Real-Time Reliable Multicast Using Proactive Forward Error Correction*

Dan Rubenstein, Jim Kurose, and Don Towsley
Computer Science Department
University of Massachusetts at Amherst
{drubenst, kurose, towsley}@cs.umass.edu

Abstract

Real-Time reliable multicast over a best-effort service network remains a challenging research problem. Most protocols for reliable multicast use repair techniques that result in significant and variable delay, which can lead to missed deadlines in real-time scenarios. In this paper we present a repair technique that combines forward error correction (FEC) with automatic repeat request (ARQ). The novel aspect of the technique is its ability to reduce delay in reliable multicast delivery by sending repairs proactively (i.e., before they are required). The technique requires minimal state at senders and receivers, and no additional active router functionality beyond what is required by the current multicast service model. Furthermore, the technique uses only end-to-end mechanisms, where all data and repairs are transmitted by the data-originating source, leaving receivers free from any burden of sending repairs. We simulate a simple round-based version of a protocol embodying this technique to show its effectiveness in preventing repair request implosion, reducing the expected time of reliable delivery of data, and keeping bandwidth usage for repairs low. We show how a protocol using the technique can be adapted to provide delivery that is reliable before a real-time deadline with probabilities extremely close to one. Finally, we develop several variations of the protocol that use the technique in various fashions for high rate data streaming applications, and present results from additional simulations that examine performance in a variety of Internet-like heterogeneous networks.

1 Introduction

Multicast has become an important component of the Internet within the past decade. Deering's work [1] describes a framework for distributing data to multiple receivers via *multicast groups*. When multicast groups grow large, simple reliable multicast protocols suffer from a condition known as *feedback implosion*: an overload of network resources due to many receivers trying to send repair requests (henceforth referred to as NAKs) for a single packet. A number of solutions exist to avoid this implosion effect, using techniques such as randomized timers, local recovery (receivers sending repair packets), and hierarchical recovery. While such techniques are effective in providing reliability without implosion, they can result in significant and unpredictable delays, making them unsuitable for applications that have stringent real-time constraints.

In this paper, we present a technique that uses a novel combination of forward error correction (FEC) and automatic repeat request (ARQ) to reliably deliver data, with an emphasis on reducing delay and meeting real-time constraints without using randomized timers, local recovery, or hierarchical recovery. We call this technique *proactive FEC*, because it forwards error correcting packets into the network prior to their necessity.[1] It is this idea of having the data source forward repairs before they are required that differentiates our work from previous reliable multicast protocols that make use of FEC.

Our results through simulation indicate that this technique 1) offers a significant decrease in the expected time to reliably receive data, 2) can be used to meet hard real-time deadlines of reliable delivery of data with probabilities extremely close to one, 3) reduces feedback implosion, 4) competes well with other protocols in terms of required network bandwidth, 5) can reduce bandwidth requirements for meeting real-time deadlines compared to non-proactive approaches, and 6) scales to groups containing thousands of members configured in various topologies with various spatial and temporal loss characteristics. We also show that the technique functions well for high-rate data transfers in

*This material was supported in part by the National Science Foundation under Grant No. CDA-9502639, NCR-9508274, and NCR-9527163. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[1] The term is actually somewhat redundant, since the *F* in FEC implies the sending of information before its necessity. However, FEC in a multicast context commonly refers to sending encoded repair packets, as opposed to direct retransmission of the data, in response to NAKs.

networks containing large multicast groups, where receivers are at varying distances from the sender, with varying end-to-end loss rates and latencies.

The remainder of this paper is structured as follows. Section 2 gives an overview of related work, followed by a brief description in Section 3 of the coding method used to provide forward error correction, and its application within the domain of reliable multicast. The technique used to reduce delay and provide hard real-time guarantees is described and evaluated in Section 4, followed by a description and examination of low latency, reliable protocols in section 5. Real-time, probabilistically reliable (i.e., resilient) protocols are presented later on in this section as well. Section 6 further examines performance of our protocol as we vary a wide variety of network features. We conclude the paper in Section 7.

2 Related Work

In this section we discuss previous work that addresses various issues that arise when attempting to reliably multicast data in the Internet. To reduce NAK and repair transmissions, reliable multicast protocols have incorporated random delays [7]. However, this comes at the cost of increased latency. A variety of approaches have been proposed that limit the bandwidth used by NAKs and repairs, including scoping [7], and communicating via unicast to nearby receivers [10, 11] or designated repair servers [8]. Using such approaches, repair latency and bandwidth depend heavily on the location of both the repair entity and the point of loss within the network: the benefit is reduced as their distance to the receiver increases. Another approach restricts the number of entities that provide immediate feedback. This is used in [15], with the data source periodically choosing a small set of representative receivers that have priority in sending feedback. Here, the protocol's effectiveness depends on the source's ability to select a good set of receivers with its limited knowledge of the group topology and network loss characteristics.

There has also been recent interest in providing *resilient multicast* service for real-time data, where retransmissions occur only if data can be delivered before the real-time deadline. Data is not reliably delivered, but a higher good-put can be achieved than without any retransmission. Two protocols that that are designed to provide resilient multicast are STORM [10] and LVMR [12]. Both approaches form virtual trees with the source as the root and receivers as internal and leaf nodes. Recovery is implemented by sending all repair requests and retransmissions via unicast along this tree. Repairs can be performed with low latency provided that they do not need to traverse numerous links within the receiver-based tree. However, substantial delays can occur when losses occur close to the source. In such cases, the transmission path to a receiver can be significantly longer than the multicast route direct from the source. This is because repairs occur as a series of unicast transmissions between receivers.

Additional functionality within routers can also improve real-time reliable multicast performance. Several mechanisms have been suggested as a means of improving reliability [16, 9, 18]. However, this is at the cost of additional router state and / or processing.

Forward error correction (FEC) [3] is a technique that reduces the bandwidth overhead of repairing errors or losses in bit streams. It has been shown in [6] that a hybrid approach combining FEC with ARQ can significantly reduce bandwidth requirements of a large reliable multicast session. Hybrid approaches that combine FEC and ARQ have been proposed and classified for repairing loss and noise at the bit-level [23]. The Type I Hybrid approach involves sending repair bits within a packet that can correct bit errors or erasures, and retransmitting packets if the number of repair bits is insufficient to reliably receive the packet. Type II Hybrid approaches place the repair bits in packets that are distinct from the packets that contain the data. Our approach is considered to be a Type II Hybrid approach. However, our forwarding of repair packets before their necessity incorporates certain favorable features of a Type I approach as well.

[17] presents a preliminary analysis that compares the benefits of combining local recovery with an FEC/ARQ hybrid technique, and concludes that for many multicast scenarios, such a combination offers little improvement over an FEC/ARQ hybrid technique without local recovery. [13] compares real-time performance of reliable multicast techniques that use FEC to those using ARQ techniques, but does not consider hybrid approaches. An interesting approach using FEC is presented in [14], where data is delivered reliably without requiring ARQ through multicast group joins and leaves. A recent implementation utilizes ARQ as a last resort to obtain any data that could not be obtained via the joining of a particular multicast group. Present join-leave latencies for multicast groups make the approach too bandwidth inefficient to support real-time applications. Other works are showing the promise of the proactive FEC technique. One such work is [21], in which a round-based proactive FEC approach is combined with a receiver-based hierarchical recovery scheme.

3 Overview of Forward Error Correction (FEC)

Error correcting codes were initially applied in domains where bits could be erroneous or missing, but have more recently been applied to repairing packet losses at the net-

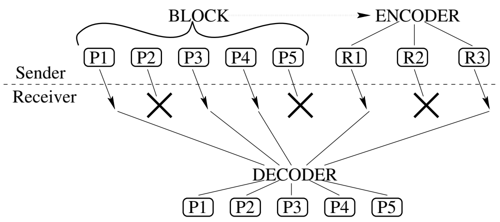

Figure 1: A sample decoding using a block built from 5 packets.

work layer. The following description is sufficient for understanding how systems, equipped with Reed-Solomon encoders and decoders, can make use of repair packets to recover from loss. The sender forms **blocks**, where each block consists of a subset of the data packets it wishes to deliver reliably. The number of data packets that are used to form a block is commonly referred to as the *block size*, k. The sender inputs the k packets into its encoder which then generates *repair packets* for that block. A receiver uses its decoder to recover the k data packets from any combination of k distinct packets that use data packets from the block, and/or repairs generated for the block. An example is shown in Figure 1, where a sender groups 5 data packets into a block, encodes 3 repair packets from this block, and transmits all 8 packets to the receiver. As soon as the receiver receives any 5 distinct packets related to the block (in the example, 3 data and 2 repair), it activates the decoder and recovers the lost data packets.

Detailed discussions of Reed-Solomon coding techniques and of packet-level FEC techniques can be found in [4] and [6] respectively; implementation issues are considered in [5, 6]. For our purposes here, we simply note that FEC techniques exist that can be used to generate as many repair packets as needed, and that this can be done at data rates on the order of 8 Mbytes/sec on commodity PC's.

