A Comparison of
Sender-Initiated and Receiver-Initiated
Reliable Multicast Protocols[1]

Don Towsley and Jim Kurose[2]
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
{towsley,kurose}@cs.umass.edu

Sridhar Pingali
Dept. of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003[3]

Abstract

Sender-initiated reliable multicast protocols based on the use of positive acknowledgments (ACKs) can suffer performance degradation as the number of receivers increases. This degradation is due to the fact that the sender must bear much of the complexity associated with reliable data transfer (e.g., maintaining state information and timers for each of the receivers and responding to receivers' ACKs). A potential solution to this problem is to shift the burden of providing reliable data transfer to the receivers - thus resulting in receiver-initiated multicast error control protocols based on the use of negative acknowledgments (NAKs). In this paper we determine the maximum throughputs for generic sender-initiated and receiver-initiated protocols for two classes of applications: *(i)* one-many applications where one participant sends data to a set of receivers, and *(ii)* many-many applications where all participants simultaneously send and receive data to/from each other. We show that a receiver-initiated error control protocol which requires receivers to transmit NAKs point-to-point to the sender provides higher throughput than a sender-initiated counterpart for both classes of applications. We further demonstrate that, in the case of a one-many application, replacing point-to-point transfer of NAKs with multicasting of NAKs coupled with a random backoff procedure provides a substantial additional increase in the throughput of a receiver-initiated error control protocol over a sender-initiated protocol. We also find, however, that such a modification leads to a throughput degradation in the case of many-many applications.

---

[2]Correspondence author.

[3]The author is now with Sapient Corporation, One Memorial Drive, Cambridge MA, 02142. spinga@msmail.sapient.com

1 Introduction

During the past three years, there has been widespread use of applications that support real-time interactive group activities over wide area networks. Applications that support video (nv[8], vic[16]) and voice (vat[12], NeVot[23]) do not require reliable data transfer, while applications such as shared whiteboards (e.g., wb [13], shdr [17]) do. The requirement of reliable data transfer for this last set of applications poses a difficult challenge to network designers - namely how to design and implement a reliable multicast protocol that can handle 100s or 1000s of participants[4]. Other group applications requiring the timely reliable delivery of data include distributed interactive simulations, [10].

In this paper we examine two different approaches to providing reliable, scalable multicast communication. The *sender-initiated* approach places the responsibility for providing reliable multicast on the sender, which maintains state information on all receivers to which it is multicasting. This is accomplished by having receivers return positive acknowledgments (ACKs) for every correctly received packet, and having the sender use timers to detect potential packet losses. The alternate approach, a *receiver-initiated* approach, shifts most of the responsibility for reliable data delivery to the receivers. Each receiver is responsible for detecting lost packets and informing the sender via negative acknowledgments (NAKs) when it requires the retransmission of a packet. In the case of an application consisting of a single sender transmitting reliably to many receivers (referred to as a *one-many* application in this paper) we observe through simple analyses that a simple receiver-initiated protocol which requires receivers to return negative acknowledgments (NAKs) to the sender over point-to-point channels provides substantially better performance (in terms of the maximum supportable throughput of successfully transmitted messages) than a sender-initiated protocol. Further substantial improvement is obtained by the multicasting of NAKs coupled with the introduction of random delays prior to the transmission of a NAK. In the case of an application where all participants act as both senders and receivers (referred to as a *many-many* application in this paper) simple analyses illustrate that the receiver-initiated protocol which requires receivers to return negative acknowledgments (NAKs) to the sender over point-to-point channels almost doubles throughput over a sender-initiated counterpart. However, unlike the one-many case, the multicasting of NAKs in the many-many scenario does not fare as well, leading to only a small increase in throughput over the sender-initiated counterpart.

Past research on reliable multicast transport protocols has focussed primarily on sender-initiated protocols, e.g., [21, 22, 24, 25, 26, 18, 1]. Several efforts, however, have proposed or examined specific NAK-based receiver-oriented protocols. For example, Ramakrishnan and Jain, [20] designed and evaluated, through simulation, a receiver-initiated window-based reliable multicast protocol suitable for a local area network. Floyd *et al.* [7] have used similar ideas to implement a reliable multicast protocols suitable (similar in spirit to protocol N2 that we consider in this paper) for the wb whiteboard program in a wide-area environment. Performance of their protocols is evaluated through simulation in linear, star, and tree networks. RAMP [2] is a NAK-based protocol that selectively unicasts or multicasts a retransmission, depending on the number of received NAKs.

Our work differs from previous work on reliable multicast protocols in three significant ways. First,

---

[4]IETF conferences multicast over the MBone have had hundreds of multicast participants worldwide [3]. wb has been used with more than 1000 users [7].

rather than provide a detailed design and analysis of a specific reliable multicast protocol, we study three generic protocols, one that is sender-initiated and two that are receiver-initiated. We focus on the *fundamental differences* between them, i.e., the use of ACKs vs. NAKs and the use of point-to-point channels for NAKs vs. broadcasting NAKs. Second, recognizing that communication bandwidths are expected to grow at a much higher rate than processing speeds during the next decade, we focus on the *processing requirements* of these protocols at both the sending and receiving hosts rather than the communication bandwidth requirements. Previous analyses have focussed on the bandwidth requirements of different reliable multicast protocols. Third, we are concerned with *scalability*: how well each of these approaches will handle large numbers of receivers. The scalability of specific ACK- or NAK-based protocols is examined by [7, 18, 1]. One related analytic work is [5], which proposes a NAK-based protocol similar to our N1 and compares it with an ACK-based protocol. The focus in [5], however, is on the performance effects of sending periodic state information. They assume lossless network and do not consider the overhead of ACK and NAK processing – considerations that are central to our analysis.

The remainder of the paper is organized as follows. The descriptions of the different protocols, applications and network are found in Section 2. Expressions for the maximum throughput at the sender and receivers for the sender-initiated protocol is contained in Section 3. Similar analyses for the receiver-initiated protocols are found in Section 4. Section 5 compares these various protocols for both one-many and many-many applications. Finally, concluding remarks are given in Section 6.

2 Protocols and System Model

We now describe the two approaches for reliable transmission of data from a sender to multiple receivers. We emphasize that we are not proposing new, specific protocols below. Rather, we seek to define generic protocols that capture the important characteristics of ACK- and NAK-based protocols in general. As noted above, the sender-initiated approach, based on the use of ACKs, places much of the burden for ensuring reliable packet transmission on the sender. The receiver-initiated approach, based on NAKs, shifts this burden to the receivers. The section ends with a description of the applications and the network model.

