

A Comparison of Known Classes of Reliable Multicast Protocols

Brian Neil Levine and JJ Garcia-Luna-Aceves
Computer Engineering Department
University of California, Santa Cruz, CA 95064, USA
{brian,jj}@cse.ucsc.edu

Abstract

We analyze the maximum throughput that the known classes of reliable multicast protocols can attain. A new taxonomy of reliable multicast protocols is introduced based on the premise that the mechanisms used to release data at the source after correct delivery should be decoupled from the mechanisms used to pace the transmission of data and to effect error recovery. Receiver-initiated protocols, which are based entirely on negative acknowledgments (NAKs) sent from the receivers to the sender, have been proposed to avoid the implosion of acknowledgments (ACKs) to the source. However, these protocols are shown to require infinite buffers in order to prevent deadlocks. Two other solutions to the ACK-implosion problem are tree-based protocols and ring-based protocols. The first organize the receivers in a tree and send ACKs along the tree; the latter send ACKs to the sender along a ring of receivers. These two classes of protocols are shown to operate correctly with finite buffers. Following our taxonomy, the maximum attainable throughput by the known classes of reliable multicast protocols is analyzed. It is shown that tree-based protocols constitute the most scalable class of all reliable multicast protocols proposed to date.

1. Introduction

The increasing popularity of real-time applications supporting either group collaboration or the reliable dissemination of multimedia information over the Internet is making the provision of reliable and unreliable end-to-end multicast services an integral part of its architecture. Although reliable broadcast protocols have existed for quite some time (e.g., see [3]), viable approaches on the provision of reliable multicasting over the Internet are just emerging. The reliable multicast problem facing the future Internet is compounded by its current size and continuing growth, which makes the handling of acknowledgments a major challenge commonly referred to as the *acknowledgment (ACK) implosion problem*.

The two most popular approaches to reliable multicasting proposed to date are called *sender-initiated* and *receiver-initiated*. In the sender-initiated approach, the sender maintains the state of all the receivers to whom it has to send information and from whom it has to receive ACKs. Each sender's transmission or retransmission is multicast to all receivers; for each packet that each receiver obtains correctly, it sends a unicast ACK to the sender. In contrast, in the receiver-initiated approach, each receiver informs the sender of the information that is in error or missing; the sender multicasts all packets, giving priority to retransmissions, and a receiver sends a negative acknowledgment (NAK) when it detects an error or a lost packet.

The first comparative analysis of sender-initiated and receiver-initiated reliable multicast protocols was presented by Pingali *et al.* [15, 16]. This analysis showed that receiver-initiated protocols are far more scalable than sender-initiated protocols, because the maximum throughput of sender-initiated protocols is dependent on the number of receivers, while the maximum throughput of receiver-initiated protocols is independent of the number of receivers (when the probability of packet loss is negligible). However, as this paper demonstrates, the receiver-initiated protocols proposed to date cannot prevent deadlocks when they operate with finite memory.

This paper addresses the question of whether a reliable multicast protocol can be designed that enjoys all the scaling properties of receiver-initiated protocols, while still being able to operate correctly with finite memory. To address this question, the previous analysis by Pingali *et al.* [15, 16] is extended to consider the maximum throughput of generic ring-based protocols, and two classes of tree-based protocols. These classes are the other three known approaches that can be used to solve the ACK-implosion problem. Our analysis shows that tree- and ring-based protocols can work correctly with finite memory, that both are scalable, and that tree-based protocols are the best choice in terms of processing and memory requirements.

The results presented in this paper are theoretical in nature and apply to generic protocols, rather than to specific implementations; however, we believe that they provide valuable architectural insight for the design of future reliable multicast protocols. Section 2 presents a new taxonomy of reliable multicast protocols that organizes known approaches into four protocol classes and discusses how many key papers in the literature fit within this taxonomy. This taxonomy is based on the premise that the analysis of the mechanisms used to release data from memory after their correct reception by all receivers can be decoupled from the study of the mechanisms used to pace the transmission of data within the session and the detection of transmission errors. Using this taxonomy, we argue that all reliable unicast and multicast protocols proposed to date that use NAKs and work correctly with finite memory use ACKs to release memory and NAKs to improve throughput. Section 3 addresses the correctness of the various classes of reliable multicast protocols introduced in our taxonomy, showing that the type of receiver-initiated protocols proposed to date require infinite memory. Section 4 extends the analysis by Pingali *et al.* [15, 16] by analyzing the maximum throughput of three protocol classes: tree-based, tree-based with local NAK-avoidance and periodic polling (tree-NAPP), and ring-based protocols. Although the maximum throughput of receiver-initiated, tree-based¹, and ring-based protocols are all independent of the number of receivers as the probability of error goes to zero, we show that only tree-based protocols can be completely scalable under any condition, i.e., when the probabil-

This work was supported in part by the Office of Naval Research under Grant N00014-94-1-0688.