4 A Proactive FEC+ARQ Technique

In this section, we present a proactive technique that delivers data reliably to a set of receivers through a combination of ARQ and FEC, and examine the impact that proactivity has on the performance of reliable data transfer to large multicast groups. The technique is *proactive* in that it allows the sender to send repairs for packet losses prior to receiving any indication that the repairs are necessary. This allows receivers to tolerate a certain amount of packet loss and still receive all data reliably without requiring retransmissions. As a result, the average number of receivers that require retransmissions decreases, reducing implosion. We will also observe that proactive FEC can reduce the number of repair packets that are needed.

We first examine the proactive FEC technique using an idealized round-based protocol. During each round, the sender sends data and repairs and awaits feedback from all receivers needing additional repairs. The sender waits for a response from the slowest responding receiver before proceeding to the next round. As a result, the protocol introduces additional latencies in attempting to meet individual receivers' real-time deadlines. In Section 5, we study protocols that do not require such synchronization.

As in most protocols that use FEC, the sender packetizes its data for delivery, and builds blocks of k packets. For simplicity of presentation, we assume a fixed block size for the entire data stream, and that blocks are built from consecutive packets within the data stream.

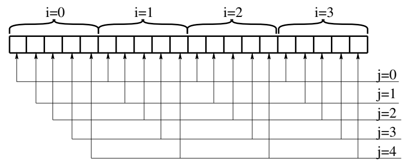

Figure 2: Partitioning a data set into 4 blocks of size 5 so that repairs can be generated using FEC.

Each packet transmitted by the sender contains a pair of identifiers (i, j), where i identifies the block, and j the position of the packet within the block. For the purpose of this paper, we assume that i and j are non-negative integers. The data packets within a block i are assigned values of $(i, 0)$ though $(i, k-1)$ corresponding to their ordering in the data stream. As repairs are required, the sender creates and sends them. For a block i, repair packets are assigned sequence numbers (i, j) with $j \geq k$. These sequence numbers are used by the decoder at the receivers' end to determine the operations that must be used to retrieve lost data. Using a block identifier within a packet permits the interleaved transmission of the various blocks, so that a transmission of a new block can commence before other blocks have completed their reliable delivery. Figure 2 gives an example of a data stream partitioned into four blocks each of size 5.

Associated with the protocol is a proactivity factor, ρ, which is a rational number larger than or equal to one. For a block of size k, the sender initially transmits $\lfloor \rho k \rceil$ packets,[2] consisting of the k data packets plus an additional $\lfloor (\rho - 1)k \rceil$ repair packets. No other repair packets are sent for the block unless receivers specifically request them.[3] Such requests are unicast by a receiver only when it fails to obtain the entire block of data, either through the reception of the original block, or through application of its decoder. Otherwise, a receiver need not send any feedback to the sender. In order to satisfy all receivers' requests, the sender responds to the NAKs by multicasting a number of repair packets that

[2] The notation $\lfloor x \rceil$ means round x to the nearest integer.

[3] Encoders such as the one presented in [5] are capable of generating a single repair packet at a time (i.e., it is not necessary to know in advance the number of repairs needed.)

satisfies the maximum from all of the receivers' requests for that round. This process continues until each receiver either obtains a sufficient number of packets to perform decoding or passes some deadline after which there is no point in retrieving the block.

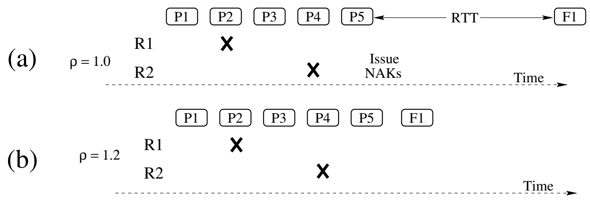

Figure 3: Illustration of possible savings in time and reduction of feedback through proactivity

Figure 3 demonstrates the potential benefits of setting the proactivity factor to a value larger than one. In Figure 3(a), two receivers, R1 and R2, are sent a block of five data packets, and each receiver loses one of these data packets. If $\rho = 1.0$, then no repair packets are sent with the initial transmission, and the receivers must NAK and wait for the sender to retransmit the data. Alternatively, in Figure 3(b), if $\rho = 1.2$, then a repair packet is transmitted with the data, and the receivers do not need to request repairs nor wait a full round trip time for the sender to transmit the repair. We point out that the round-based approach creates no interdependencies among the various blocks being transmitted, and so the sender can deliver multiple blocks at a given time.

Sender and receiver state diagrams depicting the idealized round-based protocol are given in Figures 4 and 5, respectively.

By sending repairs proactively, more receivers will obtain at least k packets in the initial round, resulting in the return of fewer repair requests to the sender. Thus, the sender can effectively control NAK implosion by sending an appropriate number of repairs proactively in the initial round. The sender benefits little from adding proactivity to subsequent rounds because the number of repair requests in each subsequent round is most often significantly smaller than the number of repair requests in the initial round. On the other hand, receivers continue to benefit when proactivity is added in subsequent rounds, since it can often reduce the number of rounds needed to reliably receive a block.

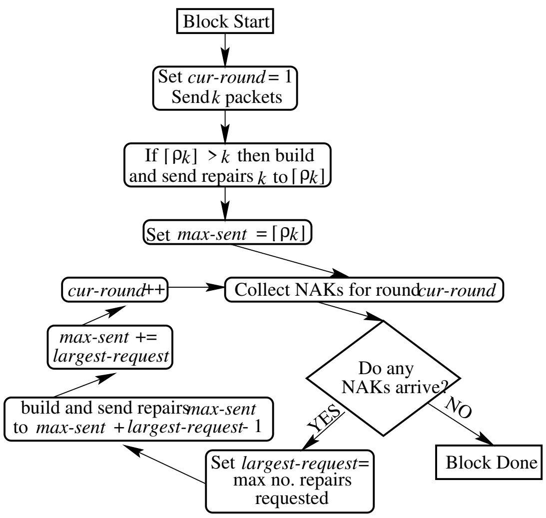

Figure 4: The round-based sender algorithm

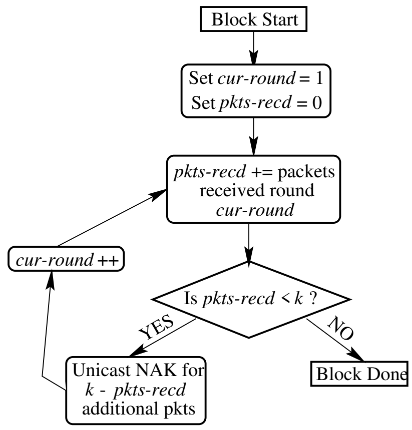

Figure 5: The round-based receiver algorithm

Receivers can control the amount of proactivity used in subsequent rounds by requesting more repair packets than are needed. This allows each receiver to request a level of proactivity that best satisfies its own requirements. The amount of needed proactivity can be determined in several ways. For instance, a receiver could apply the sender's proactivity factor to its own request, and if it needs n packets, request $\lfloor \rho n \rceil$ packets. Alternatively, it could simply request an additional packet so that it can tolerate the loss of one repair packet, or if it knows the loss rate from the sender to itself, it can request a number of repairs such that it can obtain a sufficient number of repairs with some fixed probability. In Section 4.1, we show how receiver-initiated proactivity can be used to allow each receiver to meet its own hard deadline.

4.1 Meeting hard deadlines

If a receiver can estimate the loss rate from the sender to itself, it can request more repairs for a block than the minimum number needed in order to perform decoding. Doing so allows the receiver to meet a deadline with probabilities that are extremely close to one. To do so, the receiver must be able to calculate the probability of receiving at least n packets out of a group of m packets. We represent this value by $\mathcal{D}(m, n)$. If the loss is modeled as a temporally independent loss process with packet loss probability p, then we have:

$$\mathcal{D}(m,n) = \sum_{i=n}^{m} \binom{m}{i} (1-p)^i p^{(m-i)}.$$

Since $\lim_{m\to\infty} \mathcal{D}(m,n) = 1$ (a simple proof is presented in [25]), a receiver that needs n packets can guarantee that the packets are received with a probability larger than any $\mathcal{P} < 1$ by determining an appropriate m that satisfies $\mathcal{D}(m,n) \geq \mathcal{P}$. A search for an appropriate m can be performed quickly, since $\mathcal{D}(m,n)$ satisfies the recurrence relation:

$$\mathcal{D}(m,n) \quad = \quad \mathcal{D}(m-1,n) + \binom{m-1}{n-1}(1-p)^n p^{m-n}.$$

with the initial condition $\mathcal{D}(n,n) = (1-p)^n$. The computation of $\mathcal{D}(m,n)$ for a two-state loss model is presented in the [25].

