. Suppose that the scope of the receiver is nested by t scopes, $s_1, s_2, \ldots, s_t$ (i.e., the nesting level is t), s_1 is the scope of receiver A, and s_j, $2 \leq j \leq n$, contains s_{j-1}. In nested scopes, receivers have choice to join repair groups of their nesting scopes. The receiver always listens to all the base groups of its nesting scopes. The RGI of a scope also is transmitted to its base group as well as included in every repair packet that the ZCR of that scope multicasts. Let $\phi_{x,y}, y \geq 1$ be the number of FEC repair packets to be transmitted to multicast group $g_{x,y}$.

It is nontrivial to determine the multicast groups that a receiver must belong, for obtaining the repair packets. A receiver A in scope s_1 first takes into account the number of repair packets that the ZCRs of its nesting scopes expect to receive from their own ancestor ZCRs. This is because a receiver may have lower loss rates than its ZCR; this happens sometimes when the ZCR is not at the routing path from the sender to that receiver. Since a ZCR can inject repair packets for a block only after it successfully receives that block, when the ZCR has higher loss rates than a receiver. relying on repair packet from that ZCR would prolong recovery delay for that receiver. In what follows, we sketch a more sophisticated protocol for determining the multicast groups to which a receiver belongs.

Let R_{s_j}, $1 \leq jlen$, be the number of FEC repair packets that the ZCR_{s_j} expects to receive from its own nesting ancestor scopes. R_{s_j} is part of the session information that ZCR_{s_j} multicasts. Based on the R_{s_j}'s, receiver A determines the expected number of repair packets E_{s_j} to receive from each ZCR_{s_j}. E_{s_j} is determined as follows. Let s_x be the smallest scope such that $f_A - R_{s_x}$ is positive. We set $E_{s_i} = 0$ for $1 \leq i \leq x$ and $E_{s_x} = f_A - R_{s_x}$. For each scope s_r, $x+1 \leq r \leq n-1$, find the smallest nesting scope of s_r, s_l, ($l > r$) such that $R_{s_l} - R_{s_r}$ is positive. Then we set $E_{s_i} = 0$ for $r \leq i \leq l$ and $E_{s_{l+1}} = R_{s_l} - R_{s_r}$. We repeat the above process until we reach the largest scope ($s_n = 0$). Then we set $E_0 = R_1$ if R_1 is positive, and $E_0 = 0$ otherwise. For each scope s_j for which $E_{s_j} > 0$, receiver A finds the minimum r such that joining $g_{s_j,1}, \ldots, g_{s_j,r}$ gives at least E_{s_j} repair packets, and it becomes a member of those groups.

If a scope i is in the static mode, its ZCR distributes B/K repairs to each $g_{i,j}$, $1 \leq j \leq K$. If the scope is in the dynamic mode, it uses the following algorithm. The ZCR first determines the number of FEC repairs it would get from its nesting scopes. Let ρ_i be that number, and f_{max} be the maximum number of repairs required by receivers in its local scope. The ZCR multicasts at most $f_{max} - \rho_i$ repair packets to its repair multicast groups. The algorithm in Section2.2.2 determines the distribution of these repair packets to its repair groups.

Examples. Figure 18 shows two examples for the calculation of E_i, the expected number of proactive repair packets to be received from scope i. Figure 18(a) shows a scenario where receivers have higher loss rates than their ZCRs whereas Figure 18(b) shows a different scenario where receivers may have lower loss rates than their ZCRs. For simplicity, we assume that ZCRs are in the static mode and $b/K = 1$ which means that only one repair packet is transmitted to each multicast group. In Figure 18(a), $R_1 = 2, R_2 = 3, R_3 = 4$, and $f_A = 10$. According to the equations discussed above, receiver A sets $E_3 = 10 - 4 = 6$, $E_2 = 4 - 3$, $E_1 = 3 - 2$, and $E_0 = 2$. Thus, the sum of E_i's for receiver A is 10. In Figure 18(b), $R_1 = 2, R_2 = 8, R_3 = 6$ and $f_A = 6$. Here R_3 is less than R_2. Also $E_3 = 6 - 6 = 0$. Since $R_2 > R_3$, $E_2 = 0$, and $E_1 = R_1 - R_3 = 4$ because scope 1 is the smallest scope that has the expected number of packet from its ZCR (R_1) to be smaller than R_3. $E_0 = 2$. Thus, the sum of E_i's for receiver A is 10. Note



(a) (b)

Figure 18: Examples on computing E_i.

that receiver A does not receive any packet from ZCR_2 and ZCR_3 because they have the same or worse loss rates. Thus, the recovery of blocks received by receiver A does not depend on those of ZCR_2 and ZCR_3.

A.5 Retransmission in scopes

When a receiver cannot recover a data block from receiving FEC repairs, it asks for additional transmission of FEC repair packets from other receivers within its *local* scope in the same way it was described in Section 2.2.4. But determining the time to send a request in a nested scope is not trivial. In the single scoped protocol, it can be determined by detecting gaps in the sequence numbers of packets because repairs are transmitted in a sequential order. For instance, the sender always multicasts in each repair group the FEC repair packets of a block before those of the next block. When listening to a repair group, it never receives the repair packets of earlier blocks if no reordering of packets happens in the network. Thus when a receiver gets a repair for a block before receiving a repair for its previous block, it can safely assume that it lost that repair. Unfortunately, this does not work in nested scopes because ZCRs can only transmit repair packets of a block only after they recover that block, and may recover blocks in different orders than they are transmitted. Thus, even if it receives a repair for a block before a repair for the previous block, it cannot decide whether that packet is lost or has not been sent.

