Real-Time Reliable Multicst Using Proactive Forward Error Correction

D. RUBENSTEIN, J. KUROSE
and D. TOWSLEY

CMPSCI Technical Report 98-19

March 1998

Real-Time Reliable Multicast Using Proactive Forward Error Correction*

Dan Rubenstein, Jim Kurose, and Don Towsley
Computer Science Department
University of Massachusetts
Amherst, MA 01003
{drubenst, kurose, towsley}@cs.umass.edu

Technical Report 98-19
Department of Computer Science
March 1998

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. This paper presents 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 the attempts of 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 can also send 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 delays, 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

*This material was supported in part by the National Science Foundation under Grants NCR-95-08274, NCR-95-27163 and CDA-95-02639. 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 or the University of Massachusetts.

[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.

source forward repairs before they are required that differentiates our work from other reliable multicast protocols that make use of FEC. Obtaining all repairs from the data source simplifies the protocol compared to schemes that distribute the repairing source. The single source model also makes the protocol robust over large variations in multicast group topology. Our results through simulation indicate that this technique 1) offers a significant decrease in the expected time of reliable receipt of data, 2) can be used to meet hard real-time deadlines of reliable delivery of data with probabilities extremely close to one, 3) reduces implosion, 4) competes well with other protocols in terms of network bandwidth required, 5) can reduce bandwidth requirements for meeting real-time deadlines versus 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 for networks containing large multicast groups, where receivers are at varying distances from the sender, with varying end-to-end loss rates and latencies.

The focus of this paper is to demonstrate the effectiveness and simplicity of using the proactive FEC technique on its own. For this reason we avoid examining protocols that combine proactive FEC with other techniques such as local recovery, additional router support, or the use of multiple multicast groups that could potentially further improve performance. We introduce a protocol that uses proactive FEC to deliver high rate data streams to large groups of receivers with bounded loss for hard deadlines, and no loss for soft deadlines. These protocols are shown to require little state at both sender and receiver, do not require additional router support, and can communicate over a single multicast group. We use simulation to demonstrate the protocols' effectiveness in heterogeneous environments for large multicast groups.

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 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. Further details are provided in the Appendices.

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 the impact of feedback implosion, reliable multicast protocols have incorporated random delays to reduce redundant NAK and repair transmissions [7], but 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. A third 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. The protocol's effectiveness depends on the source's ability to select a good set with its limited knowledge of the group topology and network loss characteristics.

There has also been a 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, since repairs occur as a series of unicast transmissions between receivers. Having receivers substitute for routers further increases latency.

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 over that which is consumed using stand-alone ARQ.

[17] presents a preliminary analysis that compares the benefits of combining local recovery with an FEC/ARQ hybrid technique. [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 through multicast group joins and leaves. However, present join-leave latencies for multicast groups make the approach too bandwidth inefficient to support real-time applications.

3 Overview of Forward Error Correction (FEC)

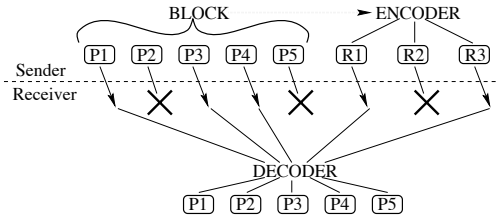


Figure 1: A sample decoding using a block built from 5 packets.

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 network layer. We do not present how the coding techniques work, and the reader is referred to [4] for an excellent description of the mathematics behind the Reed-Solomon coding techniques that we employ here. The following description is sufficient for an understanding of 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 are data packets from the block, and/or repairs generated for the block.[2] An example usage of the FEC encoder and decoder is shown via a unicast example 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.

A detailed discussion of packet-level FEC techniques can be found in [6]; 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. Hybrid approaches that combine FEC and ARQ have been proposed and classified for repairing loss and noise at the bit-level [21]. The type I hybrid approach suggests sending data and proactively sending FEC repairs, but retransmitting data directly if these repairs are insufficient. Type II hybrid FEC uses FEC to send repairs based on retransmission requests, and sends nothing proactively. Our transmission sends repairs proactively making it similar to the type I approach, but repairs are also sent using FEC.

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

[2] Requires a slight variation on the Reed-Solomon technique, which incorporates negligible amounts of additional processing.

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) that are sent by the sender per block. Because the sender multicasts each packet it sends, the bandwidth is linearly proportional to the number of packets transmitted per block, barring internal packet losses.

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. We study protocols that do not force receivers to perform according to the requirements of the slowest receiver in Section 5.

As in most protocols that use FEC, the sender packetizes its data for delivery, and builds blocks from these packets of size k. 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.

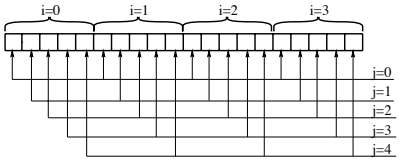


Figure 2: Partitioning a data set into 4 blocks of size 5 so that repairs can be generated using FEC.

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 purposes 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. The block identifier within a packet permits the transmission of the various blocks to be interleaved, 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 that is partitioned into 4 blocks of size 5.

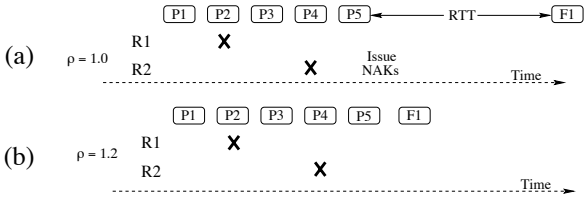


Figure 3: Illustration of possible savings in time and reduction of feedback through proactivity

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,[3] 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. 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 sending out a number of repair packets that satisfies the maximum from all of the receivers' requests for that round.[4] This process continues until

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

[4] 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 that need to be generated.)

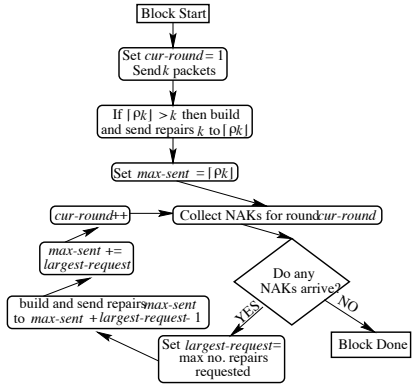


Figure 4: The round-based sender algorithm

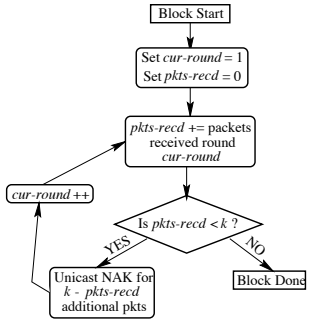


Figure 5: The round-based receiver algorithm

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 demonstrates the potential benefits of setting the proactivity factor to a value larger than one. In 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 3(b), if $\rho = 1.2$, then a repair packet is transmitted with the data, and the receivers do not need to request repairs and 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 fewer repair requests being returned 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 is generally significantly smaller than what it is 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.

A receiver can itself 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 within some fixed probability. In 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 than what it actually needs in order 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:

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

Since $\lim_{m\to\infty} \mathcal{D}(m,n) = 1$ (a simple proof is presented in Appendix A), 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:

$$\mathcal{D}(m,n) \quad = \quad \mathcal{D}(m-1,n) + \binom{m-1}{n-1} (1-p)^n p^(m-n). \tag{2}$$

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 Appendix A.

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 needs to choose 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 show in Appendix A 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 both temporally uncorrelated loss, as well as for loss that can be modeled using the two-state loss model. The receiver must simply use the appropriate formulation of $\mathcal{D}()$.

By attempting to meet a hard deadline, receivers will often send requests for more repairs than if no hard deadline existed, 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

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 we were able to accomplish via a mathematical analysis, and makes it easier than it would be through experimentation to observe effects of large multicast sessions. The analysis is restricted to a star topology network and is presented in Appendix F. Results presented here are simulated over a tree topology, where nodes on the interior of the tree represent routers with

Amortized cost to deliver a packet reliably					
	Multicast Group Size				
Block size	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. Note that a block size of 1 is identical to having no FEC, where each multicast retransmission is the sole packet within the block.

downstream fan-outs 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 .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 Appendix B. We note that an alternative means of building a realistic multicast tree is to first generate a realistic network topology using a network generating tool, choose a sender, and construct the shortest path tree within that network. We anticipate that the tree constructed via such a process would be similar to a tree produced from our simple algorithm.

Once a tree has been created with 10,000 receivers, the number of receivers is varied by selecting a subset of receivers in the tree of the desired size, and having only those receivers participate in the multicast session. For the results in this section, we randomly constructed a single tree, and 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. Other tree topologies, receiver placements, loss models, and block sizes are considered in Appendix D

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.

Before examining the benefits that can be obtained via proactivity, we examine how varying the block size affects the expected forward bandwidth used by any protocol that multicasts repairs to an entire group. The normalized, expected forward bandwidth gives the expected cost, in terms of packet transmissions, for delivering a single data packet. The value can be obtained by taking the expected cost of delivering a block, and dividing this cost by the block size. Table 1 gives the normalized, expected forward bandwidth needed to reliably deliver a packet to all receivers as a function of receiver population and block size. 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 an ARQ-only protocol.

Figures 6(a), 6(b), and 6(c) respectively show the expected number of rounds, implosion factor, and forward bandwidth 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. That is:

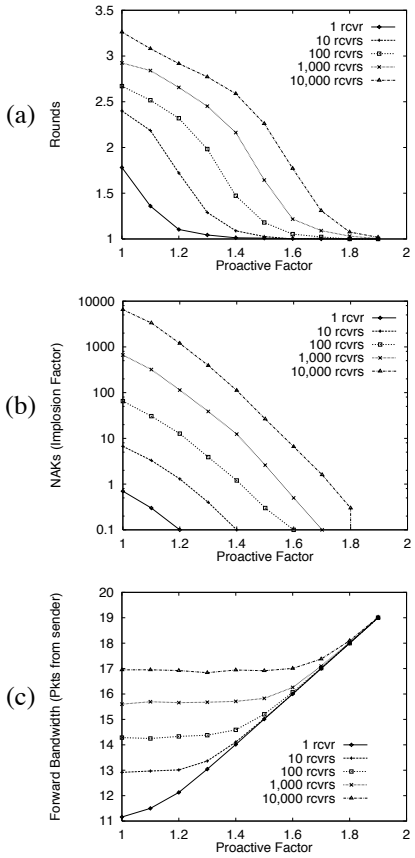


Figure 6: Results from a round-based simulation on a tree topology of receivers. Note that the y-axis of (b) is plotted on a log-scale.