A receiver can use a conservative estimate of the length of a round to determine the last round by which it must receive repairs in order to meet a deadline. If the receiver still needs n more repairs upon entering this last round, it makes a request for m repairs, choosing m large enough so that sufficient repairs arrive with a high enough probability.

We present two types of guarantees that receivers can make to meet a hard deadline:

Last round guarantee. Here, the receiver guarantees that if a last round is necessary, then enough repairs will be delivered in that round to insure that the conditional probability of being able to decode all packets in the block, given the number of packets still needed before starting the last round, is greater than $\mathcal{P}$. To make this guarantee, the receiver simply chooses m such that $\mathcal{D}(m,n) \geq \mathcal{P}$, where it needs n packets going into the last round.

Block good-put guarantee. Alternatively, the receiver may wish to achieve some overall block good-put rate, such that the probability that a block is received on or before the last round is $\mathcal{P}$. We prove in the Appendix that if a receiver needs n packets going into the last round, it is sufficient to choose m such that $\mathcal{D}(m,n) \geq (\mathcal{P} - \mathcal{D}(l,k))/(1 - \mathcal{D}(l,k))$, where l is the number of packets sent over all previous rounds, and k is the block size. Simple algebra reveals this to often be smaller and never larger than what is required to meet the last round guarantee. If $\mathcal{D}(l,k) \geq \mathcal{P}$, then no attempt is made on the last round to retrieve the block. We emphasize that these results apply for any model of loss between source and receiver. The receiver must simply compute $\mathcal{D}()$ using the appropriate loss model.

By attempting to meet a hard deadline, receivers can request more repairs than it would otherwise, thereby increasing the number of packets that the sender transmits. When a hard deadline condition exists, added proactivity can reduce the expected delay of reliable receipt by receivers as well as the expected number of packets that are transmitted by the sender. This is because additional proactivity makes it more likely that receivers will have already obtained the entire block before entering the last round, thus obviating the need to request a large number of repair packets in order to meet the guarantee probability, $\mathcal{P}$.

4.2 Examination: round-based protocol

The focus of this paper is to demonstrate the effectiveness and simplicity of using the proactive FEC technique. Performance is measured against a traditional end-to-end approach where repairs are data packets which are multicast to the entire multicast group in response to a NAK. In effect, this traditional approach does not use FEC, and repairs are not sent proactively. Furthermore, the traditional approach does not scope repairs, so all receivers in the multicast group will receive each repair transmission (regardless of where the repair transmission originated from). We believe that our proactive FEC technique competes well with many protocols that use scoping approaches such as local recovery, additional router support, or multiple multicast groups. However, we avoid direct comparison for two reasons. First, benefits due to scoping are difficult to measure: they depend heavily on network and multicast group topologies, as well as on how the various amounts of increasing and decreasing traffic on various links affects protocol and network performance. Second, we believe it is possible to combine the proactive FEC technique with a scoped approach to further improve reliable multicast protocol performance. Since scoping reduces the effectiveness of FEC techniques, determining the right way to combine FEC with scoping is still an open problem.

We evaluate the performance of the round-based protocol through simulation for networks containing up to 10,000 receivers. Simulation allows us to examine a much richer set of network scenarios than is possible via a mathematical analysis, and makes it easier to observe effects of large multicast sessions than it would be through experimentation. We have performed an analysis of a star topology network; it appears in [25]. Results presented here are simulated over a model which we now describe. Other topologies, receiver placements, loss models, and block sizes are considered in depth in [25]. A brief summary of their impact is given at the end of this section.

The model used in the simulations presented here is a randomly generated tree network, where nodes on the interior of the tree represent routers with downstream fanouts of one, two and three, with loss probabilities at each outgoing link uniformly distributed between 0.00075 and 0.00125. Nodes on the leaves of the tree that connect to the sender or receivers had downstream fan-outs ranging from 1 to 5 with loss probabilities uniformly distributed between

Amortized cost to deliver a packet reliably					
	Multicast Group Size				
Blocksize	1	10	100	1000	10000
1	1.09	1.62	2.49	3.35	4.21
5	1.14	1.36	1.60	1.82	2.04
10	1.16	1.29	1.43	1.56	1.70
15	1.11	1.25	1.37	1.46	1.56
20	1.12	1.23	1.34	1.41	1.48

Table 1: Comparison of the amortized, expected forward bandwidth used to transmit a packet reliably for various block sizes and multicast group sizes over the sample tree topology used in the experiments for this paper. There is no proactivity ($\rho = 1.0$). Note that a block size of 1 is identical to having no FEC, where each multicast retransmission is the sole packet within the block.

.0375 and .0625. These loss rates are similar to those observed in [19], and the fan-outs coincide roughly with what is observed within the Internet. These values also provide for realistic end to end properties within the Internet: the number of hops from sender to receiver varies from 3 to 28 with a mean hop-count of 15.75 and a mean end to end loss rate of .0896. The algorithm used to construct the tree is presented in [25].

Once a tree has been created with 10,000 leaves, where each leaf node represents a potential receiver, the number of receivers is varied by selecting a subset of leaf nodes in the tree of the desired size, and placing receivers only at these nodes. We have experimented with a set of approximately 20 randomly generated trees, and have found that the results vary little for the different topologies generated. For this reason, the results in this Section are computed using a single, randomly constructed tree.

We fix the set of active receivers that are used to examine a multicast group of a particular size. The set of receivers that are active in the smaller multicast groups are proper subsets of those receivers that are active in the larger multicast groups. We model loss at each router as a Bernoulli process, so that there is no temporal correlation between consecutive packets being forwarded. The block size is fixed at 10.

For each configuration described above, we performed several experiments on the same topology (around 20) and computed the average values of our measuring criteria to generate a distribution that was close to normal, and then used as many of these averaged values as needed to calculate 95% confidence interval widths that were within 5% of the point value.

We will evaluate performance using three general criteria:

Delay. The manner in which delay will be examined will depend on the goal of the protocol. For protocols that do not impose hard real-time deadlines, we will be interested in the expected delay of reliable delivery. For those that *do* impose hard deadlines, we will be interested in the expected percentage of blocks that can be received before the deadline expires. We elaborate further on the way we measure delay later in the paper.

Implosion factor. Here, we will be interested in the expected number of NAKs a sender receives from the receivers per block.

Forward bandwidth. This is the expected number of packets (repair and data) sent by the sender per block. Because the sender multicasts each packet it sends, and each packet traverses all links in the multicast tree (except links on the tree that lie below points of loss of a packet), the bandwidth usage in the network is roughly linear to this number of packets transmitted.

We begin by examining the effect of block size on the expected forward bandwidth used by any protocol that multicasts repairs to an entire group. We define the amortized (per packet) expected forward bandwidth to be the expected number of packet transmissions needed to deliver a block divided by the blocksize. Table 1 gives the amortized, expected forward bandwidth needed to reliably deliver a packet to all receivers as a function of receiver population and block size, where no proactivity is used ($\rho = 1.0$). The configuration used to compute the results for the table is the same as that used to compute the results in the remainder of this section. An examination of the table reveals that for large multicast groups, one obtains a significant reduction in expected forward bandwidth by increasing the block size. We note that a protocol using a block size of one is equivalent to a protocol that does not use FEC, where each repair can be used to repair a single loss. Thus, the top row represents the amount of forward bandwidth that would be used by a traditional end-to-end reliable multicast protocol.

Figures 6(a), 6(b), and 6(c) respectively show the expected number of rounds, implosion factor, and forward bandwidth per block for various multicast group-sizes as a function of the sender's proactivity factor. An increase in the proactivity factor decreases the expected number of rounds in a roughly linear manner. The implosion factor decreases exponentially as a function of the proactivity factor, so that NAK implosion can be reduced significantly with small increases in the proactivity factor. The most interesting result is the effect that the proactivity factor has on forward bandwidth. For large multicast groups, increasing the proactivity factor up to a certain value has an insignificant effect on the expected number of packets transmitted to reliably deliver data to all receivers. We note that for a block size of 10 and a receiver group-size of 10,000, if the sender's proactivity factor is 1.6 then the mean number of rounds is 2, the implosion factor is approximately 10 NAKs, and the forward bandwidth is 17 packets (10 original and 7

(a)

(b)

(c)

