A Buffer Management Scheme for Tree-Based Reliable Multicast Using Infrequent Acknowledgments

Jinsuk Baek Jehan-François Pâris
Department of Computer Science
University of Houston
Houston, TX 77204-3010
{jsbaek, paris}@cs.uh.edu

Abstract

Tree-based reliable multicast protocols provide scalability by distributing error-recovery tasks among several repair nodes. These repair nodes integrate the status information of several receiver nodes and perform local error recovery for these nodes using the data stored in their buffers. We propose a buffer management scheme that uses both positive and negative acknowledgments to manage these buffers in an efficient manner. Under our scheme, receiver nodes send negative acknowledgments to repair nodes to request packet retransmissions. At infrequent intervals, they also send positive acknowledgments to indicate which packets can be safely discarded from the buffer of their repair node. Our scheme reduces delay in error recovery, because the packets requested from the repair nodes are always available in their buffers. It achieves this goal without increasing the server workload because (a) each receiver node only sends infrequent positive acknowledgments and (b) their sending times are randomized among all the receiver nodes. We also show how our scheme can be extended to provide full flow control and eliminate buffer overflows.

I. Introduction

One of the most difficult issues in end-to-end multicasting is that of providing an error-free transmission mechanism. The standard method for providing reliable unicast is positive acknowledgements (ACKs). It requires the receiver to send an ACK for each packet that it has received. The sender keeps track of these ACKs and retransmits all packets that have not been properly acknowledged within a given time window. TCP [13] is a well-known protocol using positive ACKs to provide reliable unicast.

The same approach fails when applied to reliable multicasts because of the ACK implosion [1, 17, 21] it creates. Since each receiver has to acknowledge each packet it has correctly received, the sender's ability to handle these ACKs limits the number of nodes participating in a reliable multicast.

This situation has led to numerous proposals aiming at providing scalable reliable schemes. The IETF Reliable Multicast Transport (RMT) Working Group has standardized three RMT protocols based on these proposals: Asynchronous Layered Coding (ALC) [5], a NAK-based protocol [4], and a tree-based protocol [1, 9, 17].

Among these three protocols, the tree-based protocol is known to provide high scalability as well as reliability. It constructs a logical tree at the transport layer for error recovery. This tree comprises three types of nodes: a sender node, repair nodes, and receiver nodes. The sender node is the root of the logical multicast tree. It controls the overall tree construction and is responsible for retransmission of lost packets within the whole multicast group. Each repair node acts as a local server for a subset of the receiver nodes in the tree. It integrates the status information of its receiver nodes and performs local error recovery for these nodes using the data cached in its buffer. Hence, tree-based protocol achieves scalability by distributing the server retransmission workload among the repair nodes.

There are still two open issues in tree-based protocols. The first is how to construct a logical tree in an efficient manner. One of the authors has recently proposed an efficient hybrid scheme for constructing a well-organized logical tree [1]. His hybrid scheme combines the advantages of previous schemes by constructing a logical tree in a semi-concurrent manner while minimizing the number of control messages.

The second open issue is when to discard packets from the buffers of repair nodes. Discarding packets that might still be needed is unacceptable, because it would force the receiver nodes to contact the sender node whenever they need a retransmission of a discarded packet. Discarding packets too late would result in an inefficient use of the available buffer space on the repair nodes. Schemes addressing this issue can be broadly divided into ACK-based [6, 7, 17] and NAK-based schemes [2-4, 14]. As we will see, both approaches have their own limitations.

In the ACK-based schemes, receiver nodes send an ACK to their repair node every time they have correctly received packet. This allows each repair node to discard from its buffer all packets that have been acknowledged by all receiver nodes. However, the ACK-based approach does not scale up well due to the ACK implosion occurring at the repair nodes. NAK-based schemes solve this ACK implosion problem by shifting the error detection load from the repair to each receiver node. Receiver nodes reply with NAKs whenever they detect a packet loss. However, they provide no efficient mechanism to safely discard packets from the repair node buffers.

We propose an efficient buffer management scheme combining NAKs and infrequent ACKs to overcome these limitations. Under our scheme, receiver nodes send NAKs to repair nodes to request packet retransmissions. At fixed infrequent intervals, they also send ACKs to indicate which packets can be safely discarded from the repair node buffers. We avoid any risk of ACK implosion because each receiver node only sends infrequent positive acknowledgments and their sending times are randomized among all the receiver nodes.

Our proposal has two major advantages over other schemes. First, the amount of feedback from receiver nodes is significantly reduced owing to randomized ACK. This feature provides scalability, since each repair node will be able to handle more receiver nodes. Second, it guarantees fast recovery of transmission errors, since the packets requested from receiver nodes are always available in the buffers of the repair nodes. In addition, the scheme can be extended to provide flow control and eliminate any risk of buffer overflow.

The remainder of this paper is organized as follows. Section II summarizes the existing buffer management schemes for providing buffer refreshment functionality in reliable multicast. Section III introduces our new buffer management scheme. In section IV, we show the performance of the proposed scheme. Finally, section V contains our conclusions.

II. Related Works

This section describes the buffer management protocols of existing reliable multicast protocols. These protocols essentially differ in the strategies they use for deciding which participants buffer packets for retransmission and how long these packets should be retained.

*Scalable Reliable Multicas*t (SRM) [4] is a well-known receiver-initiated multicast protocol that guarantees out-of-order reliable delivery using NAKs from receivers. Whenever a receiver detects a lost packet, it multicasts NAKs to all participants in the multicast session. This allows the nearest receiver to retransmit the packet by multicasting. As a result, the protocol distributes the error recovery load from one sender to all receivers of the multicast session. The sole drawback of the SRM protocol is that all receivers have to keep all packets in their buffer for retransmission. Hence, SRM protocol cannot provide an efficient buffer management mechanism at the transport layer.