¹To avoid confusion comments on “receiver-initiated” and “tree-based” protocols are inclusive of all respective subclasses.

ity of error is non-negligible. Section 5 provides numerical results on the performance of the protocol classes under different scenarios, and discusses the implications of our results in light of recent work on reliable multicasting. Section 6 provides concluding remarks.

2. A New Taxonomy of Reliable Multicast Protocols

We now describe the four generic approaches known to date for reliable multicasting. Well-known protocols (for unicast and multicast purposes) are mapped into each class. Our taxonomy differs from prior work [15, 16, 8] addressing receiver-initiated strategies for reliable multicasting in that we decouple the definition of the mechanisms needed for pacing of data transmission from the mechanisms needed for the allocation of memory at the source. Using this approach, the protocol can be thought as using two windows: a congestion window (cw) that advances based on feedback from receivers regarding the pacing of transmissions and detection of errors, and a memory allocation window (mw) that advances based on feedback from receivers as to whether the sender can erase data from memory. In practice, protocols may use a single window for pacing and memory (e.g., TCP [10]) or separate windows (e.g., NETBLT [4]). It will become apparent that this decoupling is critical in obtaining an accurate understanding of why reliable unicast and multicasting protocols scale and work correctly with finite memory.

Each reliable protocol assumes the existence of multicast routing tree(s) that are provided by underlying multicast routing protocols. In the internet, these trees will be built using such protocols as DVMRP [6], Core Based Trees (CBT) [1] or Protocol Independent Multicast (PIM) [7].

2.1. Sender-Initiated Protocols

In the past [15, 16], sender-initiated protocols have been characterized as placing the responsibility of reliable delivery at the sender. However, this characterization is overly restrictive and does not reflect the way in which several reliable multicast protocols that rely on positive acknowledgments from the receivers to the source have been designed. In our taxonomy, a sender-initiated reliable multicast protocol is one that requires the source to receive ACKs from all the receivers, before it is allowed to release memory for the data associated with the ACKs. It is clear that the source is required to know the constituency of the receiver set, and that the scheme suffers from the ACK-implosion problem. However, this characterization leaves unspecified the mechanism used for pacing of transmissions and for the detection of transmission errors. Either the source or the receivers can be in charge of the retransmission timeouts!

The traditional approach to pacing and transmission error detection (e.g., TCP in the context of reliable unicast) is for the source to be in charge of the retransmission timeout. However, as suggested by the results reported by Floyd *et al.* [8], a better approach for pacing a multicast session is for each receiver to set its own timeout. A receiver sends ACKs to the source at a rate that it can accept, and sends a NAK to the source after not receiving a correct packet from the source for an amount of time that exceeds its retransmission timeout. An ACK can refer to a specific packet or a window of packets, depending on the specific retransmission strategy.

Notice that, regardless of whether a sender-based or receiver-based retransmission strategy is used, the source is still in charge of deallocating memory after receiving all the ACKs for a given packet or set of packets. The source keeps packets in memory until every receiver node has positively acknowledged receipt of the

data. If a sender-based retransmission strategy is used, the sender “polls” the receivers for ACKs by retransmitting after a timeout. If a receiver-based retransmission strategy is used, the receivers “poll” the source (with an ACK) after they time out².

It is important to note that, just because a reliable multicast protocol uses NAKs, it does not mean that it is receiver-initiated, i.e., that NAKs are the basis for the source to ascertain when it can release data from memory. The combination of ACKs and NAKs has been used extensively in the past for reliable unicast and multicast protocols. For example, NETBLT is a unicast protocol that uses a NAK scheme for retransmission, but only on small partitions of the data (i.e., its cw). In between the partitions, called “buffers” are ACKs for all the data in the buffer (i.e., the mw). Only upon receipt of this ACK does the source release data from memory; therefore, NETBLT is really sender-initiated. In fact, NAKs are unnecessary in NETBLT for its correctness, i.e., a buffer can be considered one large packet that eventually must be ACKed, and are important only as a mechanism to improve throughput by allowing the source to know sooner when it should retransmit some data.