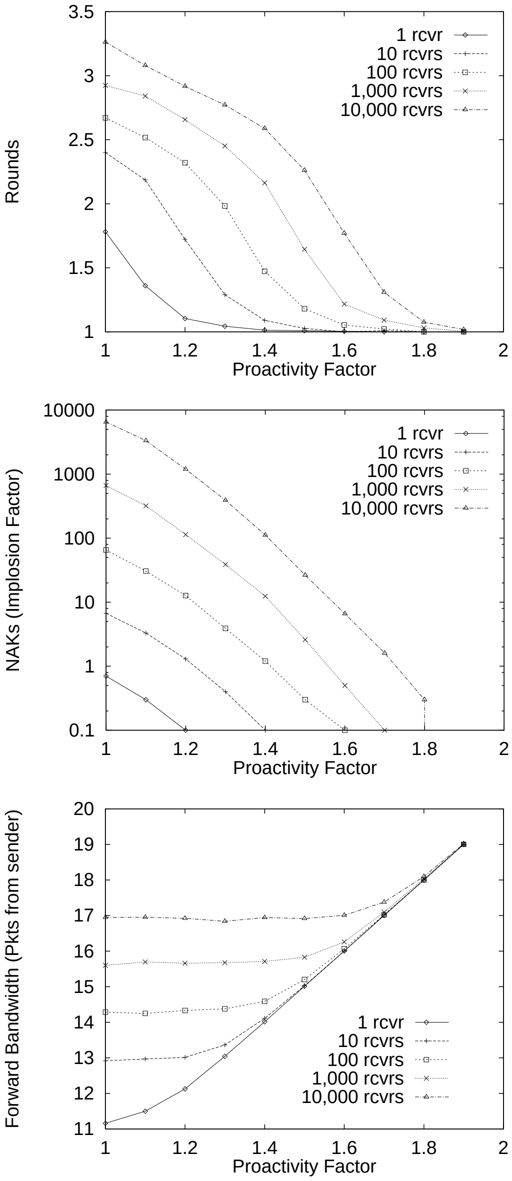

Figure 6: Results from a round-based simulation on a tree topology of receivers. Note that the y-axis of (b) is plotted on a log-scale.

repair packets). This compares favorably in every respect to an ARQ protocol without local recovery (i.e., a proactivity factor of 1.0), which would take close to 4 rounds and require the transmission of around 40 packets. In addition, the need for randomized delays to prevent NAK implosion would further increase the latencies associated with an ARQ protocol.

We can summarize these observations:

By using proactive FEC, we can reliably deliver data sooner than with traditional ARQ and with considerably less bandwidth. Furthermore, proactivity provides large decreases in repair bandwidth and delivers data reliably sooner when compared to a non-proactive FEC-ARQ hybrid approach. The proactive approach achieves these gains without using significantly more forward bandwidth than its non-proactive counterpart.

This is because with a large group, a certain number of repairs must be expected. With proactive FEC, these repairs are simply sent before it is known for certain that they are needed. Note that after some point, however, the bandwidth begins to increase along the asymptote $y = kx$, where k is the block size.

An alternate view of the data in Figure 6 is provided in Figure 7. Here, we directly plot the tradeoff between forward bandwidth and implosion factor (Figure 7(a)) and forward bandwidth and rounds (Figure 7(b)). Each curve is obtained by varying the proactivity factor. We see clearly from the flat portions of the curves that the proactivity factor can be chosen so that the implosion factor and number of rounds (delay) are low, without significantly increasing the amount of forward bandwidth required.

(a)

(b)

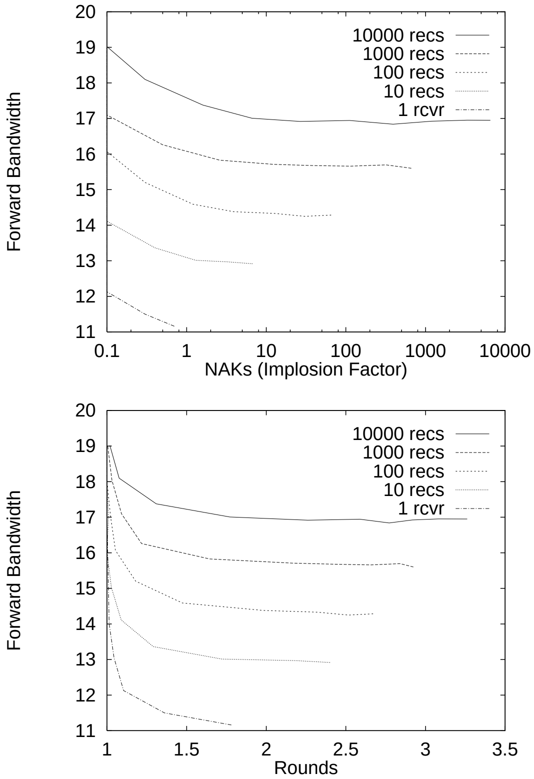

Figure 7: Tradeoff between forward bandwidth and NAK implosion and number of rounds.

We now examine the performance of the protocol in the presence of real-time constraints in the same network. Recall that in Section 4.1 we presented two ways in which receivers can use proactivity to guarantee a certain type of deadline-driven reliability. Figure 8(a) and (b) respectively show the expected forward bandwidth needed to meet a last round guarantee and a block good-put guarantee where $\mathcal{P} = 1-10^{-6}$, with all receivers requiring the data in two rounds. In other words, each receiver only participates in 2 rounds, sends at most a single NAK, and is able to meet its desired guarantee with probability greater that $1 - 10^{-6}$.

It is interesting to note that adding proactivity can actually *decrease* the expected amount of bandwidth required in order to make a last round guarantee. In contrast, a block good-put guarantee uses the least amount of bandwidth when the proactivity factor is 1.0, though the bandwidth increases slowly as a function of the proactivity factor until it starts to increase along the asymptote $y = kx$. Another interesting fact is observed by comparing the expected forward bandwidth in Figure 6(c) to Figure 8(b). We see that meeting a block good-put guarantee can actually reduce the expected forward bandwidth used in a session, even when the guarantee is a rate as high as $1 - 10^{-6}$. This is due to the fact that the receiver aborts all further attempts to obtain a particular block after some final round. The forward bandwidth savings seems to more than compensate for the additional forward bandwidth that is used on this final round to meet the guarantee.

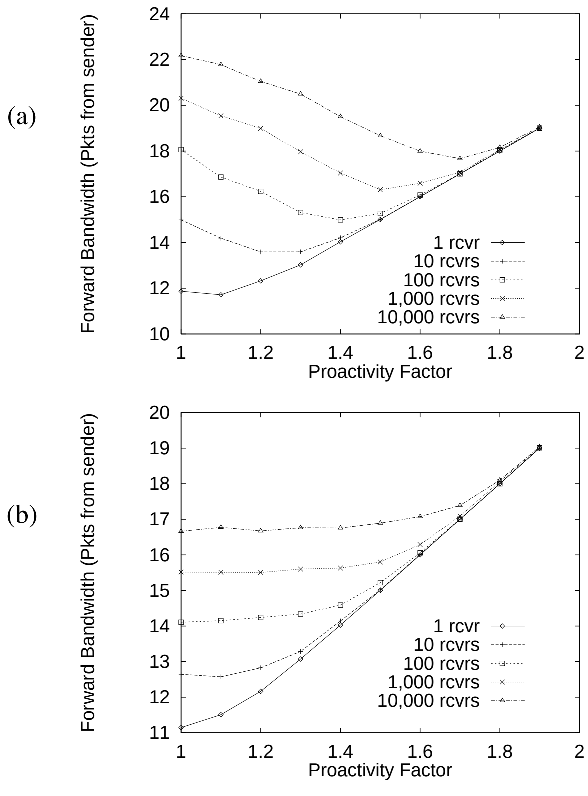

Figure 8: Bandwidth usage to provide (a) last round and (b) block good-put guarantees of $1 - 10^{-6}$ by the end of 2 rounds.

In addition to the results described above, we have examined the performance of the round-based proactive FEC protocol in a large variety of alternative network scenarios, including a variety of heterogeneous and homogeneous network topologies to examine the effects of various spatial loss correlations among the receivers. We also considered models where losses on links are similar to what is observed in [20] (i.e., various links in the backbone have high loss rates, links near the edges have low loss rates), and we examined the impact of bursty loss using 2-state loss models at routers. Finally, we also examined various ways in which receivers could add additional proactivity by sending NAKs requesting additional repairs beyond the minimum that would be necessary to complete the block. The observations made from these additional experiments vary slightly from the results presented in this section. The technique was still effective when using the loss model observed by Handley in [20] for two reasons. First, receivers that experienced high losses required high levels of proactivity. Second, the drop in loss at the link next to the source made using FEC more effective, even though there was an increase in loss (which also caused correlated loss at receivers) in the backbone. The only drawback is that the disparity in receiver loss rates caused receivers with lower loss rates to typically receive many more repairs than they actually require, in order to satisfy the high repair needs of receivers with high loss rates. In general, we found that the proactive FEC technique was less effective for very low loss rates, very bursty loss, and/or highly correlated (i.e., upstream) loss. Details are provided in [25].

5 Asynchronous Protocol

In Section 4, we demonstrated the effectiveness of the proactive technique in a heterogeneous network using a round-based protocol. Such a protocol adapts its reliable delivery rate to the requirements of the slowest receiver, reducing the protocol's effectiveness for faster receivers. We now examine a protocol that eliminates the synchronous behavior imposed by the use of rounds. This allows each receiver to communicate with the sender at a rate that is independent of the number or locations of other receivers. We assume that all receivers multicast their NAKs for the purposes of suppression. In [25], we show that this does not provide considerable benefit over unicasting NAKs.