The first tree-based reliable multicast protocol was the *Reliable Multicast Transport Protocol* (RMTP) [17]. RMTP provides reliable multicast by constructing a physical tree of the network layer. It allocates a *designated receiver* (DR) in each local region and makes this receiver responsible for error recovery for all the other receivers in that region. To reduce ACK implosion, each receiver periodically unicasts an ACK to its designated receiver instead of sending an ACK for every received packet. This ACK contains the maximum packet number that each receiver has successfully received. Unfortunately, this periodic feedback policy significantly delays error recovery. Hence, RMTP is not suitable for applications that transmit time-sensitive multimedia data. In addition, RMTP stores the whole multicast session data in the secondary memory of the DR for retransmission, which makes it poorly suited for transfers of large amounts of data. Some of these problems were addressed in RMTP-II by the addition of NAKs.

Guo [6] proposed a stability detection algorithm partitioning receivers into groups and having all receivers in a group participate in error recovery. This is achieved by letting receivers periodically exchange history information about the set of messages they have received. Eventually one receiver in the group becomes aware that all the receivers in the group have successfully received the packet and announces this to all the members in the group. Then all members can safely discard the packet from buffer. This feature causes high message traffic overhead because the algorithm requires frequent exchange of messages.

Ozkasap [14] proposed an efficient buffering policy where only a small set of receivers buffer the packet to reduce the amount of total buffer requirement. Receivers that have lost packets use a hash function to select the set of members that have the packet in the buffer and request a retransmission of the packet from one of them. However, their selection method does not consider geographic locations between different receivers. Hence, its scalability is constrained because the latency for error recovery increases with the number of participants.

Randomized Reliable Multicast Protocol (RRMP) [18] is an extended version of *Bimodal Multicast Protocol* (BMP) [2]. BMP uses a simple buffer management policy in which each member buffers packets for a fixed amount of time. RRMP uses instead a two-phase buffering policy: feedback-based short-term buffering and randomized long-term buffering. In the first phase, every member that receives a packet buffers it for a short period of time in order to facilitate retransmission of lost packets in its local region. After that, only small random subset of members in each region continues to buffer the packet. The drawback of this protocol is that it takes a long time for the receiver to search and find the correct repair nodes as the number of participants increase.

Finally, the *Search Party* protocol [3] uses a timer to discard the packet from the buffer: each member in the group simply discards packets after a fixed amount of time. The protocol remains vague on the problem of selecting the proper time interval for discarding packets.

Most of the NAK-based multicast protocols remain equally vague on that issue because the absence of a NAK from a given receiver for a given packet is not a definitive indication that the receiver has received the packet. Yamamoto *et al.* [20, 21] have proposed an interesting flow-control scheme for NAK-based multicast protocols. Their scheme requires the sender to reduce its transmission rate whenever it receives NAKs for too many of its packets. To prevent excessive decreases of the transmission rate, the sender keeps also a log of its past transmission rate. While the scheme was found to be efficient we need to point out that it minimizes occurrences of buffer overflow rather than eliminating them as a sliding window protocol would do.

III. Proposed Buffer Management Scheme

Our scheme applies to tree-based protocols where each repair node provides local recovery in its region on the logical tree. We further assume a receiver-initiated error recovery process and require receiver nodes to send a NAK to their repair node every time they detect a packet loss. In addition, we make the following assumptions:

- There are N receiver nodes for each repair node. This repair node is responsible for re-transmitting the packets requested with NAKs from its N receiver nodes.
- Each receiver node has an independent link loss probability. Let LP_i for $i = 1, 2, \ldots, N$ be the loss probability of receiver node i.
- The sender node has m data packets to transmit to the members of the multicast group.
- Each packet contains its sequence number n_s.

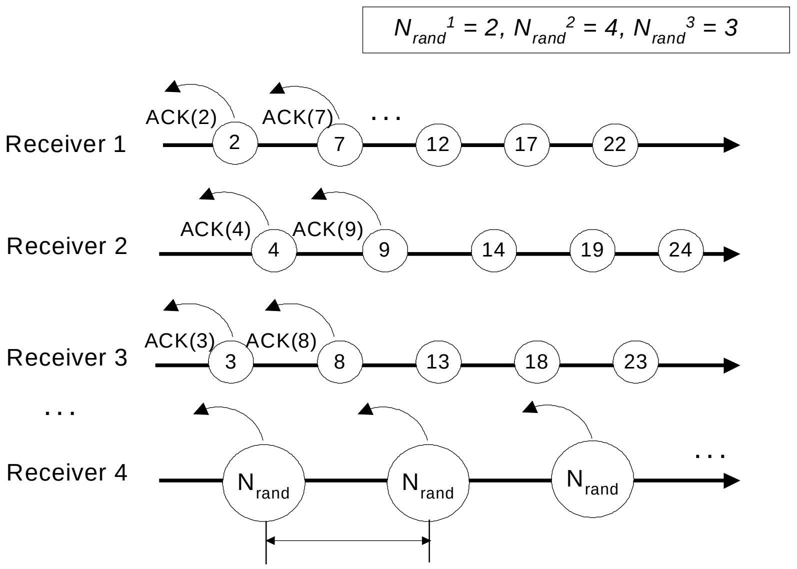

Figure 1. Example of randomized ACK scheme (with N_{max} = 5)

III.1. Basic Scheme

When a receiver node i joins a multicast session, it receives a packet from its repair node specifying the maximum number N_{max} of sent packets the repair node is normally willing to keep in its buffer. The node generates then a random number N_{rand}^i between 1 and N_{max} that identifies the first packet after which it will send an ACK packet to its repair node. This packet will act as an implicit acknowledgment for packets 1 to N_{rand}^i. After that, receiver node i will send a similar ACK packet after each packet RC_i such that

$$RC_i = N_{rand}^i + k\, N_{max} \text{ for } k = 0, 1, 2, 3,\ldots \qquad (1)$$

In addition, the receiver nodes always acknowledge the last packet of the transmission. Since these transmissions start at random offsets

$$N_{rand}^i = \text{random}(1, N_{max}) \text{ for } 1 \le i \le N, \qquad (2)$$

they will be more or less equally distributed over time. Therefore, we eliminate the risk of a sudden avalanche of ACKs sent for the same packet. Figure 1 shows an example of our randomized ACK scheme when N_{max} is equal to 5.