To handle this situation, for each repair group that it joins, a receiver a estimates the expected time within which it would receive repair packets of a block. If it is supposed to receive x repair packets from a repair group y, it computes the expected arrival time $t_{a,y}$ of all of x repair packets relative to the receiving time of the first packet in that block. $t_{a,y}$ can be computed by adding $x \times \Delta_f$ and t_b, the expected time that its ZCR b recovers all the packets in the block relative to the receiving time of the first packet. The expected time that receiver b recover all of its packets in the block can be computed by taking the maximum of the expected arrival time for every repair group it listens to. ZCR b includes t_b in its RGI so that receiver a can compute $t_{a,y}$.

When a receiver a receives a data packet i of a block where i is the sequence number of the packet within that block, and i is the first packet it receives in that block, it sets its timer, called *recovery timer*, for a repair group y to $(b - i) \times packet\_interval + t_{a,y} + 2 \times sd$, where sd is the standard deviation in delays from ZCR b to a. If it does not receive all x packets from that ZCR, it considers that those packets it have not received are lost. When a receiver detects a packet loss from which a data block cannot be recovered even if it would receive all FEC packets it is supposed to receive from all of its currently joined repair groups, it sets its request timer to be a random number within $2^i[C_1 d_{Z,a}, (C_1 + C_2)d_{Z,a}]$ where $d_{Z,a}$ is the delay from the ZCR Z of a's local scope to itself. While the request timer is pending, if it receives a repair packet that it considered lost (for instance, due to wrong estimation of expected recovery), it cancels the request timer. Even if receiving that packet, and any repair packets following it in the future does not recover their pertaining block, it sets the request timer again.

When its request timer expires, receivers a multicasts a request to its local group if a is not ZCR. If a is a ZCR, then it multicasts it to its parent's local group. When a receiver c gets a request, it sets its timer to be a random number within $[D_1 d_{a,c}, (D_1 + D_2)d_{a,c}]$.

A.5.1 Nesting vs. Non-nesting scopes

Nested scopes allow receivers to receive repairs from outer scopes, and therefore, there is more chance of recovery than non-nested scopes. A receiver does not have to rely only on the ZCR of its scope to receive FEC repairs. Since it is possible that that ZCR may not be in the routing path and experiences more losses than its child receivers, a receiver can rely on other ZCRs of its nesting scopes for FEC repairs. The protocol described in Section A.4 handles this automatically. Note that a ZCR injects repairs only when repairs required to recover its losses are not enough to recover its child receivers. Thus, if a ZCR undergoes more losses than its child receiver, the child receiver does not receive repairs from that ZCR.

These advantages, however, come at the expense of protocol complexity because receivers depend on the session information about its outer scopes in deciding the number of repairs to receive from its nesting scopes. Even if session information about a scope is included in the every packet transmitted, that information can also be lost. A problem occurs when a ZCR misses the session information from its outer scope while its child receivers receive that information. The ZCR may accidentally send less or more repair packets than necessary. Although this problem will disappear as the network stabilizes, this transient behavior may cause receivers temporarily subject to too much or little traffic than necessary. This dependency on information adds to the complexity of the protocol, making it harder to maintain or debug.

The protocol for non-nested scopes is relatively simple, and is not much different from the single scoped protocol. Since each receiver only needs to receive session information from the sender and its own ZCR, it is less likely that receivers lose both information, and that receivers accidentally receive too much more or less repairs than necessary. Furthermore, when ZCRs are placed at the routing path, then nested scopes do not provide any advantage over non-nested scopes. This is because all the descendents of the ZCRs always suffer at least the same losses as the ZCRs. However, when ZCRs are not placed at the routing paths, non-nested scopes can have larger recovery delays than nested scopes since a receiver gets all the FEC repairs from its own ZCR.

The decision whether a scope is nested within another scope or be a child non-nested scope of another is made when that scope is set up. It should be highly dependent on how the ZCR of that scope is elected. If it is elected manually by designating one receiver at the routing path as a cache, a non-nested scope is more appropriate than a nested one. If the ZCR is elected through a dynamic ZCR challenge, nested scopes may prove more effective as the closest receiver to the sender is not necessarily at the routing path.

B Psuedocode for the network delay estimation phase

```
Receiver $i$'s protocol
At  every period $T_{probe}$:
    probe_sequence ++;
    $t_s \leftarrow$ the time stamp in the last message from the sender;
    delay $\leftarrow$ the delay from the reception of the last message from the sender
    multicast a probe $P_i$(probe_sequence,$t_s$,delay)
On  receiving a probe $P_i$(s,$t_s$,delay):     //received a probe message with sequence number $s$.
    Probe[$i$].time $\leftarrow$ gettimeofday();
    Probe[$i$].seq $\leftarrow$ s;
    if (Ack[$i$].seq = s)     // if received the corresponding ack from the sender
      $D[i,j] = (D[i,S] + D[S,j] -$ Ack[$i$].delay) - (Probe[$i$].time - Ack[$i$].time);
On  receiving an ack ACK(i,s,delay,$d_{i,S}$) from the sender $S$,
    Ack[$i$].time $\leftarrow$ gettimeofday();
    Ack[$i$].seq $\leftarrow$ s;
    Ack[$i$].delay $\leftarrow$ delay;
    $D[i,S] \leftarrow d_{i,S}$;
    if (Probe[$i$].seq = s)     //if received the corresponding probe from $i$
      $D[i,j] \leftarrow (D[i,S] + D[S,j] -$ Ack[$i$].delay) - (Probe[$i$].time - Ack[$i$].time);

Sender's protocol
On  receiving a probe $P_i(s, t_s, delay)$:
    $t_{now} \leftarrow$ gettimeofday();
    $D[i,S] \leftarrow t_{now} - t_s -$ delay;
    delay $\leftarrow$ the delay from the reception of the probe message;
    multicast ACK(i,s,delay,$D[i,S]$);
```