We are able to allow receivers to operate in an asynchronous fashion by changing the format of the information that each receiver reports in its NAK. Other hybrid FEC/ARQ approaches require each NAK to request the *number of additional repairs* needed by the receiver in order to decode a particular block. We take advantage of the fact that repair packets must contain sequence numbers for decoding purposes, and perform the following subtle change to the way receivers NAK.

We use the packet sequencing format described in Section 4, where each data and repair packet contains two sequence numbers (i, j), where i indicates the block, and j the packet's its position within that block. NAKs also contain two sequence numbers (i, j) where i represents the block to which the NAK pertains. However, rather than j indicating number of additional packets desired, it indicates to the sender that it should send all data and repair packets for block i with sequence number less than or equal to j that it has not already sent. As we shall see below, performing NAKs in this fashion allows for a simple sender algorithm.

5.1 Protocol: Sender's algorithm

The sender's actions are simple and the only aspect that depends on network conditions is the value it chooses for the proactivity factor. A state diagram demonstrating the sender's algorithm for a block is given in Figure 9. The algorithm proceeds as follows: For each block i, the sender sends the data and a number of repairs which is determined by the proactivity factor. It keeps track of the sequence number (i, j) for the packet it has sent with the largest value for j. We refer to this value of (i, j) as the *largest sent sequence number for block* i, or *LSSN*. If a NAK arrives with sequence number $(i, j'), j' > j$, the sender sends all packets with sequence numbers $(i, j + 1)$ up to (i, j'), and (i, j') becomes the new value for the LSSN. It should be clear to the reader that multiple blocks can be transmitted in the same period of time, as long as the sender maintains the LSSN for each block that it is in the process of transmitting.

There are several advantages to having a sender use this simple protocol. First, the sender maintains one item of state per block: the LSSN. This makes it easy for the protocol to scale as the multicast group size grows, since the sender's state is constant with respect to the multicast group size. Second, the sender does not require any knowledge of the group topology. This is important because topology information is difficult to obtain and can vary during a session. Third, it operates in an event-driven manner, so that it doesn't need to maintain timers, except perhaps a single timer that expires when a block becomes stale and no longer requires buffering. Fourth, the sender always reacts to the repair request in a manner that satisfies the request. Alternatively, if each NAK uses the approach where it only specifies a number of repairs that are needed by the receiver, an additional burden is placed on the sender: it must determine for each arriving NAK whether or not it has already satisfied or partially satisfied that NAK's request. The answer depends on loss rates and propagation delays to the receiver that transmitted the NAK.

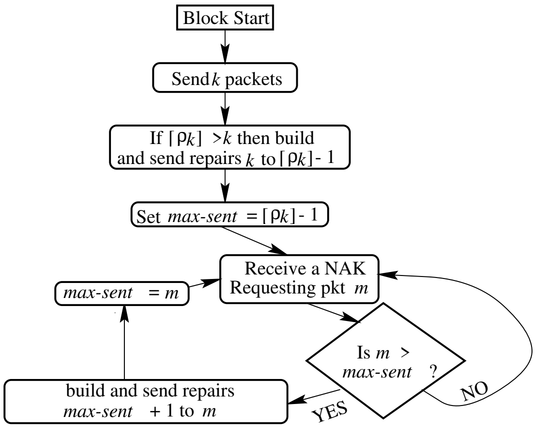

Figure 9: The sender protocol

5.2 Protocol: Receiver's algorithm

The state diagram that describes the receiver's behavior while receiving a block is shown in Figure 10. Each receiver reacts to three possible events: the arrival of a data or repair packet arrival, a NAK from another receiver, or a timeout. The way in which the algorithm reacts to these events is determined by two functions: the $NAKSeqno()$ function is used to calculate the sequence number that should appear in a NAK that is about to be sent. The $ComputeBlock$-$NextTimeout()$ function calculates the point in time when the next timeout for a particular block should occur. After a timeout, and possibly after a data or repair packet arrival, a receiver sends a NAK requesting additional repairs, using the $NAKSeqno()$ function. After processing an event, the receiver recalculates the time at which it should timeout using the $ComputeBlockNextTimeout()$ function. It then blocks until it receives another packet or another timeout occurs, at which time it repeats the process. As with the sender, multiple blocks can be transmitted at the same time, as long as the receiver maintains an independent set of states for each block that is in transmission.

The receiver algorithm is more complicated than that of the sender. However, the algorithm need only maintain a small amount of per-block state information: the time of arrival of the most recently arrived packet from the sender, the maximum sequence number over all repairs received in the block, and the time of arrival and sequence number of the NAK received with the maximum sequence number. Additionally, the receiver must maintain an upper bound on the round trip time to the sender. To meet hard guarantees, additional state is required: an upper bound on the loss rate from the sender. It should be clear that, similar to the sender's algorithm, the receiver state neither depends on the group size nor on the topology of the other receivers in the network.

Because NAKs contain intra-block sequence numbers instead of the number of repairs requested, the receiver must calculate the appropriate sequence number in order to have the sender send the appropriate number of repairs. This sequence number can be computed at the receiver by determining the desired number of repairs the sender must transmit to allow for decoding the block, and adding this value to the receiver's estimate of the sender's current LSSN for the block.

Due to loss and propagation delays, a receiver does not always accurately determine the sender's current LSSN. An overestimate or underestimate of its value respectively causes an overestimate or underestimate in the desired sequence number in the NAK. An overestimate can occur due to a previous NAK that was processed or sent by a receiver, but was lost by the sender. It can also occur if repairs incur larger than expected delays so that the receiver prematurely considers the repair lost. An underestimate can occur if the receiver loses a NAK from another receiver that reaches the sender. We note that the receiver can adjust its estimate with each packet arrival, so that the estimates are not likely to deviate dramatically from the true sender value. Also, there are several techniques that can be employed to reduce the damage that can occur due to inaccurate estimates. See [25] for details.

To prevent unnecessary NAK transmissions, after receiving a NAK, a receiver waits a conservative amount of time before reacting to a loss. When at some time t, a receiver R receives a NAK from another receiver, R', it must then give the sender an appropriate amount of time to respond to the NAK. By waiting until time $t + r$, where r is the round trip time from R to the sender, R gives the sender enough time to provide repairs based on the NAK from R'. The approach is conservative, since often the sender's repairs will arrive sooner than $t + r$.

We have experimented with two versions of the receiver algorithm that differ only in a variation of the $Compute\text{-}BlockNextTimeout()$ function. The first these, referred to as the *patient receiver protocol*, only issues a NAK for a block when it determines (with the information available to it) that all repairs issued thus-far have already been received or lost, and no NAKs are en route to the sender that will trigger additional transmissions. The second variation, referred to as the *aggressive receiver*, sends a NAK whenever it determines that the sender's current LSSN is not large enough to send enough repair packets to permit decoding for the block. Figure 11 illustrates the difference in the reaction rates of the patient and aggressive receivers.

5.3 Performance Evaluation

We now report results from simulation of the asynchronous protocol for multicast groups containing 500 receivers. The simulation was written using the TeD network simulator [22]. The group size is restricted to 500 due to the memory

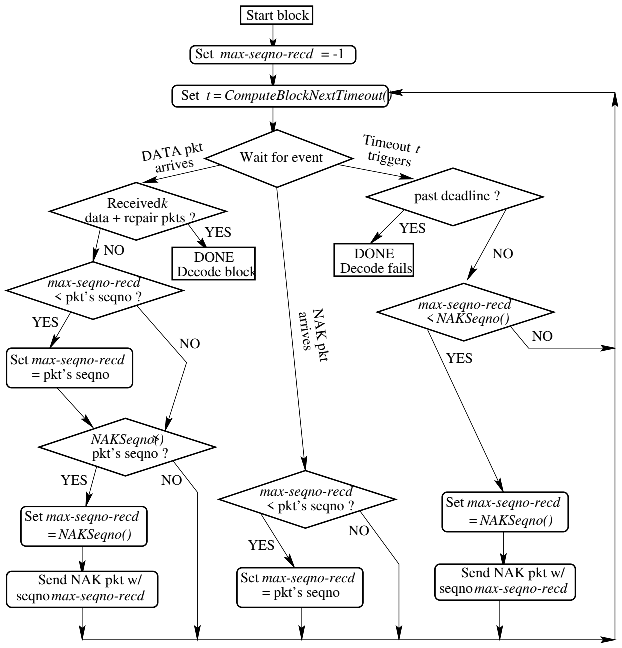

Figure 10: Receiver protocol: the different variations are constructed using different functions for $NAKSeqno()$ and $ComputeBlockNextTimeout()$.