Let L_i be the last ACK packet sent by receiver node i. Assuming that there is no pending packet retransmission, the repair node now can safely discard up to packet D such that

$$D \le \min\{L_i \mid 1 \le i \le N\} \qquad (3)$$

Receiver nodes that leave the multicast session without any notice can disrupt the multicast session for all receiver nodes. The repair node will use a timeout mechanism to detect them and cut them off.

Let us turn now our attention to the way our scheme handles lost packets and corrupted packets. When a receiver node *i* detects such a packet, it sends a NAK to the repair node. This NAK will contain the sequence number of the last packet such that itself and all its predecessors were correctly received by receiver node *i*.

Here, two options are possible. First, the receiver node *i* can continue the same sequence of ACKs. The sole disadvantage of this option is that we will not take advantage of the implicit ACK contained in the NAK. Assume, for instance, that N_{max}=10 and the first message acknowledged by *i* was 5. Node *i* is then supposed to acknowledge packet 5, 15, 25, and so on. Assume now that receiver node *i* did not receive packet number 14. It will send a NAK mentioning that packet number 14 was not correctly received and that the packet number 13 is the last packet such that itself and all its predecessors were correctly received by node *i*. Hence, the next ACK sent by the node will only acknowledge the correct reception of packets 14 and 15.

A better solution is to delay the next ACK by receiver node *i* up to packet 13+10=23. After that, receiver *i* will acknowledge packet 33, 43, and so on and will remain in that schedule until it sends another NAK. This would still ensure that the repair node receives at least one ACK from receiver node *i* every N_{max} packets. This second option has one major drawback. If most of the receivers do not receive a specific packet, they will reschedule all their ACKs for the same packets, thus creating unacceptable peek in the server workload. We decided therefore on another solution relying again on randomization. Assume that node *i* sends a NAK for packet *j* and that NAK includes an ACK for packet *k*. Node *i* will select a new random number N'^{i}_{rand} between 1 and N_{max}. The next packet to be acknowledged by receiver node *i* will then be given by

$$\max(j + N'^{i}_{rand}, L_i + N_{max}), \text{ for } 1 \le i \le N, \tag{4}$$

where L_i is the last ACK sent by receiver node *i*.

The first main advantage of our scheme is that it significantly reduces the number of feedbacks from receiver nodes, required to safely discard the packets from repair node's buffer. This feature increases the whole Internet performance by reducing unnecessary traffic.

A second advantage of our scheme is that it guarantees that the repair node will always have in its buffer all the packets that can be requested by any of its receiver nodes. We achieve this reliability using cumulative ACK mechanism. Our ACK(n) means the receiver has correctly received up to packet n. Assume, for instance, that a receiver node i has not received packet k. It will send a NAK(k) to the repair node. The repair node basically buffers at least up to L_i packet such that $L_i > k$. Hence, the packet k is always available in repair node's buffer and can be retransmitted. In our scheme, the arrival time of ACK(L_i) is always larger than that of NAK(k) because the receiver node i cannot send an ACK(L_i) until it receives up to packet L_i. It means the receiver node i does not send an ACK(L_i) until it receive a packet k. This property can be described as

$$\text{Arrival\_Time}(\text{ACK}(L_i)) > \text{Arrival\_Time}(\text{NAK}(k)), \tag{5}$$

because $L_i > k$. As a result, our scheme reduces the packet error recovery delay because no packet will ever have to be retransmitted either by an upper repair node or the original sender node. Our error recovery delay is dependent on the round-trip time between the repair node and the receiver node requesting the retransmission of the packet.

The sole disadvantage of our scheme is that each repair node will use more buffer space than a repair node receiving ACKs from all its receiver nodes for all packets sent by the sender node. The total buffer space used by all repair nodes will however remain almost constant because repair nodes receiving less ACKs will be able to manage more receiver nodes. As a result, we will have fewer repair nodes with larger buffers.

To show that let us assume that a repair node can receive up to n_{ACK} acknowledgments for each packet sent by the sender node. This means that a repair node receiving ACKs for all packets sent by the sender node will be able to handle up to n_{ACK} receiver nodes and will need a buffer capable of storing one single packet. Under our scheme, each packet is acknowledged by $1/N_{max}$ of the receiver nodes. This means that each repair node will be able to handle up to $n_{ACK} \times N_{max}$ receiver nodes. The buffer space requirements of each repair node will be multiplied by N_{max} but the total buffer space requirements of all repair nodes will remain unchanged because we will need N_{max} less of them.

III.2 Flow Control

Our buffer management scheme provides reliable transmission in a multicast session, because the packets requested from receiver nodes by NAKs are always available in repair node's buffer, as long as there is no overflow. In this section, we show that buffer overflow will occur without flow control, and we describe our flow control scheme to avoid this buffer overflow.

III.2.1 Buffer Overflow Issues

The most straightforward method to avoid buffer overflow is removing some old packets to make space for the newly arrived packets. In our case, we might have to remove some old packets from the buffer of a repair node before that repair node has received the proper randomized ACKs from its receiver nodes. As a result, the repair may not be able to honor all retransmission requests from its receiver nodes. This will increase the workload of the sender node and result in additional error recovery delay.