2.1 Sender-Initiated Protocols

A sender-initiated protocol requires the sender to maintain a list (called the ACK list), for each packet, of the receivers from which it has received a positive acknowledgment (ACK). Each time a receiver correctly receives a packet, it returns an ACK. Upon receipt of the ACK, the sender updates the ACK list for the corresponding packet. Lost packets are dealt with through the use of timers. Specifically, the sender starts a timer at the time of a packet transmission and, if it expires prior to the sender having received ACKs for this packet from every receiver, the sender retransmits the packet and restarts the timer.

Most traditional point-to-point protocols (e.g., TCP, HDLC, TP4) and many early multicast/broadcast protocols [24, 22, 9, 20, 25] are based on this approach. Rather than focus on a specific protocol, we will instead focus on a *generic* protocol which exhibits the following behavior.

3

- error recovery is selective repeat, i.e., only packets that are suspected to be lost or corrupted are retransmitted,

- whenever a sender transmits or retransmits a packet, it multicasts to all receivers and starts a timer,

- whenever a receiver receives a packet correctly, it transmits an ACK to the sender over a point-to-point connection,

- whenever a timer expires, the sender remulticasts the corresponding packet. In this paper, maximum protocol throughput will be our primary performance metric. This performance measure is invariant to the order in which packets are transmitted and thus we make no assumptions about whether new packets or retransmissions are given priority. (We note that when packet *delay* is the performance measure of interest, the transmission order is important.)

This will be referred to as protocol (A).

This protocol can be optimized in numerous ways, such as grouping ACKs for different packets into a single control packet (see [24, 22, 18] for examples of such optimizations). We will not consider such optimizations in this paper.

2.2 Receiver-Initiated Protocols

A receiver-initiated protocol places the responsibility for ensuring reliable packet delivery on the receivers. The sender continues to transmit new data packets until it receives a negative acknowledgment (NAK) from a receiver. When this occurs, the sender then retransmits (i.e., again multicasts) the packet required by that receiver. The role of the receiver is to check for lost packets. If it decides that it has not received a particular packet, it transmits a NAK to the sender. In order to guard against either the loss of the NAK or the subsequent packet retransmission, the receiver uses timers in a manner analogous to the sender's use of timers in sender-initiated protocols. Typically a receiver will detect a lost packet when it receives packets with larger sequence numbers (or after a timeout if it is expecting a retransmission). In the case that the sender does not always have packets to send (i.e., must occasionally wait for data to be produced at a higher level), it may be necessary for the sender to multicast periodic state information (e.g., giving the sequence number of the last transmitted packet) while idle [5, 7]. We can ignore this potential overhead in our ensuing analysis, as we are primarily interested in determining the maximum throughput of the protocols.

We will focus on a generic receiver-initiated protocol (N1) that exhibits the following behavior

- the sender multicasts all packets,

- whenever a receiver detects a lost packet, it transmits a NAK to the sender over a point-to-point channel and starts a timer,

- the expiration of the timer without prior reception of the corresponding packet serves as the detection of a lost packet.

and the following variation (N2) suggested by Ramakrishnan and Jain [20] for a LAN and Jacobson and Floyd [11, 7] for a WAN

- the sender multicasts all packets and state information

- whenever a receiver detects a lost packet, it waits for a random period of time and then multicasts a NAK to the sender and all other receivers, and starts a timer,

- upon receipt of a NAK for a packet which a receiver has not received, but for which it initiated the random delay prior to sending a NAK, the receiver starts the timer and behaves as if it had sent that NAK,

- the expiration of the timer without prior reception of the corresponding packet serves as the detection of a lost packet.

This second protocol attempts to ensure that at most one NAK is returned to the sender per packet transmission by staggering the generation of NAKs and multicasting them to all other receivers. Hence, the first NAK generated usually arrives to all other receivers prior to their generating additional NAKs. As with the sender-initiated protocols, there are numerous optimizations possible which we will not explore in this paper (see [20] for an example).

2.3 Application, Network, and Error Models

There are $R + 1$ participants in the application requiring the reliable transmission of data. We consider two types of applications. In the first, the *one-many* application, one sender transmits a continuous stream of packets to R receivers. This is exemplified by a telelecture or a teleconference where one participant acts as a sender all of the time or for a long period of time. In the second, the *many-many* application, all $R+1$ participants send as well as receive data reliably. More precisely, in the many-many case, each participant is equally likely to act as a sender for a randomly chosen packet (with probability $1/(R+1)$). This is exemplified by distributed interactive simulations (DISs) [10] where each participant is required to send information reliably to many other participant while, simultaneously, receiving from many other participants.

In both cases, we assume that all loss events at all receivers for all transmissions are mutually independent and that the probability of packet loss, p, is independent of receiver. We further assume that ACKs or NAKs are never lost.

The assumption that losses among receivers occur as independent events would hold if the multicast backbone loss is small (which we observe experimentally in the MBone in [27]) and if there is low loss from the sender into the backbone. If losses are correlated, we believe that it is possible, using the concept of association of random variables, [6], to show that our analyses produce pessimistic bounds on throughput as a consequence of this independence assumption. The assumption that ACKs or NAKs are never lost is reasonable as control packets are small. If desired, this assumption can be relaxed by following the analysis given in [24].

3 Throughput Analysis of Sender-Initiated Protocols

We begin by considering the transmission of a packet from one participant, henceforth referred to as the sender, to R identical participants, henceforth referred to as receivers. As the behavior of the sender differs from that of a receiver, we consider their behaviors separately. Throughout our analysis we will introduce a number of different parameters as we need them. These are collected in Table 1 for ease of lookup. We consider the sender first.

Our goal in analyzing both the sender and receiver will be to compute the expected amount of *processing time* required (at the sender and at a receiver, respectively) in order for a packet to be *successfully* delivered from the sender to all of the receivers. This processing time includes the amount of time needed to send/receive the original packet as well as any retransmissions of that packet, handle timeouts associated with that packet, and send ACK or NAK packets. The reciprocal of these processing times will give us the maximum *rates* at which a sender and a receiver can process new packets. For a given protocol ((A), (N1), or (N2)) and for a given application structure (one-many, many-many), the maximum supportable *throughput* for a given protocol can be determined from these rates, as discussed below. This protocol throughput will be our performance measure of primary interest.

Let us being by focusing on the expected processing time at the sender under the (A) protocol. This is the processing time required by the sender to *successfully* multicast a packet to all receivers. Let us denote this processing requirement by X^A. When the sender obtains data from a higher level protocol/application, a packet containing this data is constructed and transmitted and a timer set. Following this, the sender must process every ACK received for the packet. In addition, each time that the timer expires and the sender has not received ACKs from all receivers for this packet, the packet must be remulticasted and the timer restarted. Given these considerations, we have

$$X^A = X_p(1) + \sum_{m=2}^{M} (X_t(m) + X_p(m)) + \sum_{i=1}^{L^A} X_a(i) \tag{1}$$

where $X_p(m)$ is the packet processing requirement associated with the m-th transmission of a given packet, $X_t(m)$ is the requirement for processing the timer interrupt that will result in the mth transmission of the packet, $X_a(i)$ is the processing requirement for the i-th ACK received for this packet, L^A is the total number of ACKs received for the packet, and M is the total number of transmissions required to transmit the packet correctly to all R receivers. These processing times include the times required to perform a context switch along with processing costs specific to the operation. For example, $X_a(i)$ includes the processing required to take the packet up the protocol stack to the transport layer (or higher). It also includes the time required to update the ACK list and, possibly, to turn off the timer.