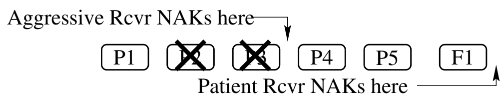

Figure 11: Patient vs. Aggressive Receiver: If the block size is 5 and the proactivity factor is 1.2, a patient receiver will allow time for receipt of all 6 transmissions of the initial packets before sending a NAK, regardless of which packets are lost. An aggressive receiver will NAK as soon as it detects 2 losses, since it knows it cannot recover the block with only 4 packets.

and time constraints imposed by the multicast group size in such a simulation.

The results presented here used a randomly constructed tree topology, where the minimum, average, maximum hops to the sender were 7, 15.1, and 25 respectively. The topology was constructed in a similar manner to the trees used in the round-by-round simulation in Section 4. We use a single tree for all multicast transmissions, including receiver NAKs. This corresponds a routing protocol like CBT. The results from our simulation are likely to be pessimistic for non-CBT-like routing, which would likely result in reduced NAK propagation delay between receivers.

We have experimented over a small set of such randomly generated topologies. With group sizes containing hundreds of receivers, we have observed similar results over the various random topologies. This is not surprising, since distance to the sender is the factor that we expect to have the largest impact on performance. Since we randomly select a

End Host processing rates used in simulation		
Operation	Time (μsec)	Performed by
Data Send	502	S
Data Process	487	R
NAK Send	87	R
NAK process	86	S & R
FEC code 1 packet	180 * block size	S (build) R (decode)
Data rate	5208	S

Table 2: Processing rates used in the simulation. An 'S' means the operation is performed by the sender, an 'R' means by each receiver.

large set of receivers within a randomly generated tree, their distances to the sender form a uniform distribution. All results presented in the remainder of the paper are based on a single, randomly generated tree topology.

Here we compare the performance of the patient receiver and aggressive receiver protocols in a simulation where the sender's proactivity factor remains fixed for the entire session, and examine how the proactivity factor affects session performance. First and last hop loss rates varied between 2% and 4% and backbone loss rates varied between 0.05% and 0.1% per router, giving end-to-end loss rates between 4.4% and 10.4%. Each router added a delay to a packet passing through it, where this delay fell in a range $[x, y]$: x was chosen from a uniform distribution between 5 ms and 6 ms for backbone routers and 0.5 ms and 3.5 ms for last hop routers. y was then chosen by adding a uniformly distributed value to x between 0 and 5 ms for backbone routers, and between 0 and 10 ms for last hop routers. To prevent out of order delivery of packets, a packet would always be delayed beyond its chosen value to prevent its arrival on a link before a previously injected packet. In other words, if packet A arrives before packet B at a router, and scheduled departure times are respectively t and t' with $t' < t$, then B is rescheduled to depart at time t plus 170 μs. Maximum round trip times varied between 74 ms and 294 ms for the various receivers.

Figure 12 gives results from 6 different runs of our simulation that run for 5 seconds of simulated time with a data rate of 1.5 Mb/s using packets of one kilobyte each (roughly 93 blocks, or 930 data packets are transmitted). In each simulation, 500 receivers appear in an identical network configuration. In the first three simulations, all receivers execute the patient protocol and the sender's proactivity factor is fixed for each simulation at a level of 1.0, 1.3, and 1.6. For the last three simulations, all receivers execute the aggressive protocol, and the proactivity factor is again set at a level of 1.0, 1.3 and 1.6. The protocols are run reliably, meaning that receivers continue to send NAKs until they have received all of the data reliably. End-host processing rates are presented in Table 2, and are taken from measurements from [6, 24].

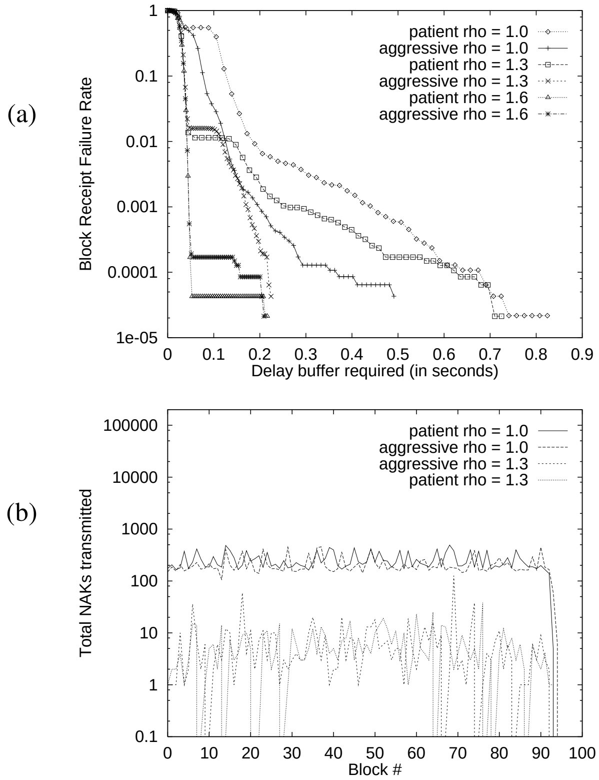

Figure 12: Examination of 500 aggressive and patient receivers with sender proactivity factors of 1.0, 1.3, and 1.6 in terms of (a) likelihood of failure to obtain an entire block vs. delay buffer, and (b) Number of NAKs transmitted per block.

We examined how the proactivity factor affects delay, the implosion factor, and wasted forward bandwidth and found that adding proactivity affected the performance of the asynchronous protocols in much the same way that it affected the synchronous protocols. We used the same metrics to measure implosion factor and forward bandwidth as we used for the round-based protocols in Section 4. However, the asynchronous nature of the protocols does not permit us to measure delay using the notion of rounds. Instead, the delay in reliably receiving a block is measured by taking the time at which the data packets could be decoded, and subtracting this from the latest time at which the first packet within the block could arrive, barring loss (i.e., the longest time it could take to arrive given its latency distribution in the model). We refer to this amount of time as the *delay buffer*.

We observed that for a proactivity factor of 1.4 or less, the forward bandwidth averaged around 15 packets, and increased linearly with the proactivity factor when the proactivity factor was at least 1.5. Figure 12(a) plots the fraction of blocks that cannot be recovered given the delay buffer.

We see that additional sender proactivity decreases the rate at which blocks fail to be decoded for a fixed amount of buffer. We also see that for a fixed proactivity factor, the fraction of blocks that can't be decoded given a fixed delay buffer is slightly smaller for the aggressive receiver than for the patient receiver. Figure 12(b) shows the number of NAKs received per block for proactivity factors of 1.0 and 1.3. We omit plotting results from the cases when the proactivity factor was 1.6, since for most blocks the number of NAKs sent was 0, and never rose above 2. As with the round-based protocol, a linear increase in the proactivity factor results in an exponential decay in the proactive factor.

5.4 Meeting real-time deadlines

(a)

(b)

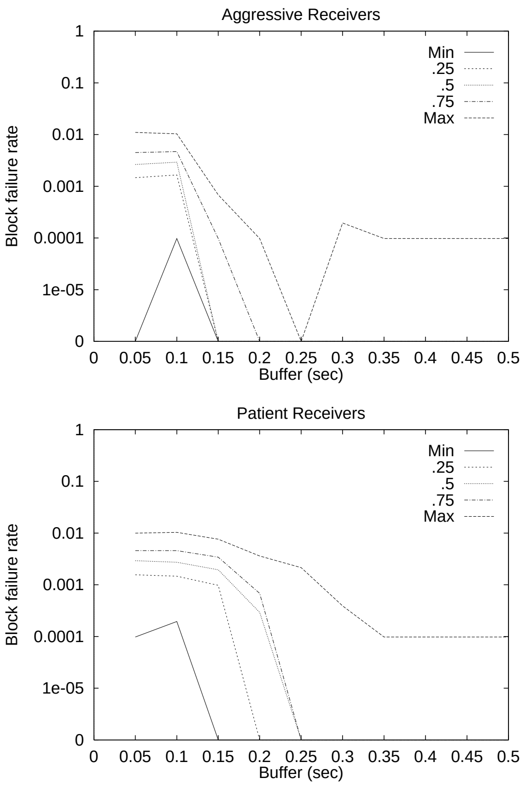

Figure 13: Block failure rates for meeting hard deadline with the block good-put guarantee set to a failure rate of .9999, where all receivers allow identical latency for retransmission. 10,000 blocks were transmitted.

Receivers can meet real-time deadlines with a high probability by determining a point in time before its deadline that gives a sufficient amount of time for it to send its NAK and receive the repairs before its deadline expires. We use a simple computation to determine what this time should be. Details on adapting the asynchronous protocols to meet real-time deadlines are discussed in [25].