Unfortunately, no buffer management scheme can avoid overflowing a finite buffer without flow control. This property can be easily proved using M/M/1/K queuing system [8]. Let us assume the packets arrive at repair node with a Poisson process with arrival rate λ. Packets are discarded from the buffer by refreshment policy. Let us call this discard rate μ. In our case, the discard rate is exponentially distributed with mean $1/\mu$, because it depends on the round trip time between the repair node and receiver nodes. If the buffer size is limited to K and an infinite number of packets arrive, we can apply the M/M/1/K queuing system [8].

$$P(\text{buffer is full}) = [(1 - (\lambda/\mu))\cdot(\lambda/\mu)^{K}]/[1 - (\lambda/\mu)^{K+1}] \qquad \text{for } \lambda < \mu.$$

The condition for the absence of the buffer overflow is then given by

$$[(1 - (\lambda/\mu))\cdot(\lambda/\mu)^{K}]/[1 - (\lambda/\mu)^{K+1}] = 0,$$

which simplifies into

$$[(1 - (\lambda/\mu))\cdot(\lambda/\mu)^{K}] = 0,$$

which can only be satisfied when either $\lambda = 0$ or K goes to infinity.

III.2.2 Proposed Flow Control Scheme

Our scheme can also perform flow control using a *sliding-window* mechanism. It will then allow the sender node to transmit up to a fixed number S_{sender} of packets without waiting for their ACKs. This window is said to be sliding because each ACK arrival at the sender allows the sender to send one more packet.

Algorithm:

```
Join multicast group
Set $Dj = 0$ for all $1 \leq j \leq N^{RP}$
Send packets 1 to $B^*$
Set C = $B^*$
Begin loop
  Switch (event)
       event :  ACK for packet $D_j$ from repair node $j$ arrives
                Select $C_{min} = \min\{D_j \mid 1 \leq j \leq N^{RP}\}$
                Send packets from $C + 1$ to $C + C_{min}$
                Set $C = C + C_{min}$
                Break
  End switch
End loop
Leave multicast group
```

Figure 2. Sender node algorithm

A first way to accommodate this sliding window is to add S_{sender} extra slots in the buffers of each repair node. Define D_j as the highest numbered packet that repair node j can safely discard based on the randomized ACKs sent by its receiver nodes. One simple mechanism to advance the sliding window would be to have all repair nodes sending to the sender node an ACK for each packet D_j they have discarded from their buffer. As we observed at the end of the previous subsection, our scheme requires N_{max} less repair nodes than a scheme where all repair nodes receive ACKs from all their receiver nodes for all packets sent by the sender node. Hence the sender node is not likely to be flooded by an inordinate number of ACK packets for a wide range of small to medium multicast group sizes.

Let us now extend our scheme to the case where different repair nodes may have different buffer sizes. Let B^* denote the minimum buffer size of all repair nodes. Then, the sender node can initially send up to packet B^*. Let D_j be the last packet acknowledged by repair node j. The highest-numbered packet C_{min} that was acknowledged all repair nodes is then given by

$$C_{min} = \min\{D_j \mid 1 \leq j \leq N^{RP}\}, \tag{6}$$

where N^{RP} is the number of repair nodes.

When all repair nodes have acknowledged packet C_{min}, the sender can now safely sends packets up to $C_{min} + B^*$.

Algorithm:

```
Join multicast group
Set $L_i = 0$ for all $1 \leq i \leq N_j$
Set $R_{rand}^j$ = random(1, $N_s$)
Set count = 0
Begin loop
   Switch (event)
      event :  Data from sender node arrives
               Store packet in buffer
               Increment count
               Break
      event :  ACK from a receiver node arrives
               Set $D = \min\{L_i \mid 1 \leq i \leq N_j\}$
               Remove up to packet $D$ from its buffer
               If count = $R_{rand}^j$ mod $N_s$ then
                      Send ACK for $D$ to sender node
               Endif
               Break
      event :  NAK from a receiver node arrives
               Retransmits missing packet
               Set $D = \min\{L_i \mid 1 \leq i \leq N_j\}$
               Remove up to packet $D$ from its buffer
               If count = $R_{rand}^j$ mod $N_s$ then
                      Send ACK for $D$ to sender node
               Endif
               Break
   End switch
End loop
Leave multicast group
```

Figure 3. Algorithm for repair node j ($1 \leq j \leq N^{RP}$)

As we mentioned earlier, this scheme will work well as long as there are not too many repair node. When this is not the case, we can randomize ACKs at the repair node level to allow the sender node to handle more repair nodes. Our fully randomized scheme assumes every repair node has a large enough, but finite, buffer. Define a slot as the transmission time of a packet. Recall that, under our scheme, repair node j receives an average of N_j/N_{max} ACKs from its N_j receiver nodes during every slot.