By using proactive FEC, we can 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 to be sent; 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. 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 7 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.

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.

To summarize, the sender can increase the proactivity factor to a point where implosion is significantly reduced, as well as reducing the expected delay of reliable delivery for a receiver, and that these improvements can often be achieved using insignificant amounts of additional bandwidth.

We now examine the performance of the protocol in the presence of realtime 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

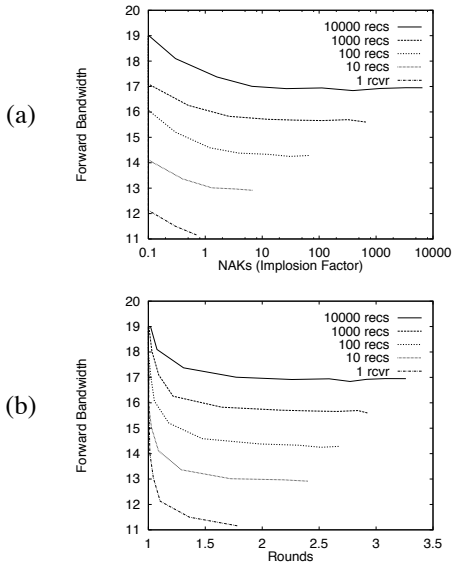


Figure 7: Tradeoff between forward bandwidth and NAK implosion and number of rounds.

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. However, the total expected bandwidth is more than that typically used to deliver data reliably over all rounds. 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 often chooses not to send a NAK, even when it has failed to complete the block. This apparently more than compensates for the extra packets it sometimes requests in the last round when it does wish to continue to attempt recovery.

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. In separate experiments, we modeled each router as a mutually independent Bernoulli process, as well as by a two-state mutually independent process. The latter experiments were done to examine the effects of losses that occur at routers in bursts. The two-state loss model was set to an equilibrium state at the start of each round; furthermore we did not consider any loss correlation between the various blocks of a data stream. Finally, we also examined variations on 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. Most noticeably, low loss rates, highly correlated (i.e. only upstream loss), and bursty losses decrease the flattening effect observed in the forward bandwidth in Figure 6 for large receiver sets and low levels of proactivity. Instead, forward bandwidth increases at a slow, but observable rate, even for low levels of proactivity. However, this increase in forward bandwidth remains negligible when compared to the forward bandwidth used in traditional ARQ approaches, such that conclusions drawn from the network model we present here are similar to what one would conclude from observing these other models mentioned. Details are provided in Appendix D.

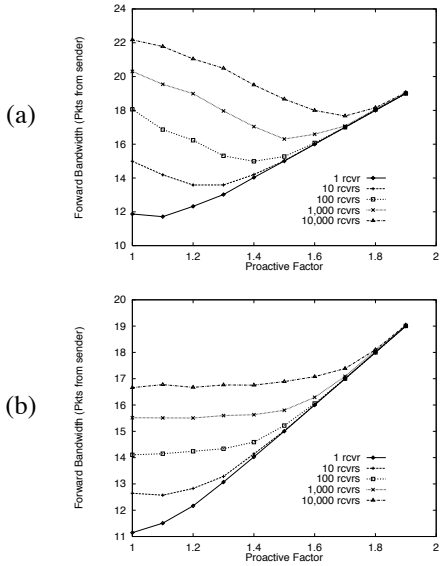


Figure 8: Bandwidth usage to provide (a) last round and (b) block good-put guarantees of $1 - 10^{-6}$ by the end of 2 rounds.

5 Asynchronous Protocol

In Section 4, we demonstrated the effectiveness of the proactive technique on 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. Doing so allows each receiver to maximize its own individual performance.

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.

In this paper, we assume that all receivers multicast their NAKs for the purposes of suppression. We examine the impact on the protocols where all NAKs are unicast in Appendix E

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 fashion that satisfies the request in the repair. 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. To operate effectively, the sender must be made aware of specific receiver information, or use some heuristic to guess at what should be done. It is likely that this approximation will cause the sender to ignore a request that should not have been ignored.

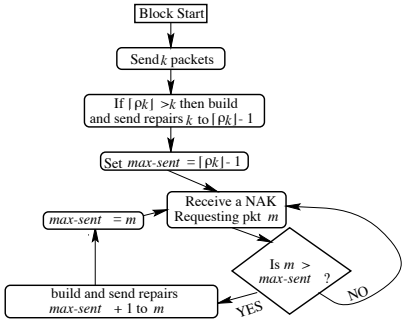


Figure 9: The sender protocol

5.2 Protocol: Receiver's algorithm

The state diagram that demonstrates the receiver's behavior in 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, we have found that the algorithm operates effectively if its state maintains only the following pieces of information for each block actively being transmitted: 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. This per-block state information can be obtained directly during the normal course of the algorithm. Additionally, the receiver must maintain an upper bound on the round trip time to the sender. A conservative estimate of this time is sufficient. To meet hard guarantees, an 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 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 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 the receiver, but was lost by the sender. It can also occur if repairs incur larger than expected delay 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. Also, there are several techniques that can be employed to reduce the damage that can occur

due to inaccurate estimates. For instance, the sender can include adding burst bits in repair packets that specify the number of packets that follow the current repair in the burst.

To prevent unnecessary NAK transmissions, after receiving a NAK, a receiver waits a conservative amount of time before reacting to a loss. When 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 the estimate.

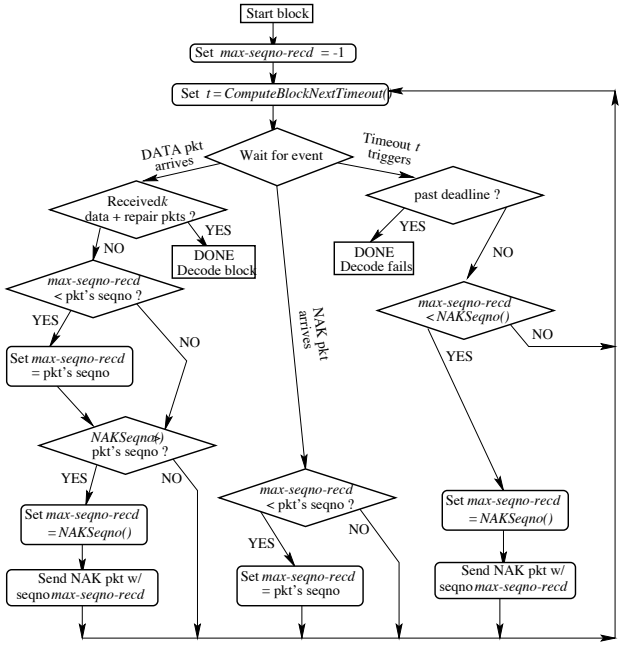


Figure 10: Receiver protocol: the different variations are constructed using different functions for $NAKSeqno()$ and $ComputeBlockNextTimeout()$.

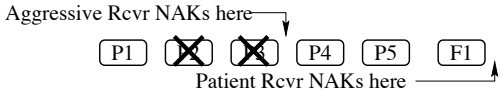


Figure 11: Patient vs. Aggressive Receiver: If the block size is 5 and the proactivity factor is 1.2, a patient receiver will allow time for receipt of all 6 transmissions of the initial packets before sending a NAK, regardless of which packets are lost. An aggressive receiver will NAK as soon as it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

We have experimented with two versions of the receiver algorithm that differ only in a variation of the $ComputeBlockNextTimeout()$ function. The first variation we refer to as the *patient receiver protocol*. The patient receiver only issues a NAK for a block when it believes 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 is referred to as the *aggressive receiver*. The aggressive receiver sends a NAK whenever it believes 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.

These two versions of the receiver protocol can operate effectively with a state that contains seven fields per block for the patient receiver, and eight fields for the aggressive receiver. For a block size of 10, this corresponds to less than one field per packet in transmission.

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.

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 [20]. The group size is restricted to 500 due to the memory and time constraints imposed by the multicast group size on 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 [2]. The results from our simulation can be considered as pessimistic bounds for protocols that can construct multiple trees depending on the source of the multicast transmission. Such protocols would reduce the propagation delay of NAKs between receivers, and would reduce unnecessary NAK transmissions.

We have experimented over a small set of such randomly generated topologies. With group sizes containing hundreds of receivers, we have found results to be similar 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 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 the quality of the session. For the experiments presented here, there were 500 active receivers in a session. 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 picked 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. This time corresponds roughly to the minimum gap in arrival times between kilobyte packets over a 45 Mbs link, such as a T3. Preventing reordering in this fashion leads to packet trains, which is a realistic phenomenon in networks. 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 Mbs 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.

We present results of how the proactivity factor affects delay, the implosion factor, and wasted forward bandwidth. The delay in reliably receiving a block is computed by taking the time at which the data packets could be decoded, and subtracting this from the latest time at which the packet with sequence number 0 could arrive, barring loss (i.e.