A protocol similar to NETBLT is the “Negative Acknowledgments with Periodic Polling” (NAPP) protocol [17]. This protocol is a broadcast protocol for LANs. Like NETBLT, NAPP groups together large partitions of the data that are periodically ACKed, while lost packets within the partition are NAKed. NAPP advances the cw by NAKs and periodically advances the mw by ACKs. Because the use of NAKs can cause a NAK-implosion at the source, NAPP uses a NAK-avoidance scheme. As in NETBLT, NAKs increase NAPP’s throughput, but are not necessary for its correct operation, albeit slow. The use of periodic polling limits NAPP to LANs, because the source can still suffer from an ACK-implosion problem even if ACKs occur less often.

Other sender-initiated protocols, like the Xpress Transfer Protocol (XTP) [18], were created for use on an internet, but still suffer from the ACK-implosion problem.

The main limitation of sender-initiated protocols is not that ACKs are used, but the need for the source to process all of the ACKs and to know the receiver set. The two known methods that address this limitation are: (a) using NAKs instead of ACKs, and (b) delegating retransmission responsibility to members of the receiver set by organizing the receivers into a ring or a tree. We discuss both approaches subsequently.

2.2. Receiver-Initiated Protocols

Previous work [15, 16] characterizes receiver-initiated protocols as placing the responsibility for ensuring reliable packet delivery at each receiver. The critical aspect of these protocols for our taxonomy is that no ACKs are used. The receivers send NAKs back to the source when a retransmission is needed, detected by either an error, a skip in the sequence numbers used, or a timeout. Because the source receives feedback from receivers only when packets are lost and not when they are delivered, the source is unable to ascertain when it can safely release data from memory. There is no explicit mechanism in a receiver-initiated protocol for the source to release data from memory (i.e., advance the mw), even though its pacing and retransmission mechanisms are scalable and efficient (i.e., advancing the cw).

2.3. Receiver-Initiated Protocols with NAK-avoidance

Because receivers communicate NAKs back to the source, receiver-initiated protocols have the possibility of experiencing a NAK-implosion problem at the source if many receivers detect

²Of course, the source still needs a timer to ascertain when its connection with a receiver has failed.

transmission errors. To remedy this problem, previous work on receiver-initiated protocols [15, 16, 8] adopts the NAK-avoidance scheme first proposed for NAPP, which is a sender-initiated protocol. Receiver-initiated with NAK-avoidance (RINA) protocols have been shown [15, 16] to have improved the performance over the basic receiver-initiated protocol. The resulting generic RINA protocol is as follows [15, 16]: The sender multicasts all packets and state information, giving priority to retransmissions. Whenever a receiver detects a packet loss, it waits for a random time period and then multicasts a NAK to the sender and all other receivers. When a receiver obtains a NAK for a packet that it has not received and for which it has started a timer to send a NAK, the receiver sets a timer and behaves as if it had sent a NAK. The expiration of a timer without the reception of the corresponding packet is the signal used to detect a lost packet. With this scheme, it is hoped that only one NAK is sent back to the source for a lost transmission for an entire receiver set. Nodes farther away from the source might not even get a chance to request a retransmission. The generic protocol does not describe how timers are set accurately; in this paper, we assume perfect setting of timers because we are only interested in the maximum attainable throughput of protocols.

The generic RINA protocol we have just described constitutes the basis for the operation of the scalable reliable multicasting (SRM) algorithm [8]. SRM has been successfully embedded into an internet collaborative whiteboard application called *wb*. SRM sets timers based on low-rate, periodic, “session-messages” multicast by every receiver. The messages specify the highest sequence number accepted from the source³ and a time-stamp used by the receivers to estimate the delay from the source. The average bandwidth consumed by session messages is kept small (e.g., by keeping the frequency of session messages low). SRM’s implementation requires that every node stores all packets (a scheme could be used to support a “distributed memory”) or that the application layer store all relevant data. However, it is clear that the sequence number in a session message is an ACK to the last packet from the source, and that a receiver can keep “polling” the source periodically to ensure that the source eventually delivers missing packets not caught by the NAK scheme. Here again, NAKs are used to advance the *cw*, which is controlled by the receivers and session messages would be used to advance the *mw*. We do not place SRM within our sender-initiated category, because in it does not take advantage of the ACKs implicit in the session messages. In practice, the persistence of session messages forces the source to know the receiver set over time. Accordingly, as defined, SRM does not scale because it defeats one of the goals of the receiver-initiated paradigm, i.e., to keep the receiver set anonymous from the source for scaling purposes.

There are other issues that limit the use of defined RINA protocols such as SRM for reliable multicasting. First, SRM requires that data needed for retransmission be rebuilt from the application. SRM’s approach is reasonable only for applications in which the immediate state of the data is exclusively desired, which is the case of a distributed whiteboard. However, the approach does not apply for multimedia applications that have no current state, but only a stream of transition states.