At the beginning of each multicast, the sender node will send to all repair nodes a packet containing an integer N_s specifying the factor by which all repair nodes must divide the number of ACKs they will send to the sender node. Each repair node j will then select a random number R_{rand}^j between 1 and N_s

$$R_{rand}^j = \text{random}\ (1, N_s), \qquad (7)$$

```
Algorithm:
  Join multicast group
  Select $N_{rand}^i$ = random(1, $N_{max}$)
  Set count = 0
  Set $k$ = 0
  Begin loop
    Switch (event)
      event : Data from sender node arrives
          Store packet in buffer
          Increment count
          Note packet as successfully received
          If count = $N_{rand}^i$ mod $N_{max}$ then
              Send ACK for count to sender node
              Increment $k$
          Endif
          Break
      event : Missing packet detected
          Send NAK to repair node
          Find last packet that was correctly received
          as well as all its predecessors
          Piggyback ACK for that packet
          Select $N'^{i}_{rand}$ = random(1, $N_{max}$)
          If count + $N'^{i}_{rand} > N_{rand}^i + kN_{max}$ then
              Set $N_{rand}^i$ = (count + $N'^{i}_{rand}$) mod $N_{max}$
          Endif
          Break
    End switch
  End loop
  Leave multicast group
```

Figure 4. Algorithm for receiver node *i* ($1 \le i \le N$)

Rather than sending an ACK to the sender node for each packet it discards from it buffer, repair node will send its first ACK for the packet it will discard during slot R_{rand}^j. After that, it will only send ACKs to the sender every N_s slots. Hence it will only send during slots RP_j such that

$$RP_j = R_{rand}^i + kN_s, \text{ for } k = 0, 1, 2, 3, \ldots \tag{8}$$

The total number M_{RAND} of ACKs sent by repair nodes during a multicast session of duration M packets is thus given by

$$M_{RAND} = M/N \tag{9}$$

We now show the minimum buffer size required to implement a given sliding window of duration S_{sender} under our fully randomized scheme:

Each receiver node will wait for up to $N_{max} - 1$ slots to notify the repair node it has successfully received a given packet. Each repair node will wait for up to $N_s - 1$ slots to notify the repair node it has discarded a given packet from its buffer. As a result, the minimum buffer size required for achieving a sliding window of duration S_{sender} is given by

$$S_{sender} + N_{max} + N_s - 2 \tag{10}$$

Our flow control scheme has two advantages. First, it provides scalability, because it reduces the number of ACKs sent by repair nodes. This feature reduces not only the sender's workload, but also network traffic in a multicast session. Second, it guarantees reliability, as we never loose any packet anywhere and the buffers of the repair nodes never experience any overflow. The summarized algorithm for each type of nodes is given in Figure 2 to Figure 4.

IV. Performance Analysis

In this section, we show the performance of the proposed buffer management scheme by computer simulation. All the simulation experiments are performed for up to 100 receiver nodes per repair node.

IV.1 Feedback Implosion

The first main advantage of our scheme is that it significantly reduces the number of feedbacks from receiver nodes, required to safely discard the packets from repair node's buffer. Over a session involving the transmission of m packets, the maximum number of feedbacks from receiver node i is given by

$$\lceil m/N_{max} \rceil + 1 + m \cdot LP_i ,$$

where $\lceil m/N_{max} \rceil + 1$ is the number of ACKs sent by node i, including the ACK for the last packet of the transmission and $m \cdot LP_i$ is the number of NAKs sent by the same node.

Hence the total number F_{RAND} of feedbacks received by the repair node from its N receiver nodes will obey the inequality

$$F_{RAND} \leq \sum_{i=1}^{N} \left(\lceil m/N_{max} \rceil + 1 + m \cdot LP_i \right) \tag{11}$$

When all link failure probabilities are equal, that is, $LP_1 = LP_2 = \ldots = LP_N = LP$, equation (11) simplifies into

$$F_{RAND} \leq N \left(\lceil m/N_{max} \rceil + 1 + m \cdot LP \right) \tag{12}$$

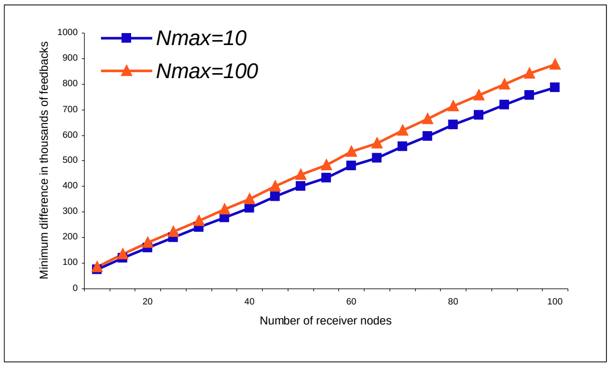

Figure 5. Minimum $\Delta_{\min}$

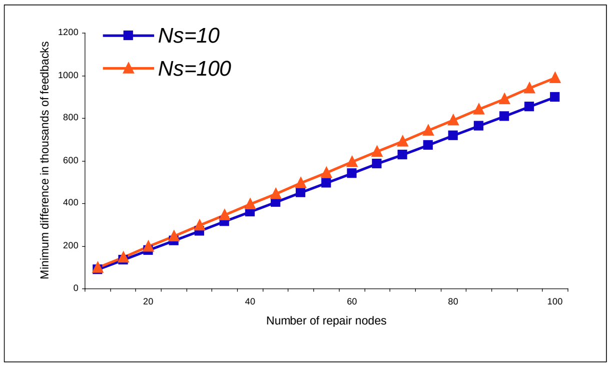

Figure 6. Minimum $\Delta_{\min}'$

Under the same assumptions, the number of feedbacks F_{ACK} for an ACK-based scheme, where all receiver nodes acknowledge all the packets they receive, is given by

$$F_{ACK} = mN$$

The minimum difference $\Delta_{\min}$ between the numbers of feedbacks of the two schemes will obey the inequality

$$\Delta_{\min} \geq N\,(m - \lceil m/N_{max} \rceil - 1 - m \cdot LP) \qquad (13)$$

This means our scheme guarantees at least a $\Delta_{\min}$ reduction of the number of feedbacks. We also compare the number of feedbacks sent by repair nodes for sender node's flow control. The minimum difference $\Delta_{\min}'$ between the numbers of feedbacks for the two schemes can be given by

$$\Delta_{\min}' \geq N_{rp}(m - \lceil m/N_s \rceil) \qquad (14)$$

where N_{rp} is the number of repair nodes in multicast session and N_s is the maximum random number given by sender node.