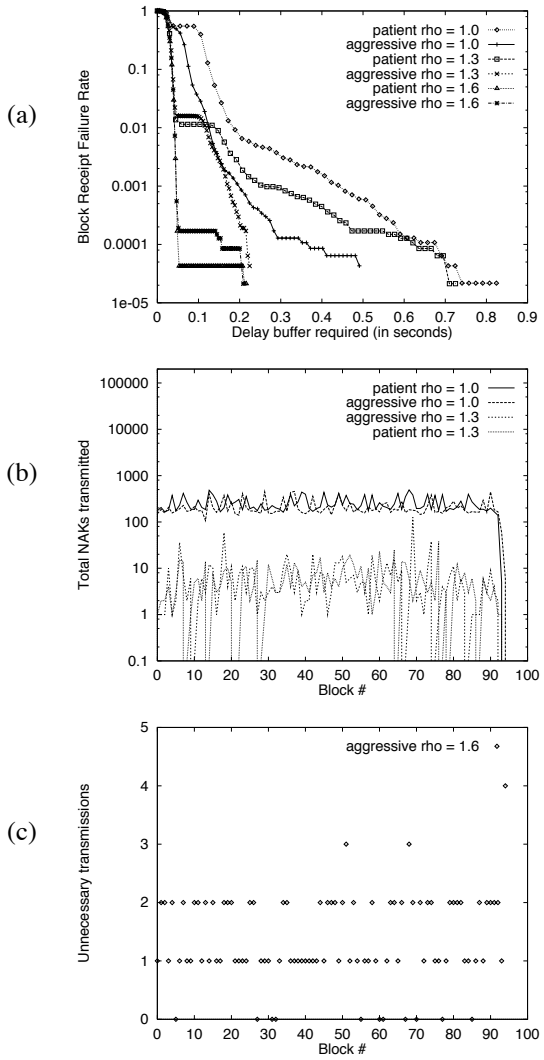


Figure 12: Examination of 500 aggressive and patient receivers with sender proactivity factors of 1.0, 1.3, and 1.6 in terms of (a) likelihood of failure to obtain an entire block vs. delay buffer, (b) Number of NAKs transmitted per block, and (c) packets sent per block that weren't needed by any receivers (wasted bandwidth)

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*. 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 see that increasing proactivity has a dramatic effect on the number of NAKs received. Regardless of whether the aggressive or patient protocol was used, between 100 and 200 NAKs were received per block when the proactivity was set to 1.0. For a proactivity factor of 1.3, the number of NAKs lies between 1 and 20 for the patient protocol. The aggressive receiver protocol has roughly the same performance for this level of proactivity, except that occasionally, on the order of 50 NAKs occur for a block. 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.

Figure 12(c) shows the number of packets that were sent by the sender beyond what was needed by any receiver. This shows the amount of wasted forward bandwidth, since these packets could have been omitted from transit and

all receivers would still be able to obtain reliable delivery of the block. For purposes of clarity, we illustrate the unnecessary packets for only the experiment when the proactivity factor is set to 1.6 and the receivers are aggressive. We see that even with such a high proactivity factor, the expected number of unnecessary transmissions per block remains quite low. We also note that the number of unnecessary transmissions is roughly the same for the patient and receiver protocols, since the extra packets are due to the proactivity factor being set higher than is most often necessary. For lower levels of proactivity, the number of blocks for which the sender transmits unnecessary packets is negligible.

5.4 Meeting real-time deadlines

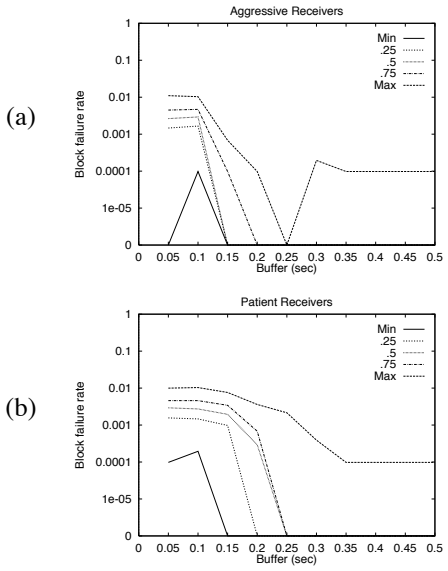


Figure 13: Block failure rates for meeting hard deadline with the block good-put guarantee set to a failure rate of .9999, where all receivers allow identical latency for retransmission. 10,000 blocks were transmitted.

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 is. The computation is performed by subtracting from the time of the deadline a pessimistic estimate of the time that allows delivery of all packets to be requested, as well as a sufficient amount of time for decoding. This computation's details are presented in Appendix C. Both aggressive and patient receiver protocols can be adapted to meet hard real-time deadlines by altering the $ComputeBlockNextTimeout()$ function to always perform a timeout when this last transmission can take place. At that time, the receiver sends its final NAK, where $NAKSeqno()$ uses the hard deadline formulas presented in section 4.1 to determine the number of additional packets it requires the sender to transmit. We have added a requirement that the receiver not send its final NAK until after all the original data packets have been given ample time to arrive. This is to prevent a receiver with a very short delay buffer from sending a large NAK for every block.

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 reliability 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.

Plots 14(a) and (b) are similar to 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 probability of failing to receive a sufficient number of packets to decode a block is desired

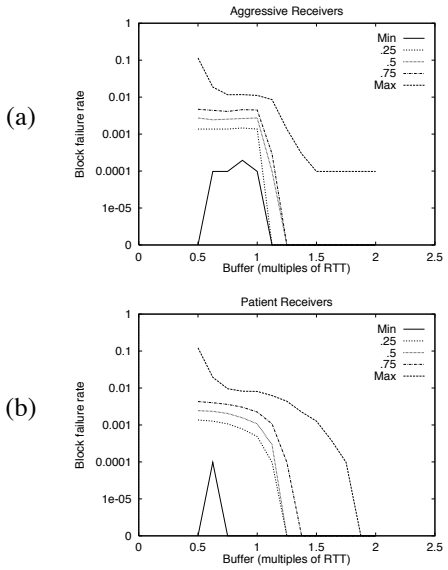


Figure 14: Block failure rates for meeting hard deadline with with the block good-put guarantee set to a failure rate of .9999, where each receivers' allowed latency is a multiple of its RTT to the sender. 10,000 blocks were transmitted.

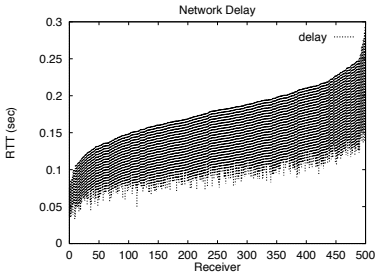


Figure 15: Block failure rates for meeting hard realtime deadlines.

to be .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 illicit 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 Adapting to network conditions

So far, we have demonstrated the performance of our protocol in a heterogeneous, but controlled environment. We now examine how certain 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

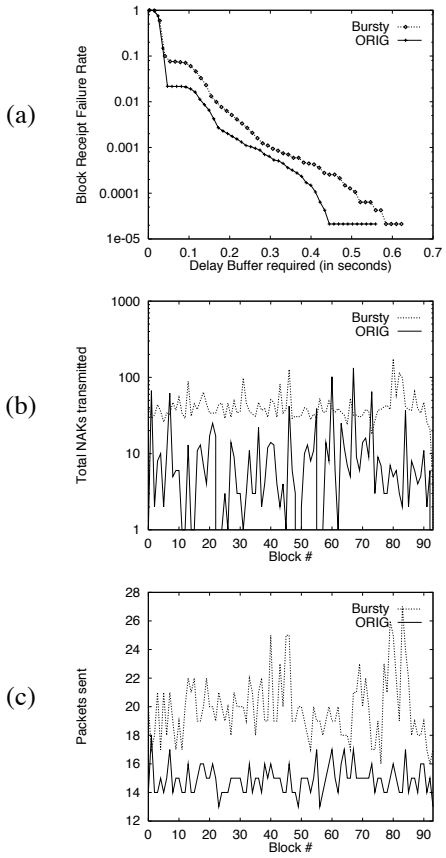


Figure 16: How results change as we add features of a more realistic network in terms of (a) good-put failure rate vs. buffer latency, (b) NAKs received per block, (c) packets sent by the sender that weren't needed by any receivers (i.e. wasted forward bandwidth), and (d) Total number of packets sent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

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 is a sufficient amount of redundancy such 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, where the expected length of a burst 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 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 the proactivity factor of 1.3. In each figure, plots labeled ORIG assume a Bernoulli process for data / 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.

6.1 Intra-session group membership changes

As we have observed in Section 4, an appropriate value for the sender's proactivity factor depends on the group-size. We have developed and experimented with an adaptive mechanism that is executed by the sender which varies the its proactivity factor with a constant increase in state per block, such that the state for a protocol using this mechanism remains independent of multicast group size and topology. 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.

6.2 Congestion Control

We do not examine mechanisms for congestion control in this paper. However, we point out that:

1. FEC combined with ARQ reduces repair overhead compared to traditional ARQ approaches,
2. one can easily incorporate known methods to perform congestion control in the case of soft real-time traffic,
3. congestion control for reliable multicast with hard realtime deadlines is still an open issue.

6.3 Preliminary Experimental Results

We have developed a prototype implementation, with which we have begun to perform experiments on the MBone. The current prototype transfers 33 byte data packets with a period of .03 seconds between transmissions. Repairs are also 33 byte packets, and in the current version of the implementation, no FEC encoding or decoding is performed: the byte stream in the repair has no relevance, other than the sequence number. When the sender issues a repair, it is sent immediately into the network.

We ran several experiments at several different times over the Internet using small multicast sessions containing fewer than 10 receivers. Because the protocol was designed for large multicast groups, these experiments were only meant to show proof of concept. However, we observed patterns of loss on the MBone that would hinder performance of any protocol that relies solely on multicast transmissions. We observed loss between the sender and some of the receivers that were periodic and lasted for periods of time close to a second. As a result, all transmissions and proactive repairs were lost, as well as the NAKs that were multicast to notify the sender of the loss. We are considering ways in which the protocol can be adapted so that it that might provide more suitable performance over the MBone.

7 Conclusion

We have presented a technique that uses a hybrid of FEC and ARQ approaches that can repair data extremely in an extremely efficient manner. Furthermore, the technique can be used to reduce two critical issues in reliable delivery of data: expected time of reliable delivery, or probability of reliable delivery before a fixed deadline. 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 the 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.

Acknowledgments

We would like to thank Rohan Kumar who has converted our simulation code into a prototype implementation as well as the researchers at University of Maryland, Washington University in St. Louis, Georgia Tech, University of Kentucky, Carnegie Mellon University, University of Washington, University College London, UCLA, and the Swedish Institute of Computer Science for allowing us to use their machines for our experiments.

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" Proc. ACM SIGCOMM '93 , pp. 85-95, ACM, 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*. ACM SIGCOMM '90, Sept. 1990 Philadelphia, pp. 297-306.