We assume that the processing requirements have general distributions, that they are independent of each other, and that $\{X_t(m)\}$, $\{X_p(m)\}$ and $\{X_a(i)\}$ are each identically distributed sequences of random variables. Henceforth, we omit the arguments m and i. The statistics for the random variables L^A and M will be obtained shortly.

We are interested in the mean processing requirement per packet in order for the sender to multicast

packets reliably to all of the receivers. It is given by

$$
\begin{aligned}
E[X^A] &= E[M]E[X_p] + (E[M] - 1)E[X_t] + \\
&\quad E[L^A]E[X_a].
\end{aligned}
\tag{2}
$$

Next consider L^A. Define L_r^A to be the number of ACKs generated by receiver r. Now, $E[L_r^A] = E[M](1-p)$ is the expected number of ACKs generated by r. As the receivers are statistically identical, we have

$$
E[L^A] = RE[M](1 - p).
\tag{3}
$$

Note that we have assumed here that a receiver generates an ACK for each successfully received packet, regardless of whether it has already previously acknowledged the packet. Substituting (3) into (2) yields the following expression for $E[X^A]$,

$$
\begin{aligned}
E[X^A] &= E[M]E[X_p] + (E[M] - 1)E[X_t] + \\
&\quad RE[M](1 - p)E[X_a].
\end{aligned}
\tag{4}
$$

The only unknown in the right hand side of equations (3) and (4) is $E[M]$. Define M_r to be the number of transmissions required for receiver r to receive the packet correctly. Now, $P[M_r \leq m] = 1 - p^m$, $m = 1, \ldots$ and $E[M_r] = 1/(1 - p)$. As the events of lost packets at different receivers are independent, we have

$$
\begin{aligned}
P[M \leq m] &= \prod_{r=1}^{R} P[M_r \leq m] \\
&= (1 - p^m)^R.
\end{aligned}
\tag{5}
$$
$$
\tag{6}
$$

Therefore,

$$
\begin{aligned}
E[M] &= \sum_{m=1}^{\infty} mP[M = m] = \sum_{m=1}^{\infty} P[M \geq m] \\
&= \sum_{m=1}^{\infty} \left(1 - \left(1 - p^{(m-1)} \right)^R \right)
\end{aligned}
\tag{7}
$$
$$
\tag{8}
$$

In our numerical results in the following section, we truncated the sum in equation 7 when the mth value was less that 10^{-6}. We note that the right hand side of equation 7 can be simplified to

$$
E[M] = \sum_{i=0}^{R} \binom{R}{i} (-1)^{i+1} \frac{1}{(1 - p^i)}
$$

X_p	-	the time to process the transmission of a packet
X_a, X_n, X_t	-	the times to process an ACK, a NAK and a timeout at the sender respectively
Y_p, Y_t	-	the times to process a newly received packet or a timeout.
Y_a, Y_n	-	the times to process and transmit an ACK and a NAK from the receiver respectively
Y_n'	-	the time required to receive and process a NAK at a receiver
p	-	the probability of loss at a receiver; losses at different receivers are assumed to be independent events
L_r^w	-	the number of NAKs from receiver r, $w \in \{N1, N2\}$
L^w	-	the total number of NAKs from all receivers per packet, $w \in \{N1, N2\}$
L_r^A	-	the number of ACKs from receiver r
L^A	-	the total number of ACKs from all receivers per packet
M_r	-	the number of transmissions necessary for receiver r to successfully receive a packet
M	-	the number of transmissions for all receivers to receive the packet correctly; $M = \max_r \{M_r\}$
X^w, Y^w	-	the send and receive per packet processing times in protocol $w \in \{A, N1, N2\}$
Λ_s^w, Λ_r^w	-	the send and receive per packet processing rates for protocol $w \in \{A, N1, N2\}$
Λ_o^w, Λ_m^w	-	the throughputs for protocol $w \in \{A, N1, N2\}$ for one-many and many-many applications respectively

Table 1: Notation

but that the alternating signs in this sum are difficult to handle numerically [19].

If we let Λ_s^A denote the rate at which the sender can successfully transmit packets to all receivers then $\Lambda_s^A = 1/E[X^A]$.

In a similar fashion, the mean processing requirement at the receiver for a randomly chosen packet is

$$E[Y^A] = E[M](1-p)E[Y_p] + E[M](1-p)E[Y_a].$$

Here the first term corresponds to the mean time spent processing packets received, and the second term corresponds to the mean time spent producing and transmitting acknowledgments. In the above equation, $E[M]$ is as given in (7). The maximum packet processing rate at the receiver is $\Lambda_r^A = 1/E[Y^A]$.

It is easily shown that $E[M]$ is $O(1+p/(1-p)\ln R)$. Hence, $E[X^A]$ is $O(R(1+p/(1-p)\ln R))$ and $E[Y^A] = O(1-p+p\ln R)$. Observe that, as $p \to 0$, $E[X^A] \to O(R)$. Consequently, this protocol places a significant processing burden on the sender for large values of R even in the case of a highly reliable system.

One-many applications: In the case of a one-many application, the overall protocol throughput, Λ_o^A, for the (A) protocol is given by the minimum of the per-packet processing rates at the sender and at the

receiver

$$\Lambda_o^A = \min\{\Lambda_s^A, \Lambda_r^A\}. \tag{9}$$

Many-many applications: In the case of a many-many application in which each packet is equally likely to send a packet to the other participants, the mean packet processing time is $E[X^A]/(R+1) + RE[Y^A]/(R+1)$ and the overall protocol throughput is Λ_m^A, is

$$\Lambda_m^A = \frac{R+1}{E[X^A] + RE[Y^A]}. \tag{10}$$

4   Throughput Analysis of Receiver-Initiated Protocols

We now analyze the receiver-initiated protocol (N1) first by considering the sender. Let X^{N1} denote the packet processing requirement at the sender under (N1). This processing time can be expressed as

$$X^{N1} = \sum_{m=1}^{M} X_p(m) + \sum_{i=1}^{L^{N1}} X_n(i) \tag{11}$$

where the first term corresponds to the processing time associated with the M different transmissions of the packet and the second term corresponds to the processing time for the NAKs that are transmitted from the receivers to the sender. As before, we make no assumptions regarding the distributions of these random variables except that they are independent of each other and that $\{X_p(m)\}$ and $\{X_n(i)\}$ are sequences of identically distributed random variables. As before, M is the number of transmissions required and L^{N1} is the number of NAKs received.