Figure 5 shows how the minimum difference increases as N increases for two N_{max} values. Also, figure 6 shows that for performing flow control for two N_s values. We assumed that m was equal to 10,000, which roughly represents a transfer of 10 megabytes when the packet size is 1 kilobyte. We also assumed that the individual loss probabilities LP_i would be uniformly distributed between 0.01 and 0.2. When there are 100 receiver nodes, the minimum difference is about one million feedbacks. This result indicates that our scheme provides efficient buffer management functionality for repair node by reducing the number of feedbacks sent by receiver nodes. This feature provides scalability, since each repair node will be able to handle more receiver nodes.

Table I. Configuration Parameters

Sending rate *S*	128packets/second
Avg_RTT	40ms
Avg_OTT	15ms
N	100 receiver nodes
NAK_TIMER_i, $1 \le i \le N$	30ms
m	10,000 packets (≅10Mbyte)

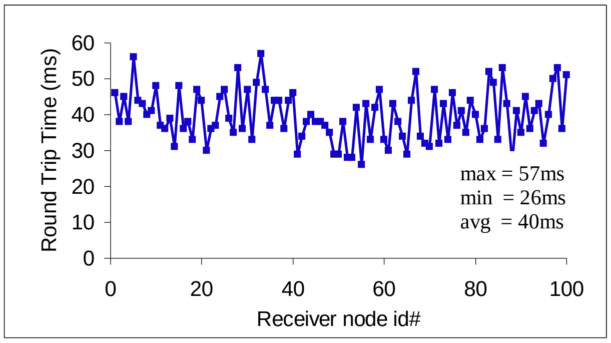

Figure 7. Simulated round trip time

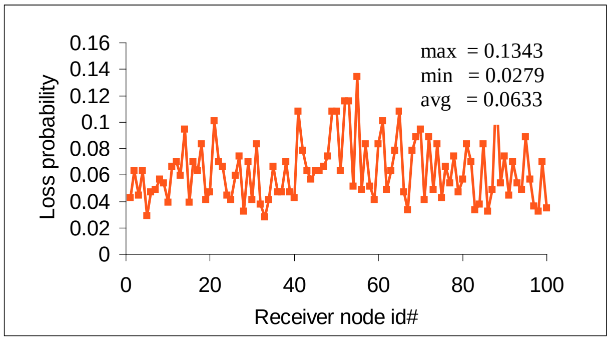

Figure 8. Simulated loss probability

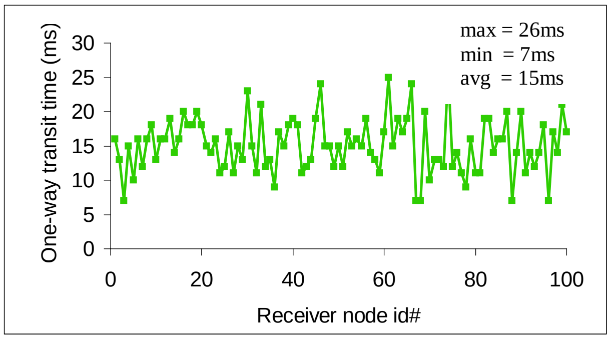

Figure 9. Simulated one-way transit time

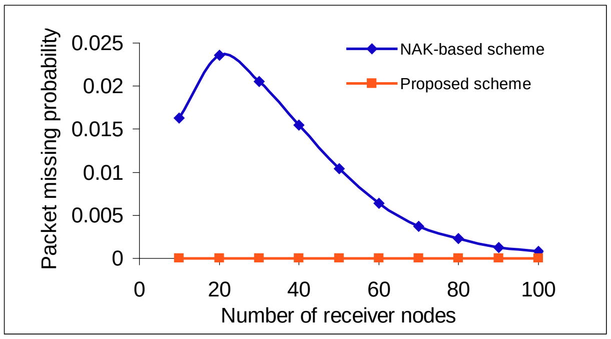

Figure 10. Packet loss probability

IV.2 Additional Retransmissions

The proposed scheme does not require any additional retransmission either from its upper-stream repair nodes or sender node, because it always has in its buffer all packets that can be requested by any of its receiver nodes. This feature provides fast error recovery for receiver nodes and reduces network traffics between the repair nodes.

In NAK-based schemes, the repair node batches NAKs for a packet and retransmits the packet periodically as long as there is a pending NAK for that packet. Let us call the period δ and assume that the packets arrive at repair node with a Poisson process with mean arrival rate λ. If the repair node has B buffers, we can define the random variable $N_A(\delta)$ to represent the number of packet arrivals at the repair node within a time interval of length δ. In order to perform at least one retransmission successfully, the following condition should be satisfied.

$$P(N_A(\delta) \geq B) = 1 - \sum_{n=0}^{B-1} \frac{(\lambda\delta)^n e^{-\lambda\delta}}{n!} = 0, \tag{15}$$

which simplifies into

$$\sum_{n=0}^{B-1} \frac{(\lambda\delta)^n e^{-\lambda\delta}}{n!} = 1$$

$$\sum_{n=0}^{B-1} \frac{(\lambda\delta)^n}{n!} = e^{\lambda\delta}$$

Since we have $e^{\lambda\delta} = \sum_{n=0}^{\infty} \frac{(\lambda\delta)^n}{n!}$, equation (15) can only be satisfied when B goes to infinity.

Hence, a NAK-based scheme must require the repair nodes to buffer all packets for an infinitely long amount of time to achieve full coverage of all retransmission requests by the repair node.