[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*, ACM SIGCOMM '97, Sept. 1997 Cannes, France, pp. 289-300.

[7] S. Floyd, V. Jacobson, C. Liu, S. McCanne, L. Zhang. *A reliable multicast framework for light-weight sessions and application level framing*, to appear in IEEE/ACM Trans. on Networking, 1997.

[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*. To appear in 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*, Proc. 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.

[16] S. Kasera, J. Kurose, D. Towsley, *A Comparison of Server-Based and Receiver-Based Local Recovery Approaches for Scalable Reliable Multicast*, to appear in IEEE Infocom98

[17] J. Nonnenmacher, M. Lacher, M. Jung, E. W. Biersack and Georg Carle, *How bad is reliable multicast without local recovery?*, To appear in Proc. of IEEE INFOCOM'98, San Francisco, CA, USA, 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, November 1996.

[20] 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.

[21] 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.

[22] Jean Bolot, Private Communication, 1998.

Appendix

A Real-Time delivery requirements

Lemma 1 *Given a packet loss probability of $p < 1$, then for any $0 < \epsilon < 1$, it is possible to send a number of packets n such that at least k of n packets are received with probability greater than $1 - \epsilon$.*

Proof: Equations 2 and 5 through 6 present recursive solutions for computing $\mathcal{D}(n,k)$. These functions are clearly monotonically increasing with increasing n, and since they represent probability functions, they must be bounded by 1. It must be shown that the function does not converge to a value less than 1 with increasing n. We prove this by contradiction. Assume it does converge to a value of less than 1. Then because this is a Cauchy sequence, there is some $\epsilon > 0$ where $lim_{n\to\infty}\mathcal{D}(n,k) = 1-\epsilon$. Since this is the limit of a monotonically non-decreasing sequence, for any δ where where $\epsilon < \delta < 1$, there is an n where $\mathcal{D}(n,k) > 1-\delta$. If we choose δ such that $\delta = \epsilon^{3/4} > \epsilon$, and let n_δ be a satisfactory value such that $\mathcal{D}(n_\delta,k) > 1-\delta$. Then $\mathcal{D}(2n_\delta,k)$, which is the probability of getting at least k out of $2n_\delta$ packets, is greater than the probability of getting k packets in the the first set of n_δ packets, or getting k packets in the second set of n_δ packets. This can be written mathematically as $\mathcal{D}(2n_\delta,k) > \mathcal{D}(n_\delta,k) + \mathcal{D}(n_\delta,k) - \mathcal{D}(n_\delta,k)^2 = 2 - 2\delta - (1-\delta)^2 = 1-\delta^2 = 1-\epsilon^{3/2} > 1-\epsilon$. Thus, $1-\epsilon$ cannot be the limit of this monotonically non-decreasing sequence. Since the limit cannot be less than 1, it must be equal to 1. ∎

A.1 A bound for a block good put rate

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 2 *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:

$$\begin{aligned} P(C \wedge (N=n)) &= \sum_{l=0}^{k} P(C \wedge (L=l) \wedge (N=n)) \\ &= P(C \wedge (L=0) \mid (N=n))P(N=n) + \\ &\quad \sum_{l=1}^{k} P(C \mid (N=n) \wedge (L=l))P((N=n) \wedge (L=l)). \end{aligned} \tag{3}$$

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:

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

$$
\begin{aligned}
&= \mathcal{D}(n,k)\,P(N=n) + \frac{\mathcal{P}-\mathcal{D}(n,k)}{1-\mathcal{D}(n,k)} \times P(N=n)(1-P(L=0 \mid N=n)) \\
&= \mathcal{D}(n,k)\,P(N=n) + \frac{\mathcal{P}-\mathcal{D}(n,k)}{1-\mathcal{D}(n,k)}\,P(N=n)(1-\mathcal{D}(n,k)) \\
&= \mathcal{P}P(N=n). \qquad (4)
\end{aligned}
$$

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

A.2 Two-state model calculation of loss

Since today's routers currently perform drop-tail routing, losses in the network occur in a bursty fashion. The previous results assume that losses are not temporally correlated, and will produce inaccurate calculations for networks with burst loss. It has been shown that burst losses in the Internet can be accurately modeled using a 2-state loss model [22]. If a receiver is able to formulate its end-to-end loss characteristics using such a model, then the following should be used to compute the number of packets in the last round.

The loss model used by the receiver to represent end-to-end bursty loss contains states R and L. State R represents receipt of a packet, and state L represents loss. If in state R, the probability of getting a packet and staying in state R is α, and moving to state L is $1-\alpha$. If in state L, the probability of losing a packet and staying in state L is β, and getting a packet and moving to state R is $1-\beta$.

Let $\mathcal{D}(m,n,L)$ be the probability of sending m packets, and receiving at least n and winding up in state L, and $\mathcal{D}(m,n,R)$ be the similar probability of receipt, but that one winds up in state n. It follows that:

$$
\begin{aligned}
\mathcal{D}(m,n,L) &= \mathcal{D}(m-1,n,L)\beta + \mathcal{D}(m-1,n,R)(1-\alpha) \\
\mathcal{D}(m,n,R) &= \mathcal{D}(m-1,n-1,L)(1-\beta) + \mathcal{D}(m-1,n-1,R)\alpha \qquad (5) \\
\mathcal{D}(m,n) &= \mathcal{D}(m,n,L) + \mathcal{D}(m,n,R) \qquad (6)
\end{aligned}
$$

The recursive equation can make use of base conditions:

$$
\begin{aligned}
\mathcal{D}(n,n,L) &= 0 \qquad (7) \\
\mathcal{D}(n,n,R) &= p_\alpha \alpha^n + p_\beta(1-\beta)\alpha^{n-1} \qquad (8) \\
\mathcal{D}(n,0,L) &= p_\alpha(1-\alpha)\beta^{n-1} + p_\beta\beta^n \qquad (9) \\
\mathcal{D}(n,0,R) &= 0 \qquad (10)
\end{aligned}
$$

where $n > 0$, p_α, is the probability of the system initially residing in state R, and p_β is the probability that it initially resides in state L. Because the gaps between rounds tends to be large, it is probably most appropriate to use a steady state distribution for these values. Namely, $p_\alpha = \alpha/(\alpha+\beta)$, and $p_\beta = 1 - p_\alpha$.

Conditional and unconditional bounds can be computed using the same methods used to compute the bounds in the single state models by simply replacing the single state computation of $\mathcal{D}$ with the 2-state version.

B Random Tree Topology Generation Algorithm

A random tree topology is generated as follows:

1. Put all receivers in a bin A
2. While there are still receivers in bin A
3. pick a random number n between 1 and $LOCAL\_BRANCH\_MAX$
4. choose n receivers from bin A (or all receivers if there are less than n) and remove them from the bin
5. Construct a new node, attach the removed receivers to the node, put the node in bin B

6. end while
7. While there are still nodes in bin B
8. pick a random number n between 1 and $BACKBONE\_BRANCH\_MAX$
9. choose n nodes from bin B (or all the nodes if there are less than n and remove them from the bin)
10. Construct a new node, attach the removed nodes to the new node, and put the new node in bin B.
11. end while
12. Make the last node constructed the sender.

C Computing The Final Timeout before a Hard Deadline

k The block size

τ The rate at which the sender can inject packets into the network (includes encoding time).

b The number of data packets that are lost, which equals the number of data packets that must be obtained through decoding.

$t(b,k)$ The time it takes to decode and recover b packets for a block size of size k. Here, we assume the use of Reed-Solomon encoding, where the decoding time is linear with the increase in block size per packet that requires decoding. Thus, the decode time can be written as bkt.

$\mathcal{T}$ The time of the earliest deadline for any data packet in the block that has not yet been received.

r Estimated (upper bound on) round trip time to / from sender. We also assume that the time from the sender to receiver is $r/2$.

p_n Number of additional packets that the receiver is requesting that the sender transmit.

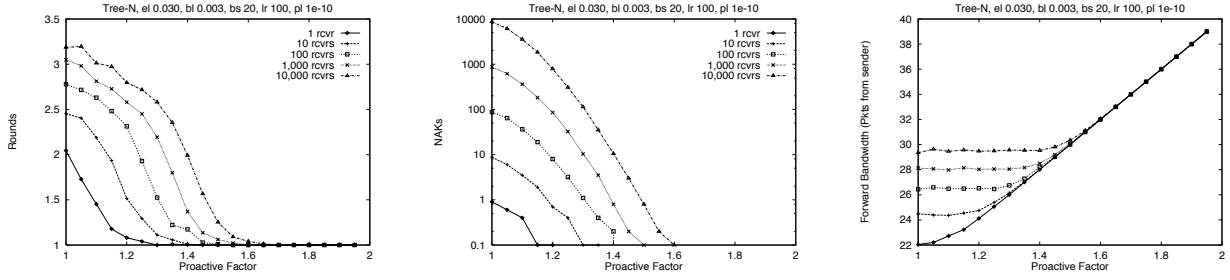
Table 3: How to compute when to send the final NAK

Our approach to allowing a receiver to meet a real-time deadline with a high probability requires that the receiver send some final NAK. The repairs from the sender must be received and decoded by the receiver before the deadline.