(a)

(b)

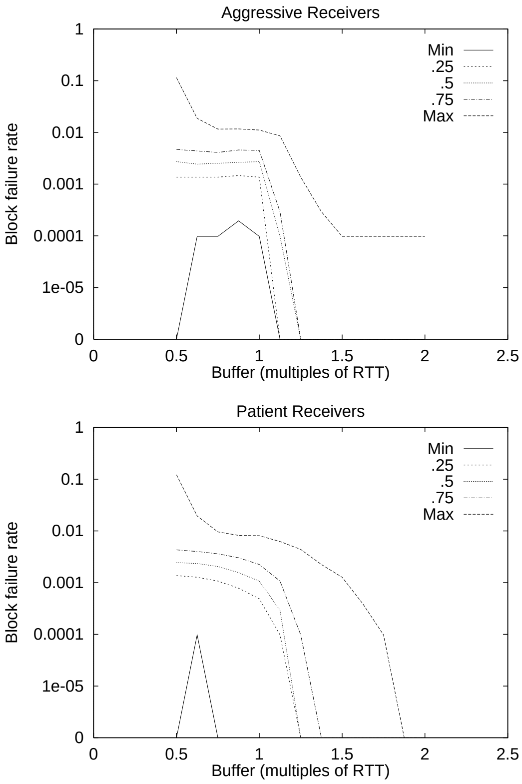

Figure 14: Block failure rates for meeting hard deadline with with the block good-put guarantee set to a failure rate of .9999, where each receivers' allowed latency is a multiple of its RTT to the sender. 10,000 blocks were transmitted.

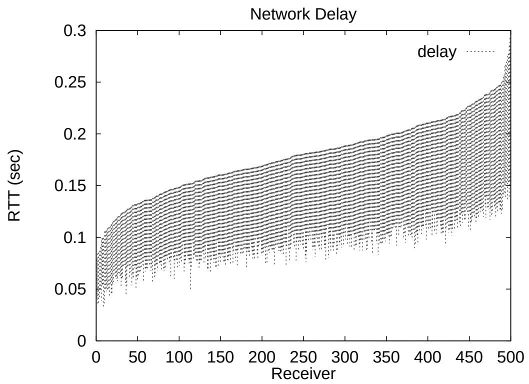

Figure 15: Variation in receiver round trip times to the source.

Results of applying the real-time algorithms are shown over the same topology of 500 receivers, with the sender using a fixed proactivity factor of 1.4. Figures 13(a) and 13(b) demonstrate the block failure rates for patient and aggressive receivers respectively in terms of the fraction of blocks that a particular receiver fails to decode, where each receiver employs an identical delay buffer. The plot labeled Min plots the fraction of lost blocks in each experiment by the receiver that observed the lowest rate of loss. The plot labeled Max plots the loss rate as seen by the receiver with the highest rate of loss, and the others plot the loss rate observed by receivers whose rates of loss lie at the 25%, 50%, and 75% quartiles when compared with the loss rates of the remaining receivers.

Figures 14(a) and (b) are similar to Figures 13(a) and (b), except that each point compares failure rates where the ratio of each receiver's delay buffer to its round trip time to the sender is identical. In all plots, the block good-put guarantee rate is set to .9999, such that the desired probability of failing to receive a sufficient number of packets to decode a block is .9999. For each point, we only transmit 10,000 blocks due to the time such an experiment takes to run on such a large scale simulation. Figure 15 displays the round trip time delays that can occur for the various receivers to and from the sender. The delay for each receiver is uniformly distributed along its vertical strip of the darkened interval. The delay shown here accounts only for the propagation time, and does not include any processing that might occur at end-hosts.

We observe from the graphs that given identical delay buffers, aggressive receivers typically reduce failure rates beyond their patient receiver counterparts. Figure 14 shows that for block good-put guarantees to be successful, receivers require at least a round trip time in order to be able to meet the guarantee. This is clearly due to the fact that any time less than a round trip time does not give a receiver enough time to send a NAK and elicit a response. We observe from Figure 13 that receivers with larger round trip times can expect some improvement in the ability to meet their block good-put guarantees a result of receivers with smaller round trip times imposing their own block good-put guarantees.

6 Varying the loss model

So far, we have demonstrated the performance of our protocol in a heterogeneous environment. We now examine how further variations in this environment affect performance.

We begin by reexamining two assumptions that were made within our model. All previous experiments assumed that NAKs sent by receivers were never lost. We considered what happens if NAKs are dropped at a router according to a Bernoulli process identical to that used to describe data and repair packet losses. We found that for large multicast groups, such loss had no noticeable impact. We believe this is due to the fact that the protocol limits, but does not suppress, all redundant loss requests. Therefore, there is a high probability that even when NAKs can be lost, there are enough duplicate NAKs sent that at least one NAK will still reach the sender.

We also consider the impact of bursty loss at each router has through the use of a continuous two-state loss model,

(a) (b) (c)

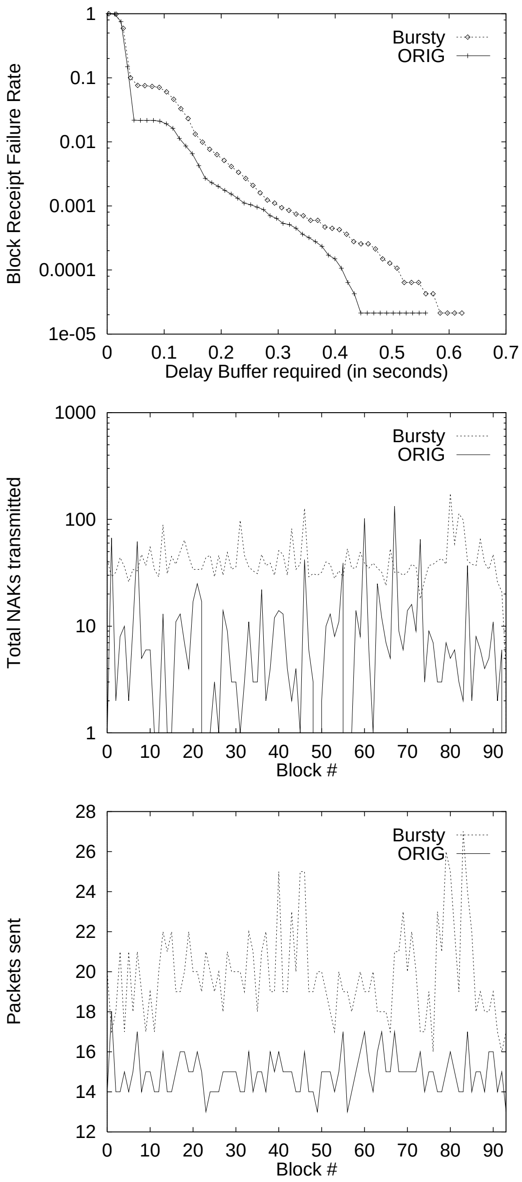

Figure 16: How results change as we add features of a more realistic network in terms of (a) good-put failure rate vs. buffer latency, (b) NAKs received per block, (c) packets sent by the sender that weren't needed by any receivers (i.e., wasted forward bandwidth), and (d) Total number of packets sent per block.

where the expected length of a burst at each router is 6 ms, while keeping the overall probability of dropping a packet fixed. Our algorithm that prevents reordering creates packet trains where the time lag between packets is 1.7 ms, thus the expected loss burst length can cover 4 packets passing through a router.

Figure 16 demonstrates the impact that bursty loss has on protocol performance with all other network features identical to those used in Section 5. Each plot here is for the case that all receivers use the aggressive receiver algorithm, and a proactivity factor of 1.3. In each figure, plots labeled ORIG assume a Bernoulli process for data and repair loss, whereas those labeled Bursty use the bursty loss model. As in Figures 12(a) and 12(b), Figures 16(a) and 16(b) plot performance in terms of block receipt failure rate and total NAKs transmitted, respectively. Figure 16(c) plots the total number of packets that were transmitted for a block.

We observe that a network that exhibits high amounts of bursty loss can alter the performance substantially. We see that much of the time, an increasing number of repairs must be sent to satisfy all receivers. Also, latencies and the number of NAKs transmitted increases. Similar results hold for the patient receiver algorithm. One way to minimize the impact of burstiness is to increase the block size used. However, increasing the blocksize increases the amount of time between the first and last packets within a block, which increases the expected minimum latency between the first lost packet and the first repair.

7 Conclusions and Future Work

We have presented a technique that uses a hybrid of FEC and ARQ approaches that can repair data in an extremely efficient manner. We show that it is possible to make significant improvements over current commonly used techniques, and that FEC can be used to keep bandwidth requirements and NAK implosion to a minimum. We show that efficient support for large multicast groups can be performed in an end-to-end, unscoped manner, without significantly increasing bandwidth utilization over other unscoped FEC techniques, and uses considerably less bandwidth than unscoped ARQ approaches such as SRM. Also, we show that through minor bandwidth increases, receivers can improve upon performance in terms of expected time or reliability and rate of failure to meet their deadlines. We present a variety of protocols that use the technique and show their success in medium-sized multicast groups in heterogeneous networks through simulation. A prototype implementation has been developed and has been tested on the MBone. A detailed report of the results can be found in [26].