In NAK-based schemes using a timer mechanism, repair nodes discard some packets from its buffer after a time interval I without considering whether these packets were received by all its receiver nodes. As a result, some packets might be removed from the repair node buffer while their retransmission could still be requested by one of the receiver nodes. In this case, the missing packets will have to be resent from either an upper repair nodes or the sender node. In most cases, the packets will have to be resent by the sender node, especially when all repair nodes apply the same buffer management policy and discard the same packets at the same time. This generates unnecessary traffics decreasing the whole Internet performance.

To evaluate the number of these additional retransmissions, we define M_{NAK} as a missing probability of repair node. The miss occurs when some receiver nodes request retransmission for the packets that have already been discarded from the buffer. Hence, M_{NAK} means the probability that the repair node cannot retransmit the packet for any other receiver because it was already removed from buffer.

In NAK-based scheme, some packets are available if the NAKs arrive before the timer is expired. Let us call this probability *A*. The probability might be very close to 1 if the repair node has large enough timer value. If we assume that *A* is equal to 0.9, the repair node will only be unable to deal with 10% of the retransmission requests sent by other nodes, because the requested packet will be removed before any NAK arrives. We also need to take into account the impact of lost NAKs. If the NAKs of all the receiver nodes that did not receive the packet fail to reach the repair node, then the repair node will discard the packet before it receives a second request for that packet from one of the receiver nodes. Hence, a realistic estimate of the probability M_{NAK} is given by

$$\begin{aligned} M_{NAK} &= (1-A)\times P(\text{some other nodes did not receive the packet}) \qquad (16)\\ &\quad + A \times P(\text{all other nodes that did not receive the packet failed to notify the repair node})\\ &= (1-A)\times (1-P(\text{all other nodes have received the packet}))\\ &\quad + A \times P(\text{all other nodes that did not receive the packet failed to notify the repair node})\\ &= (1-A)(1-\prod_{i=1}^{N}(1-LP_i)) + A(\prod_{i=1}^{N}(1-LP_i+LP_i^2) - \prod_{i=1}^{N}(1-LP_i)) \end{aligned}$$

If *A* is equal to 1, the minimum probability of M_{NAK} is given by

$$M_{NAK} \geq \prod_{i=1}^{N}(1-LP_i+LP_i^2) - \prod_{i=1}^{N}(1-LP_i) \qquad (17)$$

Given the difficulty of finding a closed-form expression for the parameter *A*, we decided to simulate the behavior of a system with 100 receiver nodes per repair node. The parameters of this model are summarized in table I. To generate the loss probability of each receiver node, we applied the formula $S = 1.22/(RTT_{s,i}\sqrt{LP_i})$ (from [11]), where *S* is the packet sending rate in packets/sec, $RTT_{s,i}$ is the round trip time from the sender node to receiver node *i* and LP_i is the loss probability between the sender node and receiver node *i*. This assumes that the sender node transmits packets in TCP-friendly manner and each node in the multicast session uses the UDP protocol.

We simulated the round-trip times $RTT_{s,i}$ as Poisson random variables each having mean *Avg_RTT*. Similarly, the one-way transit times $OTT_{i,rp}$ between a receiver node *i* and its repair node *rp* were also simulated by Poisson random variables with mean *Avg_OTT*. Figure 7, 8 and 9 respectively show our measurements for roundtrip time, loss probability and one-way transit time for 100 receiver nodes.

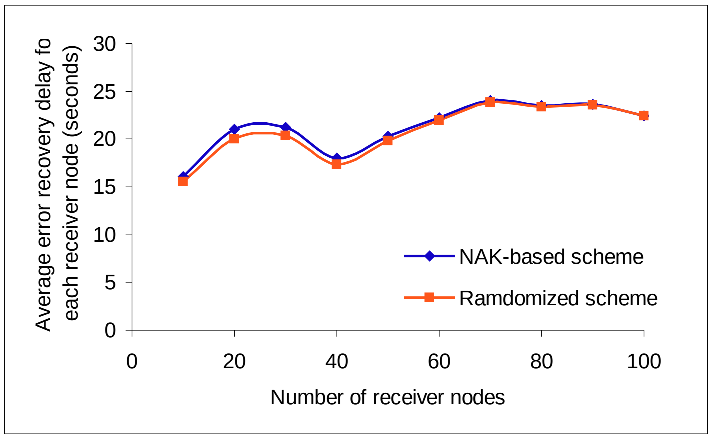

Figure 11. Difference between the error recovery delays of our scheme and a NAK-based scheme

Using the configuration parameters in Table I, we evaluate the probability that a requested packet will not be present in the repair node. In particular, we compared the performance of our scheme with that of a NAK-based scheme keeping all packets in the repair node buffer for 60 ms. Figure 10 shows how the probability of not finding a requested packet in the repair node buffer is affected by the number of receiving nodes per repair node. As proved in previous section, the missing probability of our scheme is zero, because the requested packets from receiver nodes are always available in the repair node's buffer.

We also see that the NAK-based scheme performs significantly worse than our scheme despite having a timer delay equal to the sum of two one-trip time in one NAK_TIMER delay. We should also mention that the conditions under which the comparison is performed are very favorable to NAK-based scheme as we assumed that all requests from receiver nodes arrive before timer is expired (*A*=1). The performance of NAK-based scheme will improve whenever the repair nodes have very large buffers as well as a long enough timer values. However, this would result in an inefficient use of the available buffer space, because too many packets will remain in buffer for a long time. In addition, the absence of an efficient buffer management scheme is likely to cause sooner or later buffer overflow.

IV.3 Error Recovery Delay

The additional retransmissions, evaluated in previous subsection, increase the error recovery delay, because the repair node cannot retransmit the requested packet immediately. The packets should be retransmitted from its original sender node or upper stream repair node. This might double or triple the error recovery delay. Also, these additional retransmissions cause unnecessary traffics between the repair nodes.