The NAK must be issued in advance of the hard deadline so that it allows time for the receiver to build and send the NAK, the sender to build and send repairs, the receiver to receive and decode the necessary repairs, and for all propagation delays that occur between sender and receiver events. Table 3 contains the parameters that we use within the computation. The time by which a NAK should be issued is:

$$\mathcal{T} - (r + p_n * \tau + bkt) \tag{11}$$

D Additional Examination of the Round Based Protocol



(a) (b) (c)

Figure 17: Block size of 20

Section 4 described a round-based protocol that uses proactive FEC. The performance of this protocol was demonstrated in Section 4.2, using a block size of 10 over a tree network topology, where loss at each router was modeled using a Bernoulli loss process. Furthermore, receivers' NAKs always requested exactly the number of packets needed by the receiver (unless meeting a hard real-time guarantee). We now examine how changing various aspects of the model affects the performance of the protocol by considering a set of alternative models. We consider how these variations affect the expected number of rounds, implosion factor,

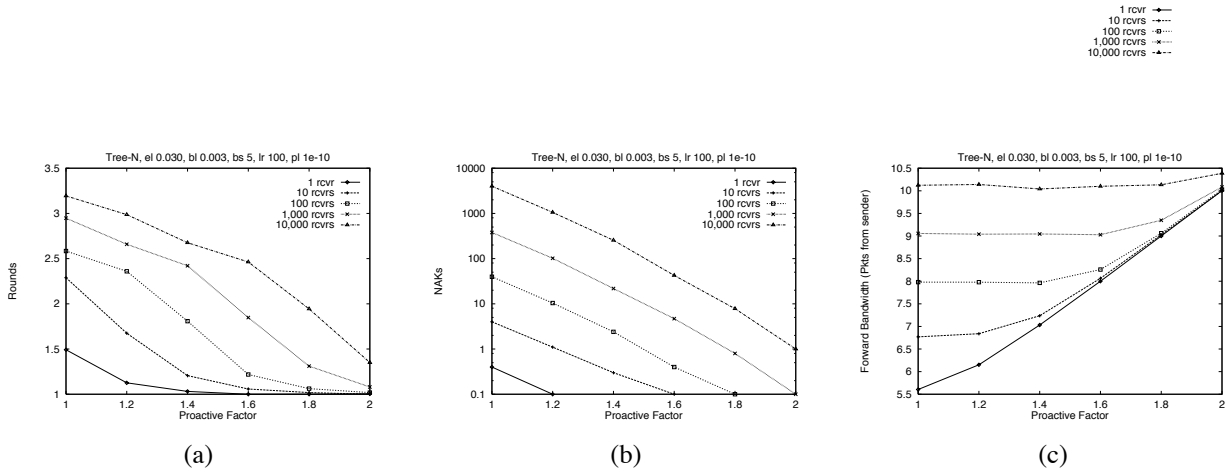


Figure 18: Block size of 5

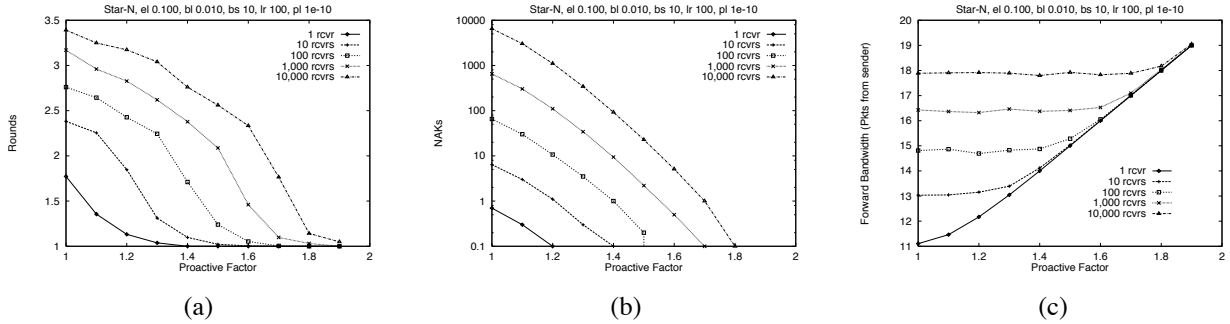


Figure 19: Star topology.

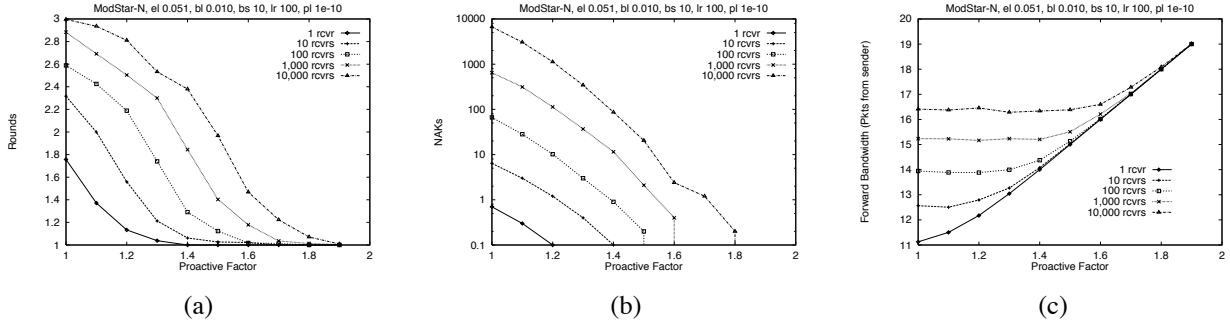


Figure 20: Modified star topology.

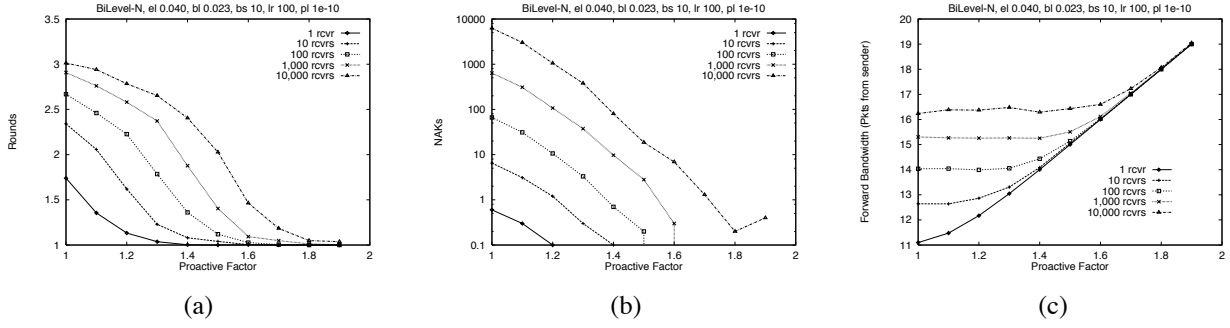
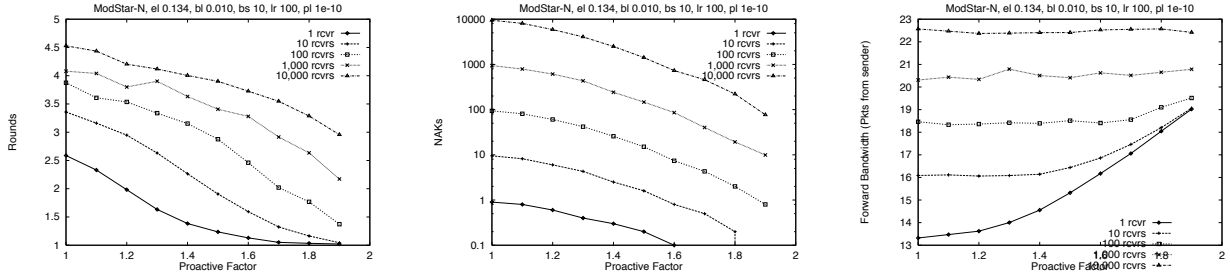


Figure 21: A two-level topology

and forward bandwidth to perform a reliable delivery of a block of data. Each alternative model is identical to the model used in Section 4.2, except for those aspects that are specified as being different.

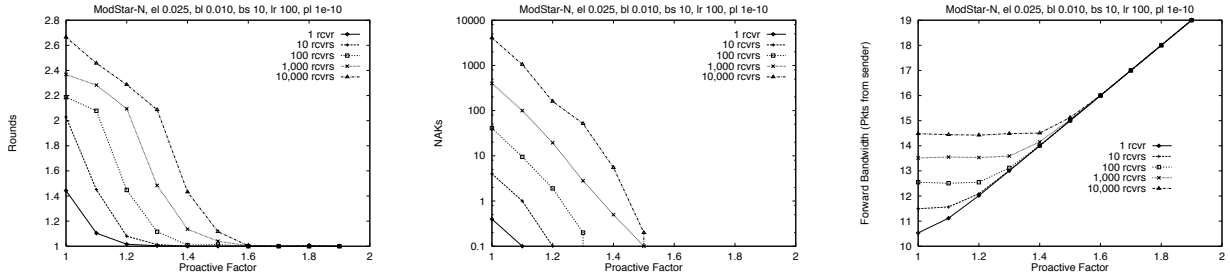
Figures 17 and 18 examine performance when the model is altered to use block sizes of 20 and 5 respectively. We see that proactive factor has a larger impact on the expected number of rounds, implosion factor, and forward bandwidth when the block size is large.

We next consider the impact that topology has on the performance of the protocol. Figure 19 considers a star topology, where there is no spatially shared loss among receivers. Figure 20 considers a modified star topology where the sender connects to a single central router which then connects to the receivers via a star topology. Loss rates are identical on the links from sender to the central router and on the link from the central router to the receivers. Figure 21 considers a bi-level topology similar to that proposed in [16], except that we do not attach repair servers to the network. The sender is connected to a single, central node, which then connects to intermediate nodes. The network contains ten intermediate nodes, and each intermediate node connects to either $\lfloor R/10 \rfloor$ or $\lceil R/10 \rceil$ receivers. The loss rate on the end-links (connecting to sender or receiver) is 0.04. The loss on the backbone link (between central node and an intermediate node) is approximately .023. The loss rates were chosen so that the end-to-end loss rate from the sender to the receiver in each topology was .1. We observe that adding proactivity becomes slightly less effective over networks with more spatial correlation in loss. This is intuitive, since an increase in spatial correlation should have the same impact as reducing the size of the multicast group on a network with no spatial correlation.