We have developed and experimented with an adaptive mechanism that is executed by the sender which varies the its proactivity factor. The mechanism works well for the network conditions that we model in this paper. However, we are exploring alternative mechanisms that are simpler and more intuitive. For this reason, we do not present an adaptive mechanism at this time.

Acknowledgments

We would like to thank Rohan Kumar who has converted our simulation code into a prototype implementation, the anonymous reviewers, Jon Crowcroft, and Joerg Nonnenmacher for their helpful comments.

References

[1] S.E. Deering, D.R. Cheriton. *Multicast Routing in Datagram Internetworks and Extended LANs*, ACM Trans. on Computer Systems , 8, 2, pp. 85-110, May 1990.

[2] T. Ballardie, J. Crowcroft, P. Francis. "Core based trees (CBT) An architecture for Scalable Interdomain Multicast Routing" Proceedings of ACM SIGCOMM'93, pp. 85-95, September 1993.

[3] Richard E. Blahut, *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, MA, 1983.

[4] Anthony J. McAuley, *Reliable Broadband Communication Using a Burst Erasure Correcting Code*. Proceedings of ACM SIGCOMM'90, pp. 297-306, Sept. 1990 Philadelphia, PA.

[5] Luigi Rizzo, *Effective Erasure Codes for Reliable Computer Communication Protocols*, Computer Communication Review, April 1997.

[6] Jorg Nonnenmacher, Ernst Biersack, and Don Towsley, *Parity-Based Loss Recovery for Reliable Multicast Transmission*, Proceedings of ACM SIGCOMM'97, pp. 289-300, Sept. 1997 Cannes, France.

[7] S. Floyd, V. Jacobson, C. Liu, S. McCanne, L. Zhang. *A reliable multicast framework for light-weight sessions and application level framing*, IEEE/ACM Trans. on Networking, 1998.

[8] J. C. Lin and Sanjoy Paul. *Reliable Multicast Transport Protocol (RMTP)*, IEEE JSAC, 407, 3, 1414-1424,April 1997.

[9] Christos Pappadopoulos, Guru Parulkar, and George Varghese, *An Error Control Scheme for Large-Scale Multicast Applications*. Proceedings of IEEE INFOCOM'98, San Francisco, CA.

[10] Rex Xi Xu, Andrew C. Myeres, Hui Zhang, and Raj Yavatkar, *Resilient Multicast Support for Continues Media Applications*, NOSSDAV 1997.

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

[12] Xue Li, Sanjoy Paul and Mostafa Ammar, *Layered Video multicast with Retransmissions (LVMR): Evaluation of Hierarchical Rate Control*, NOSSDAV, 1997.

[13] Matthew T. Lucas, Bert. J Dempsey, and Alfred C. Weaver, *MESH: Distributed Error Recovery for Multimedia Streams in Wide-Area Multicast*, Proceedings Sixth International Conference on Computer Communications and Networks (IC3N), 1997

[14] Luigi Rizzo and Lorenzo Vicisano, *RMDP: An FEC-based Reliable Multicast Protocol for Wireless Environments*, Mobile Computing and Communications Review, Volume 2, Number 2, April 1998.

[15] D. DeLucia, K. Obraczka, *Multicast Feedback Suppression Using Representatives*, Proceedings of IEEE INFOCOM'97, Kobe, Japan, March 1997.

[16] S. Kasera, J. Kurose, D. Towsley, *A Comparison of Server-Based and Receiver-Based Local Recovery Approaches for Scalable Reliable Multicast*, Proceedings of IEEE INFOCOM'98, San Francisco, CA, March 1998.

[17] J. Nonnenmacher, M. Lacher, M. Jung, E. W. Biersack and Georg Carle, *How bad is reliable multicast without local recovery?*, Proceedings of IEEE INFOCOM'98, San Francisco, CA, March 1998.

[18] T. Speakman, D. Farinacci, S. Lin, and A. Tweedly, *Pretty Good Multicast (PGM) Transport Protocol Specification*, Internet Draft draft-speakman-pgm-spec-00.txt, January 1998.

[19] M. Yajnik, J. Kurose, D. Towsley, *Packet Loss Correlation in the MBone Multicast Network*, IEEE Global Internet Conference. London, UK, November 1996.

[20] M. Handley, *An Examination of MBone Performance*, Technical Report, UCL and ISI, January 1998.

[21] R. Kermode, *Scoped Hybrid Automatic Repeat Request with Forward Error Correction (SHARQFEC)*, to appear in ACM SIGCOMM'98, Vancouver, CA, September 1998.

[22] K. Perumalla, A. Ogielski and R. Fujimoto, *MetaTeD: A Meta Language for Modeling Telecommunication Networks*, GIT-CC-96-32, Technical Report, College of Computing, Georgia Institute of Technology, 1997.

[23] H.R. Deng and M.L. Lin, *A Type I Hybrid ARQ system with Adaptive Code Rates*, IEEE Transactions on Communications, COM-43 (2/3/4):733-737, February/March/April 1995.

[24] S. Kasera, J. Kurose, and D. Towsley, *Scalable Reliable Multicast Using Multiple Multicast Groups*, CMPSCI Tech Report 96-73, University of Massachusetts, October, 1996 (A shorter version of this paper appeared in ACM SIGMETRICS '97).

[25] D. Rubenstein, J. Kurose, and D. Towsley, *Real-Time Reliable Multicast Using Proactive Forward Error Correction*, CMPSCI Tech Report 98-19, University of Massachusetts at Amherst, March 1998.

[26] R. Kumar, *Design and Implementation of a Proactive Real-Time Reliable Multicast Protocol using Forward Error Correction (FEC)*, Masters Thesis, University of Massachusetts at Amherst, May 1998, in preparation.

[27] Jean Bolot, Private Communication, 1998.

Appendix

The following lemma applies for generating block good-put rates within hard deadlines. If C is the event that a block is decodable, then ensuring that $P(C) \geq \mathcal{P}$ gives a block good-put rate of at least $\mathcal{P}$. In section 4.1, we propose that a receiver wait until the last possible moment to send a final NAK for that block. At this point, the receiver knows the number of packets, l, that it still needs to recover the block. Regardless of the number of packets received from this final request, a receiver does not request any further NAKs.

Lemma 1 *If the receiver chooses m such that $\mathcal{D}(m,l) \geq (\mathcal{P} - \mathcal{D}(n,k))/(1-\mathcal{D}(n,k))$, then $P(C) \geq \mathcal{P}$.*

Proof: Let N be an R.V. that equals the number of packets from the block sent by the sender prior to the final NAK transmission, and L be an R.V. that equals the number of packets required by the receiver at this time. Then:

$$\begin{aligned} P(C \wedge (N=n)) &= \sum_{l=0}^{k} P(C \wedge (L=l) \wedge (N=n)) \\ &= P(C \wedge (L=0) \mid (N=n))P(N=n) + \\ &\sum_{l=1}^{k} P(C \mid (N=n) \wedge (L=l))P((N=n) \wedge (L=l)). \end{aligned}$$

If $L=0$, then C is guaranteed to hold, since the receiver needs no further packets to complete the block. Thus, $P(C \wedge L = 0 \mid N=n) = P(L=0 \mid N=n) = \mathcal{D}(n,k)$. It is also the case that if the final NAK requests m packets, then $P(C \mid N = n \wedge L = l) = \mathcal{D}(m,l)$. Selecting m as stated in the lemma therefore gives us that $P(C \mid N = n \wedge L = l) = \mathcal{D}(m,l) \geq (\mathcal{P} - \mathcal{D}(n,k))/(1-\mathcal{D}(n,k))$. Thus:

$$\begin{aligned} & P(C \wedge N=n) \geq \mathcal{D}(n,k)\, P(N=n) + \\ & \quad \sum_{l=1}^{k} \frac{\mathcal{P} - \mathcal{D}(n,k)}{1-\mathcal{D}(n,k)} P(L=l \mid N=n)\, P(N=n) \\ &= \mathcal{D}(n,k)\, P(N=n) + \frac{\mathcal{P} - \mathcal{D}(n,k)}{1-\mathcal{D}(n,k)} \times \\ & \qquad P(N=n) \sum_{l=1}^{k} P(L=l \mid N=n) \\ &= \mathcal{D}(n,k)\, P(N=n) + \frac{\mathcal{P} - \mathcal{D}(n,k)}{1-\mathcal{D}(n,k)} \times \\ & \qquad P(N=n)(1 - P(L=0 \mid N=n)) \\ &= \mathcal{D}(n,k)\, P(N=n) + \\ & \qquad \frac{\mathcal{P} - \mathcal{D}(n,k)}{1-\mathcal{D}(n,k)}\, P(N=n)(1-\mathcal{D}(n,k)) \qquad (1) \\ &= \mathcal{P}P(N=n). \end{aligned}$$

It follows that $P(C) = \sum_{i=0}^{\infty} P(C \wedge N=n) \geq \sum_{i=0}^{\infty} \mathcal{P}P(N=n) = \mathcal{P}$. ∎