The mean processing time is given as

$$E[X^{N1}] = E[M]E[X_p] + E[L^{N1}]E[X_n].$$

Now, the number of NAKs returned to the sender by receiver r is $M_r - 1$ with mean $p/(1-p)$. Hence the mean number of NAKs returned by all receivers is $E[L^{N1}] = Rp/(1-p)$, the mean per packet processing time at the sender is

$$E[X^{N1}] = E[M]E[X_p] + RpE[X_n]/(1-p) \tag{12}$$

and the sender packet processing rate is $\Lambda_s^{N1} = 1/E[X^{N1}]$.

We focus next on the mean per packet processing time at a receiver. Similar to the analysis leading to (2), we have

$$\begin{aligned}
E[Y^{N1}] &= E[M](1-p)E[Y_p] \\
&\quad + P[M_r > 1](E[M_r|M_r > 1] - 1)E[Y_n] \\
&\quad + P[M_r > 2](E[M_r|M_r > 2] - 2)E[Y_t].
\end{aligned}$$

9

Here the first term corresponds to the processing required to receive a packet. The second term corresponds to the processing required to prepare and return NAKs. Note that this only occurs each time the receiver determines the packet to be lost *prior to the first correct receipt of this packet.* The last term corresponds to processing of the timer when it expires. Again, this is only required after the first transmission (if lost) up to, but not including, the first correct reception of a given packet.

From the distribution of M_r, it follows that

$$E[M_r|M_r > 1] = (2 - p)/(1 - p),$$
$$E[M_r|M_r > 2] = (3 - 2p)/(1 - p).$$

Substituting into our equation above for $E[Y^{N1}]$ gives:

$$E[Y^{N1}] = E[M](1 - p)E[Y_p]$$
$$+ pE[Y_n]/(1 - p) + p^2 E[Y_t]/(1 - p)$$

and the receiver packet processing rate is $\Lambda_r^{N1} = 1/E[Y^{N1}]$.

For this protocol we have $E[X^{N1}] = O(1 + pR/(1 - p))$ and $E[Y^{N1}] = O(1 - p + p \ln R)$. Consequently, it is better suited to large scale multicasts than the generic ACK-based protocol (A). Observe that $E[X^{N1}] = O(1)$ and $E[Y^{N1}] = O(1)$ when $p \to 0$.

We end this section with the analysis of the receiver-initiated protocol (N2). Protocol (N2) differs from (N1) in two ways. First, NAKs are multicast from a receiver to all other receivers as well as to the sender. If a receiver receives a NAK for a packet that it has not received correctly prior to sending its own NAK, then it need not return a NAK. The addition of a random delay at a receiver prior to returning a NAK ensures that *most of the time* only one NAK will be generated among all of the receivers and that it will act as a signal to the remaining receivers to not return a NAK. How well this mechanism works to limit the number of NAKs per transmission depends on the network topology and the choice of the random delay distribution. As we are solely concerned with throughput, we will assume that the delays are sufficiently long that the event of two or more NAKs being generated by a transmission is sufficiently small to be ignored.

With these modifications, it is easy to see that

$$1/\Lambda_s^{N2} = E[X^{N2}] = E[M]E[X_p] + (E[M] - 1)E[X_n] \tag{13}$$

and

$$1/\Lambda_r^{N2} = E[Y^{N2}] = E[M](1 - p)E[Y_p]$$
$$+ (E[M] - 1)(E[Y_n]/R + (R - 1)E[Y_n']/R)$$
$$+ P[M_r > 2](E[M_r|M_r > 2] - 2)E[Y_t]$$

Note that $E[X^{N2}] = O(1 + p \ln R/(1 - p))$ and $E[Y^{N2}] = O(1 - p + p \ln R)$. Hence this protocol shows the best potential for handling multicasts to large receiver groups. Furthermore, $E[X^{N2}] = E[Y^{N2}] = O(1)$ when $p \to 0$.

One-many applications: In the case of a one-many application, the overall protocol throughputs for the two receiver oriented protocols are

$$\Lambda_o^{N1} = \min\{\Lambda_s^{N1}, \Lambda_r^{N1}\}, \tag{14}$$

$$\Lambda_o^{N2} = \min\{\Lambda_s^{N2}, \Lambda_r^{N2}\}. \tag{15}$$

Many-many applications: In the case of a many-many application in which each packet is equally likely to send a packet to the other participants, the overall protocol throughputs for the two receiver oriented protocols are

$$\Lambda_m^{N1} = \frac{R+1}{E[X^{N1}] + RE[Y^{N1}]} \tag{16}$$

$$\Lambda_m^{N2} = \frac{R+1}{E[X^{N2}] + RE[Y^{N2}]} \tag{17}$$

5   Numerical Results

We now examine the relative performance of protocols (A), (N1), and (N2).

In order to provide concrete values for the amounts of processing time needed to send/receive a data, ACK, or NAK packet and handle a timeout, we use the measurements reported by Kay and Pasquale [14, 15] . These measurements were taken on an in-kernel implementation of the UDP/IP/FDDI protocol stack in DEC's ULTRIX 4.2a operating system running on a DECstation 5000/200. Although a newer architecture would undoubtedly result in shorter processing times, our focus will be primarily on the relative (rather than absolute) throughput of the sender and receiver and the relative performance of (N1), (N2) and (A). We expect the relative performance to be relatively insensitive to changes in processing capabilities as long as the relative costs of the various processing steps remains the same. In our numerical examples we use $E[X_p] = E[Y_p] = 1000$ μsecs for the processing time needed to send or receive a 2K data packet and $E[X_a] = E[X_n] = E[Y_a] = E[Y_n] = E[Y_n'] = 500$ μsecs as the processing time to send or receive a small ACK/NAK packet (from Figures 6 and 7 in [14]). We use $E[X_t] = E[Y_t] = 24$ μsecs (from Figure 10b in [15]) for the timer overhead. We examine performance for loss probabilities in the range 0.01–0.25 as they typify the current loss characteristics of the MBone, [27].

With these values and the throughput characterizations from the previous section, we can now compare the performance of the three generic protocols. Let us begin by comparing the achievable packet send and receive processing rates under the three protocols.