(a) (b) (c)

Figure 22: Modified star topology with an end to end loss rate of .25



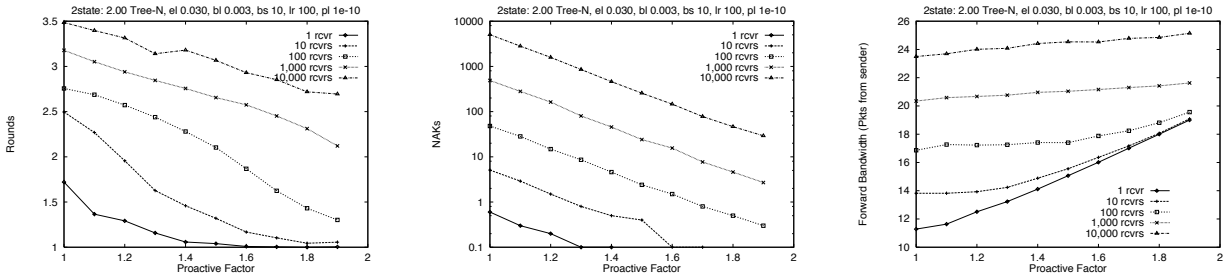
(a) (b) (c)

Figure 23: Modified star topology with an end to end loss rate of .05

Figure 22 examines protocol performance over a modified star topology as in Figure 20, except that end-to-end loss rates to receivers are .25 instead of .1. Figure 23 shows results for the same topology with end-to-end loss rates of .05. We see that the amount of proactivity that should be used increases with an increasing loss rate.

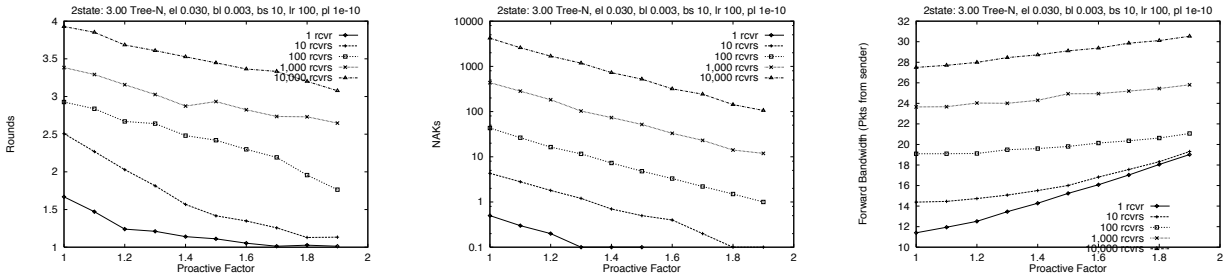
Figures 24 and 25 consider the impact of bursty loss. Each router uses a two-state loss model. The router enters a state of the process with a probability equal to the stationary probability for that state at the start of each round (i.e. there is no temporal dependence across rounds). The probability of loss at each router is chosen in a manner similar to what is described in Section 4.2. The parameters of the loss model can be chosen so that the expected loss burst length is larger than the expected loss burst for a single state model with an identical loss rate. For the plots in Figure 24, the length of the burst is double what it is for the single state model, and is tripled in Figure 25. Bursty loss has an overall negative impact on protocols that use forward error correction. It follows that it would also negatively impact the effectiveness of the proactivity factor. We see that adding proactivity becomes less effective at reducing the number of expected rounds, as well as the number of NAKs. Furthermore, there is always a cost to increasing in terms of bandwidth for increasing the proactivity factor. However, this increase is a small percentage of the overall bandwidth that is used to deliver the data.

Finally, we examine the impact of having each receiver request an additional repair in its NAK. For instance, a receiver that needs 3 additional repairs at the end of a round would request 4 repairs. This allows the receiver to tolerate the loss of a single



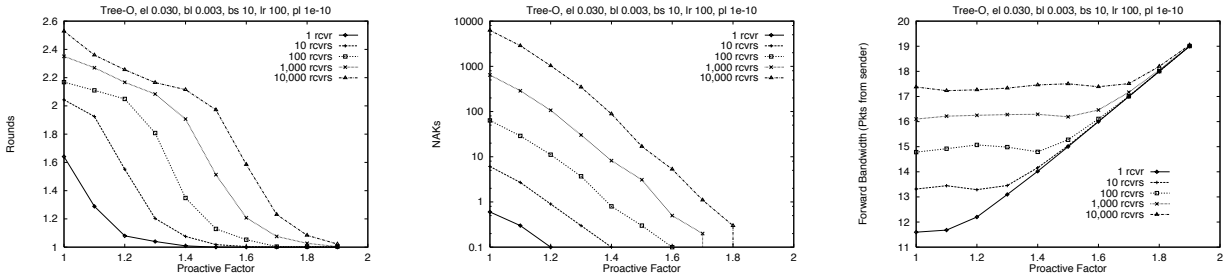
(a) (b) (c)

Figure 24: Bursty loss at each router: The expected burst length is two times the length of the expected burst in a temporally independent loss model



(a) (b) (c)

Figure 25: Bursty loss at each router: The expected burst length is three times the length of the expected burst in a temporally independent loss model



(a) (b) (c)

Figure 26: Receivers request an additional repair in each NAK

repair during each round. We see that such an approach causes a significant reduction in the number of expected rounds. It has a small impact on the number of NAKs, since the majority of NAKs occur at the end of the initial round, before the increase in the NAK has any impact. Finally, there is an increase in the expected bandwidth consumed. This increase is approximately equal to an additional repair packet. This is due to the fact that the expected number of packets to allow all receivers to decode the block stays the same, but most of the time, the request for an additional packet results in an additional, unnecessary repair transmission.

E Unicasting NAKs

Results reported in section 5 assume that all NAKs sent by receivers are multicast. We now consider the affect of having all receivers unicast their NAKs while leaving all other aspects of the protocol unchanged. This is the only modification that is made to the protocol. Figure 27 plots the performance of the unicast NAK model as Figure 12 did for the multicast NAK model. We observe negligible effects on performance, which indicates that there is little benefit to having receivers perform NAK suppression in such protocols.

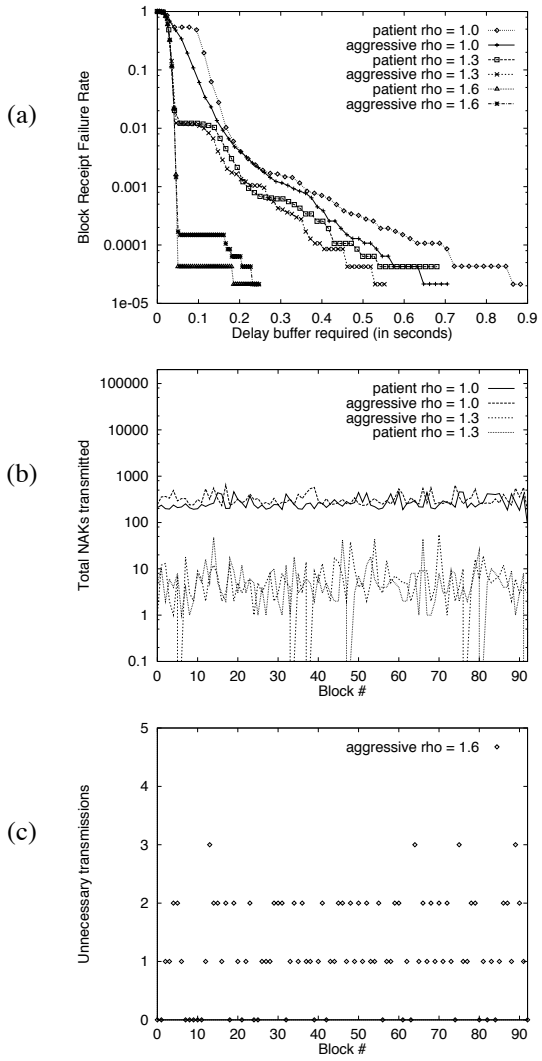


Figure 27: Examination of 500 aggressive and patient receivers with sender proactivity factors of 1.0, 1.3, and 1.6 who *unicast all NAKs*.

F A mathematical analysis

We now consider a mathematical analysis of the round-based protocol. In Section F.1, we consider the case where receivers use no proactivity in their NAKs and give equations for the expected number of packets transmitted and NAKs sent by receivers in the initial round. In Section F.2, we perform an analysis of a single receiver where the receiver can add proactivity to its NAK, and extend these results to the multiple receiver case in Section F.3. All analyses are performed over a star topology using a Bernoulli loss process (there is no spatial or temporal correlation among loss), and loss of a packet occurs with probability p for each receiver. Feedback from receivers (NAKs) are not lost, We use k to represent the block size and r to represent the number of receivers in the

multicast group.

We number the rounds starting with round 0, which is the round where the sender makes the initial transmission of the block and the receivers send their initial NAKs. In each round, $n > 0$, the sender reacts to the feedback from what we define as *the lossiest receiver for round n*. This is the receiver whose NAK requests the maximal number of packets in the previous round. We also define the function $\rho(n, i)$ for $n > 0$ to be the number of packets requested by the lossiest receiver, given that the receiver needs i packets to complete the block at the end of round $n-1$. For $n = 0$, $\rho(0, k) = \lfloor \rho k \rceil$, the number of packets sent by the sender in the initial round. We refer to this function $\rho()$ as the policy, and assume that all receivers use the same policy.

F.1 No Receiver Proactivity

Let L_R be the number of repair packets that are sent after the initial round to satisfy a single receiver, R. Let L be the number of repairs sent by the sender after the initial round to transmit a single block to all receivers. If the sender uses a proactive factor of ρ and receivers do not perform any proactivity, then $\rho(n, i) = i$ for any i and any $n > 0$. Using the analysis in [6], we can compute $E[L]$ as follows:

$$P(L_R = 0) = \sum_{j=0}^{\lfloor (\rho-1)k \rceil} \binom{\lfloor \rho k \rceil}{j} p^j (1-p)^{\lfloor \rho k \rceil - j}, \tag{12}$$

$$P(L_R = m) = \binom{\lfloor \rho k \rceil + m - 1}{k-1} p^{m + \lfloor (\rho-1)k \rceil} (1-p)^k, m > 0 \tag{13}$$

$$P(L_R \leq m) = \sum_{i=0}^{m} P(L_R = i), m \geq 0 \tag{14}$$

$$P(L \leq m) = [P(L_R \leq m)]^r \tag{15}$$

$$E[L] = \sum_{m=0}^{\infty} (1 - P(L \leq m)). \tag{16}$$

It follows that the expected number of packet transmissions is $\lfloor \rho k \rceil + E[L]$.