To evaluate the minimum delay difference between both schemes, we assume that the additional retransmission is always correctly transmitted from the upper stream repair node. Otherwise, the difference would be much more prominent. Under these assumptions, each receiver node measures the average recovery delay over all packet loss it experienced. These results are shown in Figure 11. Even under simulated parameter values, the proposed scheme performs slightly better than the NAK-based scheme. Also, the result indicates that the most reasonable number of receiver nodes per repair node is equal to 40.

Note that the difference becomes progressively close to 0 when the number of receiver nodes per repair node increases above 20. This is because the packet missing probability of the NAK-based scheme is decreased with group size. However, we should consider that the repair node has to retransmit the requested packets to more receiver nodes to maintain this performance, which generates unacceptable traffics between the repair node and its receiver nodes.

V. Conclusion

We have proposed a buffer management scheme combining NAKs and infrequent ACKs to provide scalability and reliability in a multicast session. Under our scheme, receiver nodes send negative acknowledgments to repair nodes to request packet retransmissions. At infrequent intervals, they also send ACKs to their repair nodes to indicate which packets they can safely discard. Our scheme reduces delay in error recovery, because the packets requested from the repair nodes are always available in their buffers. It achieves this goal without increasing the server workload because (a) each receiver node only sends infrequent positive acknowledgments and (b) their sending times are randomized among all the receiver nodes. In addition, it greatly reduces the number of repair nodes required to handle a given number of receiver nodes. We have also shown how our scheme can provide full flow control and eliminate buffer overflows by having repair nodes sending ACKs at infrequent intervals to the sender node. Hence it provides an acceptable trade-off between ACK-based and NAK-based schemes, using both positive and negative acknowledgments to achieve reliability and scalability.

More work is still needed to ascertain the optimal randomization intervals for both receiver nodes and repair nodes.

References

[1] J. Baek, "A Hybrid Configuration of ACK Tree for Multicast Protocol," *Proceedings of the 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems* (SPECTS 2002), pp. 852–856, San Diego, USA, July 2002.

[2] K. P. Birman et al., "Bimodal Multicast," *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.

[3] M. Costello and S. McCanne, "Search Party: Using Randomcast for Reliable Multicast with Local Recovery," *Proceedings of the 18th IEEE Conference on Computer Communication*s (INFOCOM '99), pp. 1256–1264, New York, NY, March 1999.

[4] S. Floyd et al, "A Reliable Multicast Framework for Lightweight Sessions and Application-Level Framing," *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[5] T. Gemmel et al., "The use of Forward Error Correction in Reliable Multicast," IETF draft-ietf-rmt-info-fec-02.txt, October 2002

[6] K. Guo, and I. Rhee, "Message Stability Detection for Reliable Multicast," *Proceedings of the 19th IEEE Conference on Computer Communications* (INFOCOM 2000), pp. 814–823, New York, USA, March 2000.

[7] M. Kadansky et al., "Reliable Multicast Transport Building Block: Tree Auto-Configuration," IETF Internet Draft, draft-ietf-rmt-bb-tree-config-01.txt, November 2000.

[8] L. Kleinrock, *Queuing Systems*, Volume I: Theory, John Wiley & Sons, 1975.

[9] S. J. Koh et al., "Configuration of ACK Trees for Multicast Transport Protocols," ETRI Journal, 23(3):111–120, September 2001.

[10] B. Levine, and J. J. Garcia-Luna-Aceves, "A Comparison of Reliable Multicast Protocols," *ACM Multimedia Systems Journal*, 6(5): 334–344, August 1998.

[11] J. Mahdavi and S.Floyd, "TCP-friendly unicast rate-based flow control," http://www.psc.edu/networking/papers/tcp_friendly.html, January 1997.

[12] C. Maihofer and K. Rothermel, "A Robust and Efficient Mechanism for Constructing Multicast Acknowledgment Trees," *Proceedings of the 8th IEEE International Conference on Computer Communications and Networks* (ICCCN '99), pp. 139–145, Boston-Natick, MA, October 1999.

[13] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgement Options," RFC 2018, October 1996.

[14] O. Ozkasap, R. van Renesse, K. P. Birman, and Z. Xiao, "Efficient Buffering in Reliable Multicast Protocols," *Proceedings of the First International Workshop on Networked Group Communication* (NGC99), pp. 188–203, Pisa, Italy, November 1999.

[15] S. Pingali, D. Towsley, J. F. Kurose, "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols," *IEEE Journal on Selected Areas in Communications*, pp. 221–230, April 1997.

[16] R. Renesse, Y. Minsky, and M. Hayden, "A Gossip-Style Failure Detection Service," Proceedings of MIDDLEWARE '98, pp. 55–70, The Lake District, England, September 1998.

[17] B. Whetten and G. Taskale, "The Overview of Reliable Multicast Transport Protocol II," *IEEE Networks*, 14(1):37–47, January-February 2000.

[18] Z. Xiao, K. P. Birman, R. Renesse, "Optimizing Buffer Management for Reliable Multicast," *Proceedings of the International Conference on Dependable Systems and Networks* (DSN'02), pp. 187–202, Washington, DC, June 2002.

[19] M. Yamamoto, Y. Sawa, S. Fukatsu and H. Ikeda, "NAK-based Flow Control Scheme for Reliable Multicast Communications," *IEICE Transactions on Communications*, E82-B(5):712–720, May 1999.

[20] K. Yamamoto, M. Yamamoto, H. Ikeda, "Performance Evaluation of ACK-based and NAK-based Flow Control Mechanisms for Reliable Multicast Communications," *IEICE Transactions on Communications*, E84-B(8) 2313–2316, Aug. 2001.

[21] R. Avatar, J. Griffon, M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications," *Proceedings of the 3rd ACM International Conference on Multimedia*, pp. 333–344, San Francisco, CA, November 1995.