Figures 1, 2 and 3 show the absolute values of send and receive processing rates for (A), (N1) and (N2) for $p = 0.05$ and $p = 0.25$. Figure 1 shows that the sender is the bottleneck under protocol (A) for both values of p. As the number of receivers increases, the send rate decreases because of the need to process more acknowledgments. The receive processing rate is higher than the send processing rate, but also degrades as the number of receivers increases. This is because, with increasing R, there
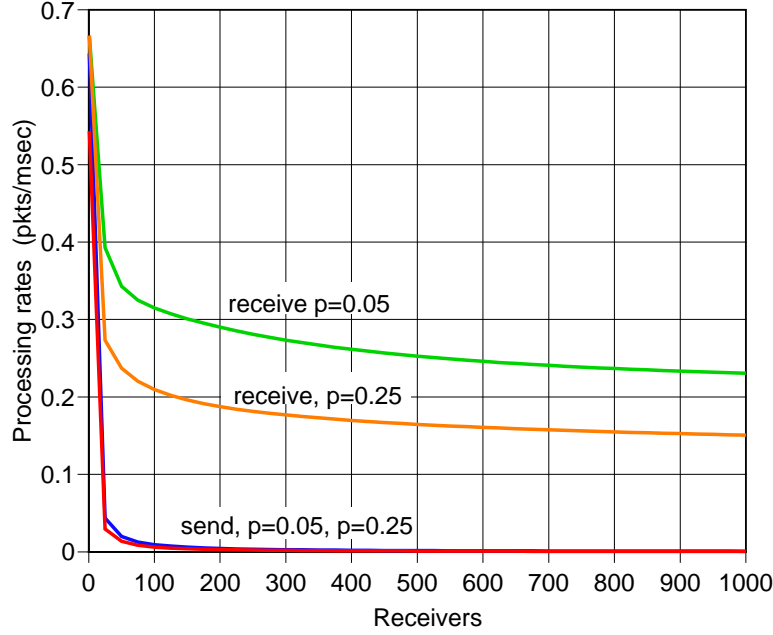
11

Figure 1: Sender and receiver processing rates under A

is an increasing number of retransmissions from the sender, as evidenced in equation 7. Figures 2 and 3 illustrate that the send rates remains smaller under protocols (N1) and (N2), although the differences in send and receive rates are much less pronounced under (N2) than under (N1) or (A). The more closely matched send and receive rates under (N2) suggest that this protocol provides a more balanced distribution of the processing burden between the sender and the receivers.

Having now examined the send and receive rates for each of the three protocols, we can now compare their relative performance for the two classes of applications.

5.1 One-many Applications

In the case of one-many applications, the sender and receiver processing rates reported in the previous section indicate that the throughput of each of the three generic protocols is determined by sender behavior (for the loss probabilities of interest to us),

$$\Lambda_o^w = \lambda_s^w, \quad , w \in \{A, N1, N2\}$$

Figure 4 plots the ratio of the protocol throughputs of (N1) and (A) versus R for various loss probabilities, for the case of one-many applications. Note that the (N1) protocol always achieves a higher throughput than the (A) protocol. As expected, when the loss probability is low (say 0.01), the difference in relative performance of (N1) over (A) is greater than when the loss probability is high (say 0.25).
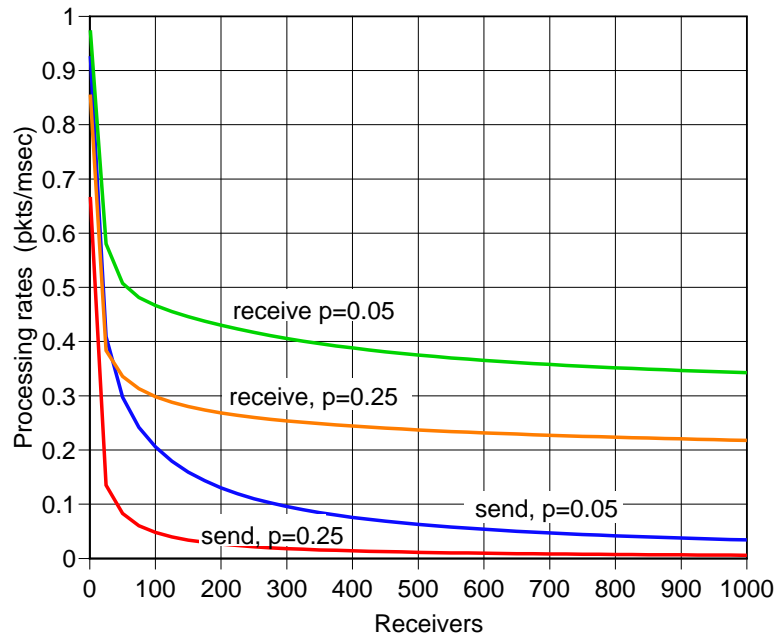
12

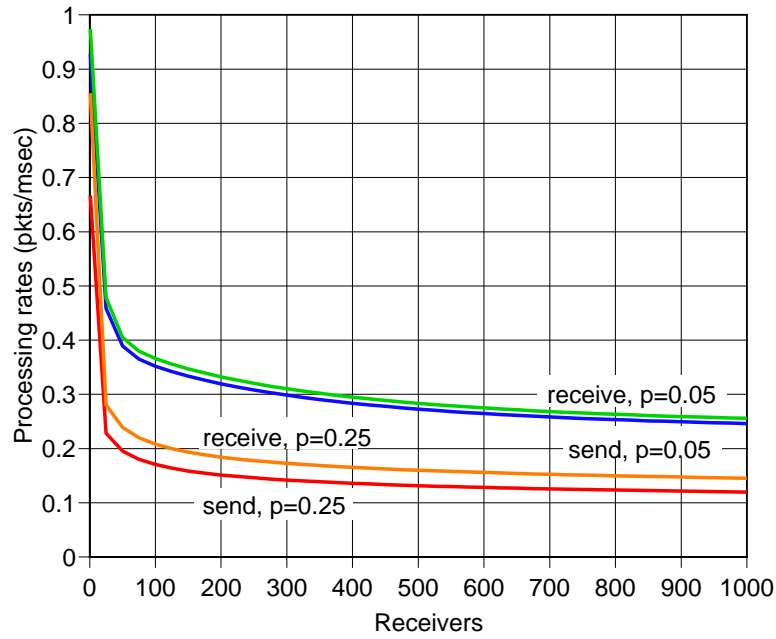Figure 2: Sender and receiver processing rates under N1