To compute the number of NAKs in the initial round, we define N_R to be a random variable which equals 1 when receiver R sends a NAK, and 0 when it does not. Let N be the random variable that equals the total number of NAKs transmitted by all receivers in the first round.

$$P(N_R = 1) = 1 - P(L_R = 0) \tag{17}$$

$$E[N_R] = 1 - P(L_R = 0) \tag{18}$$

$$E[N] = rE[N_R] = r(1 - P(L_R = 0)) \tag{19}$$

$$= r \sum_{j=0}^{\lfloor (\rho-1)k \rceil} \binom{\lfloor \rho k \rceil}{j} p^j (1-p)^{\lfloor \rho k \rceil - j} \tag{20}$$

Computing the total number of NAKs for a block (i.e. not just limited to those in the initial round) is dependent on the number of rounds it takes to deliver the entire block to all receivers. This calculation is formulated in the following sections.

F.2 Receivers with proactivity: Single Receiver Case

We now consider a session with a single receiver, where the receiver proactively requests repairs in its NAK. Define $\mathcal{P}_{k,\rho()}(m, n)$ to be the probability that after n rounds using policy $\rho()$, the receiver still needs m packets for $0 < m \leq k$. We construct a recursive solution for $\mathcal{P}_{k,\rho()}(m, n)$ which recurses over m and n:

$$\mathcal{P}_{k,\rho()}(m,n) = \left\{ \begin{array}{ll} \sum_{i=0}^{k-m} \mathcal{P}_{k,\rho()}(m+i, n-1) \binom{\rho(n,m+i)}{i} (1-p)^i p^{\rho(n,m+i)-i} & m > 0, n > 0. \\ \binom{\rho(0,k)}{k-m} (1-p)^{k-m} p^{\rho(0,k)-k+m} & m > 0, n = 0. \end{array} \right\} \tag{21}$$

i iterates over the number of packets that the receiver receives during the transmission in round $n-1$. Equation 21 gives a probability distribution for when $m > 0$: the receiver will still need further transmissions after round n. We define $\mathcal{Q}_{k,\rho()}(m, n)$

to be the probability that given the blocksize of k and policy $\rho()$, a receiver receives all k packets in round n, and in this round, the kth packet received is followed by m subsequent packets from the sender which may or may not be lost en route to the receiver. Then

$$\mathcal{Q}_{k,\rho()}(m,n) = \left\{ \begin{array}{ll} \sum_{i=1}^{k} \mathcal{P}_{k,\rho()}(i,n-1)\binom{\rho(n,i)-m-1}{i-1}(1-p)^i p^{\rho(n,i)-m-i} & m \geq 0, n > 0. \\ \binom{\rho(0,k)-m-1}{k-1}(1-p)^k p^{\rho(0,k)-m-k} & m \geq 0, n = 0. \end{array} \right\} \quad (22)$$

In equation 22, i iterates over the number of packets that the receiver needs to decode the block at the end of round $n-1$.

We define $\mathcal{P}_{k,\rho()}(0,n)$ to be the probability that round n is the last round needed by any receiver to obtain repairs. Then

$$\mathcal{P}_{k,\rho()}(0,n) = \sum_{i=0}^{\rho(n-1,k)} \mathcal{Q}_{k,\rho()}(i,n). \quad (23)$$

F.2.1 Expected number of packets sent

We now compute the expected number of packets that are sent over n rounds given a block size of k and a particular policy $\rho()$. Define the variable X_i to be the number of packets sent in the ith round. Thus, the expected value we wish to compute can be written as:

$$E[\sum_{i=0}^{n} X_i] = \sum_{i=0}^{n} E[X_i]. \quad (24)$$

It therefore suffices to determine each $E[X_i]$. We know that for $i = 0, X_i$ is always $\rho(0,k)$, so it is always the case that $E[X_0] = \rho(0,k)$. We now calculate $E[X_i]$ for $0 < i \leq n$. If Y_i is defined to be the number of packets that must still be received by the receiver after round i, then $X_i = \rho(i-1, Y_{i-1})$. Thus, $P(Y_{i-1} = m) = P(X_i = \rho(i-1,m)) = \mathcal{P}_{k,\rho()}(m, i-1)$. Since X_i must be a value from the set $\cup_{0<j\leq k}\{\rho(i-1,j)\}$, we have:

$$\begin{aligned} E(X_i) &= \sum_{j=1}^{k} \rho(i-1,j) \cdot P(X_i = \rho(i-1,j)) \\ &= \sum_{j=1}^{k} \rho(i-1,j) \cdot P(Y_{i-1} = j) \\ &= \sum_{j=1}^{k} \rho(i-1,j) \cdot \mathcal{P}_{k,\rho()}(j, i-1). \end{aligned} \quad (25)$$

Note that we use the fact that $\rho(i,0) = 0$ to allow the sum to start at $j = 1$.

F.3 Multiple Receivers

We now consider a multicast group that contains multiple receivers that proactively request repairs.

We make two assumptions about the proactive policy used by receivers:

- All receivers use the same policy, $\rho()$.
- $\forall i \geq 0, j > 0, \rho(i,j) \geq \rho(i-1,j)$. In other words, for each round, $\rho()$ is a monotonically non-decreasing function of the number of packets needed.

Define $\mathcal{P}_{r,k,\rho()}(m,n)$ to be the probability that given a blocksize of k, a policy $\rho()$, and r receivers that after n rounds, the lossiest receiver for round n needs m more packets. We are unable to compute this value recursively as was done for the single-receiver case. The difficulty is due to the fact that the lossiest receiver for round n might differ from the lossiest receiver for round $n-1$.

To enable a recursive computation, we define $\mathcal{P}_{r,k,\rho()}(m,n,l)$ to equal to the probability as stated for $\mathcal{P}_{r,k,\rho()}(m,n)$ with the added condition that after the nth round, a total of l packets have been sent. Thus,

$$\mathcal{P}_{r,k,\rho()}(m,n) = \sum_{l=n+1}^{S} \mathcal{P}_{r,k,\rho()}(m,n,l). \tag{26}$$

S must be chosen large enough such that $\mathcal{P}_{r,k,\rho()}(m,n,S+i) = 0$ for all $i > 0$. Since $\rho((,n),i)$ is a monotonically non-decreasing function, the largest total number of packets that can be sent by round n is $\sum_{i=0}^{n} \rho(i,k)$. It is therefore sufficient to set S to this value.

We define 2 random variables:

- $\mathcal{M}_{r,l}$ is the maximum number of packets needed by any receiver to complete the block of k packets after l packets have been sent.
- $\mathcal{R}_{r,n}$ is a random variable equal to the total number of packets that are sent by the end of the nth round.

For example, $\mathcal{M}_{50,15} = 3$ implies that there are 50 receivers, and after 15 packets have been sent, there is at least one receiver that still needs 3 packets to decode the FEC block, and no receiver needs more than 3 packets. For another example, if there are 50 receivers, and 10 packets are sent the zeroth round, 8 packets the first round, and 7 packets the second round, then $\mathcal{R}_{50,0} = 10, \mathcal{R}_{50,1} = 18$, and $\mathcal{R}_{50,2} = 25$.

Then the following holds:

$$\begin{aligned}
&\mathcal{P}_{r,k,\rho()}(m,n,l) \\
&= P((\mathcal{M}_{r,l} = m) \wedge (\mathcal{R}_{r,n} = l)) \\
&= \sum_{i=0}^{k-m} [P([(\mathcal{M}_{r,l} = m) \wedge (\mathcal{R}_{r,n} = l)] \mid [(\mathcal{M}_{r,l-\rho(n,m+i)} = m+i) \wedge (\mathcal{R}_{r,n-1} = l - \rho(n,m+i))]) \times \\
&\quad P([(\mathcal{M}_{r,l-\rho(n,m+i)} = m+i)] \wedge [(\mathcal{R}_{r,n-1} = l - \rho(n,m+i))])] \\
&= \sum_{i=0}^{k-m} [P([(\mathcal{M}_{r,l} = m) \wedge (\mathcal{R}_{r,n} = l)] \mid [(\mathcal{M}_{r,l-\rho(n,m+i)} = m+i) \wedge (\mathcal{R}_{r,n-1} = l - \rho(n,m+i))]) \times \\
&\quad \mathcal{P}_{r,k,\rho()}(m+i, n-1, l-\rho(n,m+i))] \qquad (27) \\
&= \sum_{i=0}^{k-m} [P(\mathcal{M}_{r,l} = m \mid \mathcal{M}_{r,l-\rho(n,m+i)} = m+i) \times \mathcal{P}_{r,k,\rho()}(m+i, n-1, l-\rho(n,m+i))]. \qquad (28)
\end{aligned}$$

Here, i iterates over the possible number of packets received on the nth round.

The equality of equations (27) and (28) is given by the following lemma:

Lemma 3

$$P([(\mathcal{M}_{r,l} = m) \wedge (\mathcal{R}_{r,n} = l)] \mid [(\mathcal{M}_{r,l-\rho(n,m+i)} = m+i) \wedge (\mathcal{R}_{r,n-1} = l - \rho(n,m+i))]) = P(\mathcal{M}_{r,l} = m \mid \mathcal{M}_{r,l-\rho(n,m+i)} = m+i).$$

Proof: It is always the case that $\mathcal{R}_{r,n} = \mathcal{R}_{r,n-1} + \mathcal{M}_{r,\mathcal{R}_{r,n-1}}$. Thus, if $\mathcal{M}_{r,l-\rho(n,m+i)} = m+i$ and $\mathcal{R}_{r,n-1} = l - \rho(n,m+i)$, then it follows that $\mathcal{R}_{r,n} = l$. Thus,

$$P([(\mathcal{M}_{r,l} = m) \wedge (\mathcal{R}_{r,n} = l)] \mid [(\mathcal{M}_{r,l-\rho(n,m+i)} = m+i) \wedge (\mathcal{R}_{r,n-1} = l - \rho(n,m+i))]) = P(\mathcal{M}_{r,l} = m \mid [(\mathcal{M}_{r,l-\rho(n,m+i)} = m+i) \wedge (\mathcal{R}_{r,n-1} = l - \rho(n,m+i))]).$$

Next, we note that $\mathcal{M}_{r,l}$ is independent of the points at which rounds end. It simply depends on the packet loss up to the lth packet. Thus,

$$P(\mathcal{M}_{r,l} = m \mid ((\mathcal{M}_{r,l-\rho(n,m+i)} = m+i) \wedge (\mathcal{R}_{r,n-1} = l - \rho(n,m+i)))) = P(\mathcal{M}_{r,l} = m \mid \mathcal{M}_{r,l-\rho(n,m+i)} = m+i) \tag{29}$$

∎

This gives a recursive solution for $\mathcal{P}_{r,k,\rho()}(m,n,l)$ given that we can solve for $P(\mathcal{M}_{r,l} = m \mid \mathcal{M}_{r,l-\rho(n,m+i)} = m+i)$. We show how to compute $P(\mathcal{M}_{r,s} = i \mid \mathcal{M}_{r,t} = j)$ where $s > t, i \leq j$ in Section F.5.