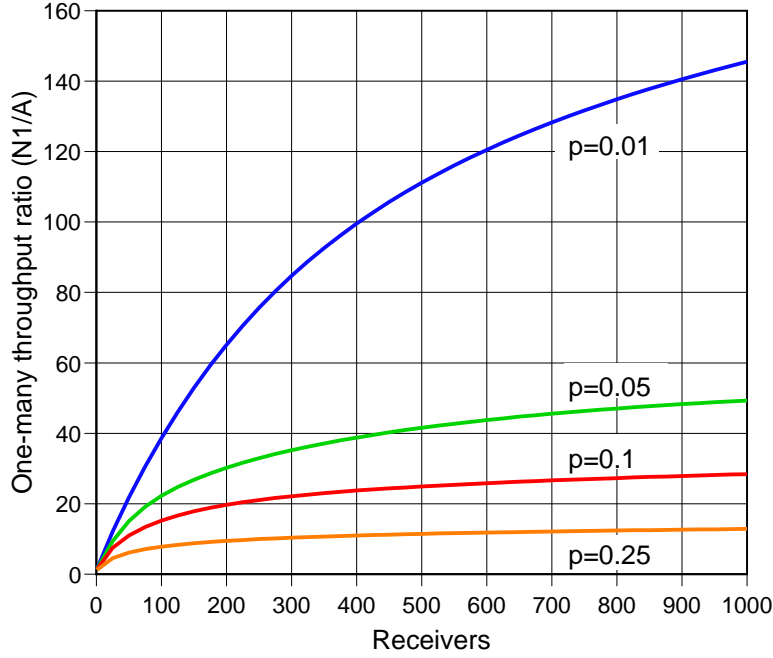Figure 3: Sender and receiver processing rates under N2

13

Figure 4: Ratio of one-many application throughputs for (N1) and (A): $\Lambda_o^{N1}/\Lambda_o^A$ vs R

This is because a receiver-initiated scheme places a very light burden on the sender for low loss probabilities while a sender-initiated scheme generates many acknowledgments that need to be processed at the sender. For a given loss probability, the relative performance of (N1) improves over that of (A) with increasing R. This improvement is logarithmic as indicated in the complexity results presented in the previous section.

Figure 5 shows the ratio of the throughputs of one-many applications under (N2) and (N1). Note that multicasting NAKs yields additional increases in throughput, especially as the number of receivers increases and for environments characterized by high loss rates.

5.2   Many-many Applications

In this section we perform a similar investigation of the behavior of the three generic protocols for the case of many-many applications. We begin by focusing on the effects of shifting responsibility for reliable data transfer from the participant sending the packet to those receiving the packet.

Figure 6 plots the ratio of the many-many application throughputs under (N1) and (A). Although we observe a nearly a twofold performance improvement when the receive functionality takes greater responsibility for reliable data transfer, this increase in throughput is significantly less than what was observed for one-many applications. The reason for this reduced improvement is that a participant in the application is most likely to perform receive-side processing on a packet (with probability $R/(R+1)$)
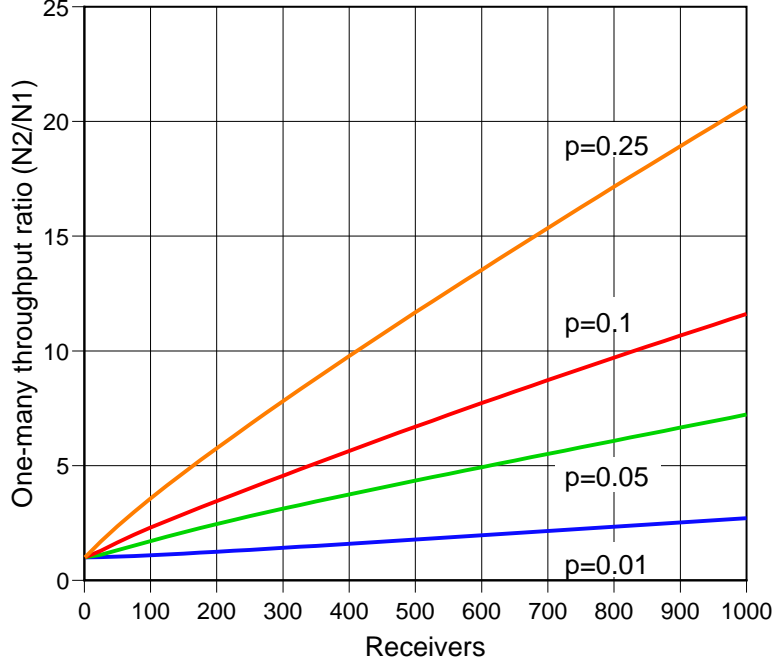
Figure 5: Ratio of one-many application throughputs for (N2) and (N1): $\Lambda_o^{N2}/\Lambda_o^{N1}$ vs R

rather than send-side processing (with probability $1/(R+1)$). The relative receiver processing rates of (N1) and (A), illustrated in Figure 7, show less than a 50% difference between them. This explains the behavior of $\Lambda_m^{N1}/\Lambda_m^{A}$ in Figure 6.

The additional effect of replacing the point-point transfer of NAKs with the multicasting of NAKs as part of the receive side functionality is illustrated in Figure 8, where the ratio of the throughputs of (N2) and (N1) are plotted for many-many applications. We observe that, unlike with one-many applications, multicasting NAKs *reduces* the throughput of a receiver-initiated protocol. This is because multicasting NAKs adds additional complexity to the receiver functionality which, as noted above, dominates the performance of reliable data transfer in a many-many application. The relative behavior of the receive side processing rates of (N2) and (N1) is illustrated in Figure 9.

Last, Figure 10 illustrates the ratio of the throughputs of many-many applications under (N2) and (A). We observe that, in spite of the fact that multicasting NAKs performs poorly compared to point-point transfer of NAKs, (N2) still exhibits a slight performance improvement over the sender-initiated protocol (A).
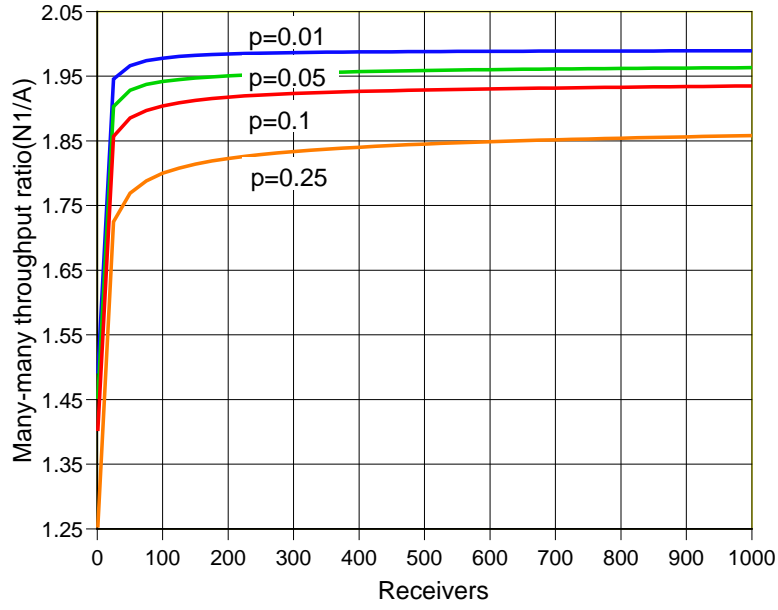
15

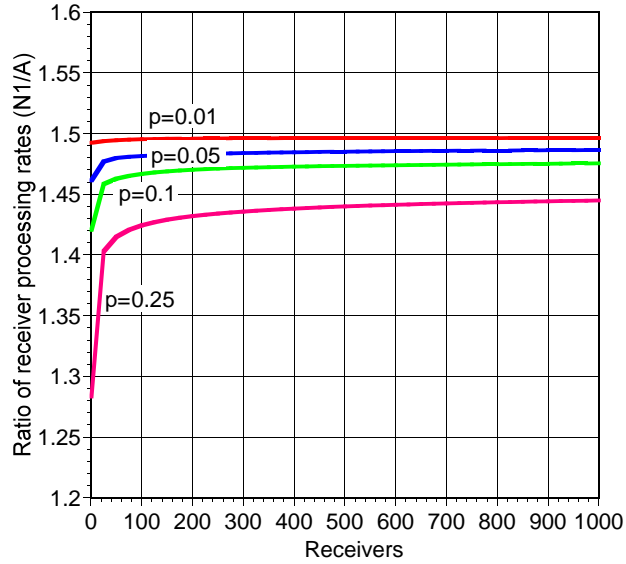Figure 6: Ratio of many-many throughputs for (N1) and (A): $\Lambda_m^{N1}/\Lambda_m^A$ vs R

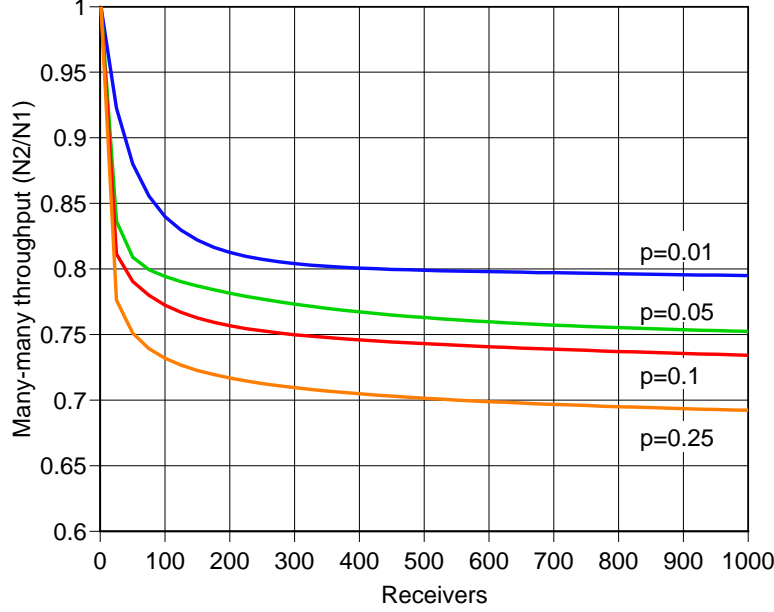Figure 7: Ratio of receive rates for (N1) and (A): $\Lambda_r^{N1}/\Lambda_r^A$ vs R

Figure 8: Ratio of many-many throughputs for (N2) and (N1): $\Lambda_m^{N2}/\Lambda_m^{N1}$ vs R

6 Summary and Future Work

In this paper, we have examined the problem of providing reliable multicast communication in large scale networks, when possibly thousands of hosts may participate in a multicast group. We studied three generic error control protocols, one that was sender-initiated (ACK-based) and two that were receiver-initiated (NAK-based). An important feature of our analysis of these protocols is that we focused on *host processing* as the constrained resource; previous analyses of reliable multicast protocols have focussed on network bandwidth as the scarce resource - a resource that we believe is becoming of less importance as network bandwidths continue to increase and multicasting emerges as an important communication paradigm. Our analyses provide a quantitative demonstration of the superiority of receiver-initiated approaches over sender-initiated approaches. We found, however, that among the two receiver initiated multicasting protocols we considered, that the (N2) protocol was preferred in the one-many scenario, while the (N1) protocol was superior in the many-to-many case. This illustrates the important point that the relative performance of different reliable multicast protocols may depend on the manner in which hosts act as senders and receivers.

Our present work can be extended in a number of ways. In our analyses, we assume independent loss events at the various receivers. While we expect this assumption to give us pessimistic bounds on the throughputs, whether relaxing this assumption affects the throughput expressions for the three protocols studied in different ways is a question open to investigation. Another open issue is that of protocol delay. So far, we have restricted our attention to throughput performance of the three protocols. We conjecture that for high loads and large numbers of receivers, the relative delay performance of the three protocols will be similar to the relative throughput performance. It is likely that for low numbers of receivers the delay performance of (N1) will be better than that for (N2), but this is a matter that remains to be
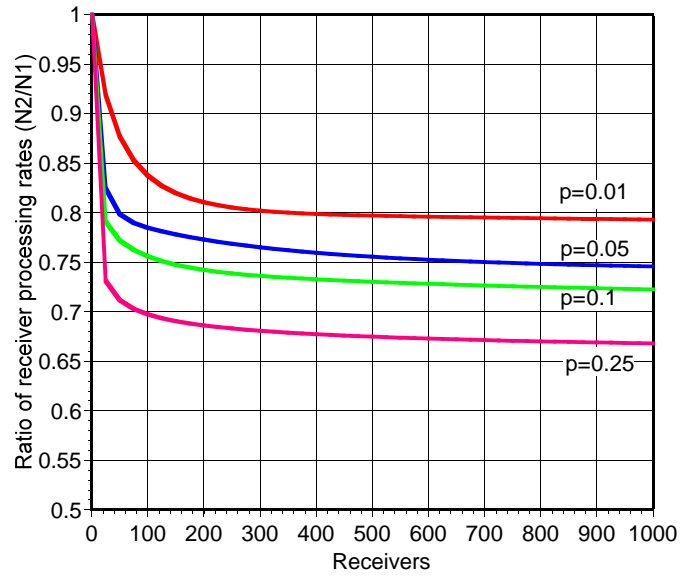
17