For the case where $n = 0$, we have that:

$$\mathcal{P}_{r,k,\rho()}(m,0,l) = P(\mathcal{M}_{r,l} = m \mid \mathcal{M}_{r,l-\rho(0,k)} = k) \tag{30}$$

We also extend the definition of $\mathcal{M}_{r,l} = i$ for $i \leq 0$ to mean that after l packets, the receiver that has received the fewest packets has received $k+i$ (essentially, the receiver could be thought of as needing $-i$ packets). Similar to lemma 3, we have for $m \leq 0$:

Lemma 4

$$P([(\mathcal{M}_{r,l} = m) \wedge (\mathcal{R}_{r,n} = l)] \mid [(\mathcal{M}_{r,l-\rho(n,i)} = i) \wedge (\mathcal{R}_{r,n-1} = l - \rho(n,i))]) = P(\mathcal{M}_{r,l} = m \mid \mathcal{M}_{r,l-\rho(n,i)} = i)$$

Proof: Similar to that of lemma 3. ∎

F.3.1 Probability that all packets are received round n

Let $\mathcal{Q}_{r,k,\rho()}(m,n)$ extend the definition of $\mathcal{Q}_{k,\rho()}(m,n)$ to that of multiple receivers in the same way that $\mathcal{P}_{r,k,\rho()}(m,n)$ extends the definition of $\mathcal{P}_{k,\rho()}(m,n)$. Also define $\mathcal{Q}_{r,k,\rho()}(m,n,l) = P((\mathcal{M}_{r,l} = -m) \wedge (\mathcal{R}_{r,n} = l))$ (this is similar to how $\mathcal{P}_{r,k,\rho()}(m,n,l)$ is defined). Then:

$$\mathcal{Q}_{r,k,\rho()}(m,n,l) = \sum_{i=1}^{k} P(\mathcal{M}_{r,l} = m \mid \mathcal{M}_{r,l-\rho(n,i)} = i) \cdot \mathcal{P}_{r,k,\rho()}(i,n-1,l-\rho(n,i)) \quad \text{for } n > 0 \tag{31}$$

$$\mathcal{Q}_{r,k,\rho()}(m,0,l) = P(\mathcal{M}_{r,l} = m \mid \mathcal{M}_{r,l-\rho(0,k)} = k) \tag{32}$$

$$\mathcal{P}_{r,k,\rho()}(0,n,l) = \sum_{m=0}^{\rho(n,k)} \mathcal{Q}_{r,k,\rho()}(m,n,l) \tag{33}$$

F.3.2 Expected # of packets sent

Equation 24 holds for the multiple-receiver case as well. To compute the expected number of packets sent, we modify equation 25 to:

$$E(X_i) = \sum_{j=1}^{k} \rho(i,j) \cdot \mathcal{P}_{r,k,\rho()}(j,i-1). \tag{34}$$

F.3.3 Expected # of rounds

Define $\mathcal{P}_{r,k,\rho()}(0,n)$ to be the probability that the last receiver to receive k packets does so on round n. Then

$$\mathcal{P}_{r,k,\rho()}(0,n) = \sum_{l=0}^{\infty} \mathcal{P}_{r,k,\rho()}(0,n,l) \tag{35}$$

Again, we point out that the bounds for l can be restricted to the finite interval $[\rho(0,k) + \sum_{i=1}^{n} \rho(i,1), \sum_{i=0}^{n} \rho(i,k)]$. The expected number of rounds can be computed as:

$$E(N) = \sum_{n=0}^{\infty} n\mathcal{P}_{r,k,\rho()}(0,n) \tag{36}$$

F.4 NAKs sent

We assume that a receiver will send a NAK after any round for which it has not received the entire block. The NAK includes the number of packets that the receiver still needs.

Let $\mathcal{N}_{r,n}$ be a random variable that equals the number of NAKs that are sent out after round n given that there are r receivers participating in the multicast session. Then

$$\begin{aligned} & P(\mathcal{N}_{r,n} = i) \\ = & \sum_{l=0}^{\infty} P((\mathcal{N}_{r,n} = i) \wedge (\mathcal{R}_{r,n} = l)) \\ = & \sum_{l=0}^{\infty} P(\mathcal{N}_{r,n} = i \mid \mathcal{R}_{r,n} = l) \cdot P(\mathcal{R}_{r,n} = l). \end{aligned} \tag{37}$$

The two probabilities in equation (37) can be solved separately as follows:

$$P(\mathcal{R}_{r,n} = l) = \sum_{m=0}^{k} \mathcal{P}_{r,k,\rho()}(m,n,l) \tag{38}$$

$$P(\mathcal{N}_{r,n} = i \mid \mathcal{R}_{r,n} = l) = \binom{r}{i} {q_l}^i (1-q_l)^{r-i} \tag{39}$$

$$q_l = \sum_{j=0}^{k-1} \binom{l}{j} p^{l-j}(1-p)^j \tag{40}$$

Here q_l is the probability that out of l packets, fewer than k have been received by a single receiver.

The expected number of NAKs sent in round n is:

$$\begin{aligned} E(\mathcal{N}_{r,n}) &= \sum_{i=0}^{r} i P(\mathcal{N}_{r,n} = i) \\ &= \sum_{i=0}^{r} \sum_{l=0}^{\infty} i \binom{r}{i} {q_l}^i (1-q_l)^{r-i} \cdot P(\mathcal{R}_{r,n} = l) \\ &= \sum_{l=0}^{\infty} \sum_{i=0}^{r} i \binom{r}{i} {q_l}^i (1-q_l)^{r-i} \cdot P(\mathcal{R}_{r,n} = l) \\ &= \sum_{l=0}^{\infty} r q_l P(\mathcal{R}_{r,n}) = l. \end{aligned} \tag{41}$$

As in equation (26), it is possible to restrict to a finite number the number of values that l must iterate over. To compute the expected number of NAKs sent up to and including the round n, we can simply sum over the expected number of NAKs computed in each round.

F.5 Computations

Here, we present a method for computing $P(\mathcal{M}_{r,s} = i \mid \mathcal{M}_{r,t} = j)$. Given $s > t, i \leq j$,

$$P(\mathcal{M}_{r,s} = i \mid \mathcal{M}_{r,t} = j) = P((\mathcal{M}_{r,s} = i) \wedge (\mathcal{M}_{r,t} = j)) \,/\, P(\mathcal{M}_{r,t} = j). \tag{42}$$

Thus, we wish to compute:

- $P((\mathcal{M}_{r,s} = i) \wedge (\mathcal{M}_{r,t} = j))$, where $s > t, i \leq j$.
- $P(\mathcal{M}_{r,t} = j)$.

$$
\begin{aligned}
&P((\mathcal{M}_{r,s}=i)\ \wedge\ (\mathcal{M}_{r,t}=j))\\
&\quad = \quad P((\mathcal{M}_{r,s}=i)\ \wedge\ (\mathcal{M}_{r,t}\leq j)) - P((\mathcal{M}_{r,s}=i)\ \wedge\ (\mathcal{M}_{r,t}\leq j-1))\\
&\quad = \quad [P((\mathcal{M}_{r,s}\leq i)\ \wedge\ (\mathcal{M}_{r,t}\leq j)) - P((\mathcal{M}_{r,s}\leq i-1)\ \wedge\ (\mathcal{M}_{r,t}\leq j))] -\\
&\qquad\quad [P((\mathcal{M}_{r,s}\leq i)\ \wedge\ (\mathcal{M}_{r,t}\leq j-1)) - P((\mathcal{M}_{r,s}\leq i-1)\ \wedge\ (\mathcal{M}_{r,t}\leq j-1))] \qquad (43)
\end{aligned}
$$

$$
\begin{aligned}
&P((\mathcal{M}_{r,s}\leq i)\ \wedge\ (\mathcal{M}_{r,t}\leq j))\\
&\quad = \quad P((\mathcal{M}_{1,s}\leq i)\ \wedge\ (\mathcal{M}_{1,t}\leq j))^r\\
&\quad = \quad \left(\sum_{v=k-i}^{s}\sum_{w=k-j}^{min[t,v]}\left[\underbrace{\binom{t}{w}(1-p)^w p^{t-w}}_{P(\mathcal{M}_{1,t}=k-w)}\underbrace{\binom{s-t}{v-w}(1-p)^{v-w}p^{s-t-v+w}}_{P(\mathcal{M}_{1,s}=k-v-w|\mathcal{M}_{1,t}=k-w)}\right]\right)^r\\
&\quad = \quad \left(\sum_{v=k-i}^{s}\sum_{w=k-j}^{min[t,v]}\left[\binom{t}{w}\binom{s-t}{v-w}(1-p)^v p^{s-v}\right]\right)^r. \qquad (44)
\end{aligned}
$$

Above, v iterates over the number of packets received out of s sent packets. w iterates over the number of packets received out of the the first t packets sent.

$P(\mathcal{M}_{r,t}=j)$ is computed in a similar fashion:

$$
P(\mathcal{M}_{r,t}=j) \quad = \quad P(\mathcal{M}_{r,t}\leq j) - P(\mathcal{M}_{r,t}\leq j-1). \qquad (45)
$$

$$
\begin{aligned}
P(\mathcal{M}_{r,t}\leq j) \quad &= \quad P(\mathcal{M}_{1,t}\leq j)^r\\
&= \quad \left(\sum_{v=k-j}^{t}\left[\binom{t}{v}(1-p)^v p^{t-v}\right]\right)^r. \qquad (46)
\end{aligned}
$$