Figure 9: Ratio of receive rates for (N2) and (N1): $\Lambda_r^{N2}/\Lambda_r^{N1}$ vs R
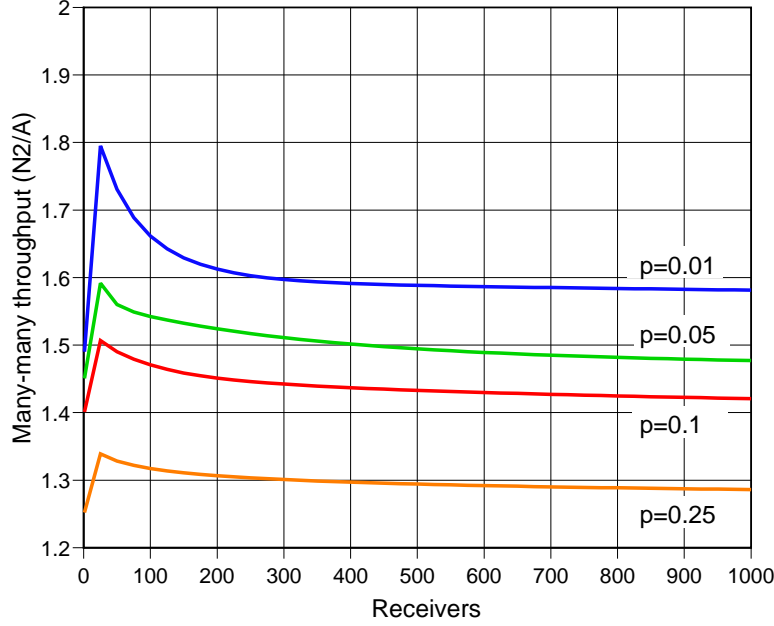
Figure 10: Ratio of many-many throughputs for (N2) and (A): $\Lambda_m^{N2}/\Lambda_m^A$ vs R

investigated in a precise, quantitative way. Finally, we note that the performance comparison of the various optimzations that are possible for protocls (A), (N1) and (N2) remains an open an interesting question.

Acknowledgments The authors wish to thank the anonymous referees, who provided many comments that were valuable in revising an earlier version of this paper.

References

[1] P. Bhagwat, P. Misra, S. Tripathi, "Effect of Topology on Performance of Reliable Multicast Communication," *Proc. IEEE Infocom 94,* (Toronto, June 1994), pp. 602 – 609.

[2] R. Braudes, S. Zabele, "Requirements for Multicast Protocols," RFC 1458, May 1993.

[3] S. Casner, "First IETF Internet Audiocast", *ACM SIGCOMM Computer Communication Review*, 22:92–97, July 1992.

[4] S.E. Deering and D.R. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs", *ACM Trans. on Computer Systems*, 8:85–110, May 1990.

[5] A. Erramilli and R.P. Singh, "A Reliable and Efficient Multicast Protocols for Broadband Broadcast Networks," *Proc. ACM Sigcomm88,* pp. 343-352, August 1988.

[6] J.D. Esary, F. Proschan, D.W. Walkup, "Association of random variables with applications", *Ann. Math. Statis.*, **38**, pp. 1466-1474, 1967.

[7] S. Floyd, V. Jacobson, S. McCanne, C.G. Liu, L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application-Level Framing," *Proc. ACM Sigcomm95,* (Aug. 1995, Boston MA), pp 342 - 356.

[8] R. Frederick, "Network Video: nv", Xerox Palo Alto Research Center, on-line software: ftp://ftp.parc.xerox.com/net-research.

[9] I. Gopal and J. Jaffe, "Point-to-Multipoint Communication over Broadcast Links", *IEEE Trans. on Communications,* 32:1034–1044, September 1984.

[10] Instiute for Simulation and Training, "Standard for Distributed Interactive Simulation - Application Protocols," Technical Report IST-CR-94-50, University of Central Florida, Orlando, Fla, 1994.

[11] V. Jacobson, 1993 ARPA Networking PI Meeting, Sept. 1993.

[12] V. Jacobson and S. McCanne, "Visual Audio Tool: vat", Lawrence Berkeley Laboratory, on-line software: ftp://ftp.ee.lbl.gov/conferencing/vat.

[13] V. Jacobson and S. McCanne, "LBL Whiteboard: wb", Lawrence Berkeley Laboratory, on-line software: ftp://ftp.ee.lbl.gov/conferencing/wb.

[14] J. Kay and J. Pasquale, "Measurement, Analysis, and Improvement of UDP/IP Throughput for the DECstation 5000," *Proc. 1993 Winter Usenix Conference,* pp. 249–258, January 1993.

[15] J. Kay and J. Pasquale, The Importance of Non-Data Touching Processing Overheads in TCP/IP," *Proc. ACM Sigcomm 93,* pp. 259–268, August 1993.

[16] S. McCanne and V. Jacobson, "vic: a Flexible Framework for Packet Video," *Proc. ACM Multimedia'95,* (Nov. 1995, San Francisco CA), pp. 511-522

[17] L. Padmanabhan, "Design and Implementation of a Shared White-Board", M.S. Project, Dept. of Computer Science, UMass, Amherst, MA 01003, May 1993.

[18] S. Paul, K. Sabnani, D. Kristol, Multicast Transport Protocols for High-Speed Networks," *Proc. IEEE Int. Conference on Network Protocols,* (Boston MA, 1994).

[19] S. Pingali, J.F. Kurose, D. Towsley, "A Comparison of Sender-initiated and Receiver-initiated Reliable Multicast Protocols," *Proc. 1994 ACM Sigmetrics Conf.,* May 1994.

[20] S. Ramakrishnan and B. N. Jain, "A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LANs," *Proc. IEEE Infocom'87*, pp 502–511, Mar-Apr 1987.

[21] S. Ram Chandran and S. Lin, "Selective-repeat-ARQ Schemes for Broadcast Links," *IEEE Trans. on Communications,* 40:12–19, January 1992.

[22] K. Sabnani and M. Schwartz, "Multidestination Protocols for Satellite Broadcast Channels," *IEEE Trans. on Communications,* 33:232–240, March 1985.

[23] H. Schulzrinne, "Voice Communication Across the Internet: A Network Voice Terminal", Reseach Report TR 92-50, Dept. of Computer Science, Univ. of Massachusetts, Amherst MA, July 1992.

[24] D. Towsley, "An Analysis of a Point-to-Multipoint Channel Using a Go-Back-N Error Control Protocol", *IEEE Trans. on Communications*, 33:282–285, March 1985.

[25] D. Towsley and S. Mithal, "A Selective Repeat ARQ Protocol for a Point to Multipoint Channel," *Proc. IEEE Infocom'87*, pp 521–526, Mar-Apr 1987.

[26] J.L. Wang and J.A. Silvester, "Delay Minimization of the Adaptive Go-Back-N ARQ Protocols for Point-to-Multipoint Communication", *Proc. IEEE Infocom'89*, pp 584–593, April 1989.

[27] M. Yajnik, J. Kurose, D. Towsley, "Packet Loss Correlation in the MBone Multicast Network," to appear in *IEEE Global Internet Conference,* Dec. 1996. ftp://gaia.cs.umass.edu/pub/Yajn96:Loss.ps.Z.

[28] R. Yavatkar and L. Manoj, "Optimistic Approaches to Large-Scale Dissemination of Multimedia Information", *Proc. ACM Multimedia '93*, August 1993.

[29] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, "RSVP: A New Resource ReSerVation Protocol", *IEEE Network*, pp 8–18, September 1993.