Second, NAKs and retransmissions must be multicast to the entire multicast group to allow suppression of NAKs. The NAK-avoidance was designed for a limited scope, such as a LAN, or a small number of Internet nodes (as it is used in tree-NAPP protocols). This is because the basic NAK-avoidance algorithm requires that timers be set based on updates multicast by every node. As the number of nodes increases, each node must do increasing amount of work! Even worse, nodes that are on congested links, LANs or regions may constantly bother the rest of the multicast group by multicasting NAKs (often referred to as the “crying baby” problem).

³Multiple sources are supported in SRM, we focus on the single-source case for simplicity.

Another example of a receiver-initiated protocol is the “log-based receiver-reliable multicast” (LBRM) [9], which uses a hierarchy of log servers that store information indefinitely and receivers recover by contacting a log server. Using log servers is feasible only for applications that can afford the servers and leaves many issues unresolved. If a single server is used, performance can degrade due to the load at the server; if multiple servers are used, mechanisms must still be implemented to ensure that such servers have consistent information.

The following two classes organize the receiver set in ways that permit the strengths of receiver-initiated protocols to be applied on a local scale, while providing explicit mechanisms for the source to release memory safely (i.e., efficient management of the *mw*).

2.4. Tree-Based Protocols

Tree-based protocols are characterized by dividing the receiver set into groups, distributing retransmission responsibility over an acknowledgment tree (ACK tree) structure. Without loss of generality, our generic protocol definition assumes that each group consists of no more than B children and a group leader. Children are likely to be the group leaders of a subgroup. Acknowledgments from children in a group, including the sources own group, are sent only to the leader. A child in a group sends its acknowledgment to its parent as soon as it receives a correct packet, not when all its own children (if any) have sent their acknowledgments. Clearly, these acknowledgments differ from ACKs or NAKs used in sender- and receiver-initiated protocols and we refer to them as hierarchical acknowledgments (HACKs). The use of HACKs is important for throughput. Notice that, if the source had to wait for ACKs to be aggregated all the way from the leaf nodes, it would have to be paced based on the slowest path in the ack tree.

Tree-based protocols delegate to leaders of subtrees the decision of when to delete packets from memory, which is conditional upon receipt of HACKs from the children in the group. HACKs are sent up a B -ary in-tree composed of up to three types of nodes: a source node, leaf nodes, and hop nodes. The source node is the originator of a new packet, which it multicasts to all the receivers, and has at most B children from which to process HACKs and to send retransmissions. Leaf nodes are at the bottom of the tree and are not responsible for any children. They play the same role as receivers in the sender-initiated protocol, except that they send their HACKs only to their group leaders (hop nodes) instead of sending ACKs to the source node. Hop nodes are group leaders in between the source and leaf nodes. They send HACKs to their own group leaders one step higher in the ack tree, and they collect HACKs from the children in their group, retransmitting if necessary. They do not release data from memory until all children have acknowledged correct transmission. Obviously an ack tree consisting of the source as the only leader and leaf nodes corresponds to the sender-initiated scheme.

To simplify our analysis, we assume that the source and group leaders control the retransmission timeouts; however, such timeouts can be controlled by the children of the source and group leaders. Accordingly, when the source sends a packet, it sets a timer, and each hop node sets a timer as it becomes aware of a new packet. If there is a timeout before all HACKs have been received, the packet is assumed to be lost and is retransmitted by the source or group leader to its children. We assume that a selective repeat strategy is used, so that once a packet is received correctly, it is never rebroadcast to the group again. Because our analysis focuses on maximum attainable throughput of protocol classes, we will assume that the ack tree perfectly mirrors the multicast routing tree created by the underlying multicast routing protocol.

The first application of tree-based protocols to reliable multicasting over the internet was reported by Paul *et al.* [14], who compare three basic schemes for reliable point-to-multipoint multicasting using hierarchical structures. Their results have been fully de-

veloped as the reliable multicast transport protocol (RMTP) [12]. While our generic protocol sends a HACK for every packet sent by the source, RMTP sends HACKS only periodically, so as to conserve bandwidth. RMTP has been implemented on several platforms and has been used successfully in AT&T's "call detail data distribution network" [13].

Tree-based protocols eliminate the ACK implosion problem and free the source from having to know the receiver set, work with finite memory, provide maximum end-to-end delays that are bounded, and operate solely on messages exchanged in local groups (between a node and its children in the ack tree). As we show in Section 4, the amount of work required at each node for tree-based protocols does not increase with the number of group members, i.e., the throughput of such protocols is not dependent on the number of group members.

We define a tree-NAPP protocol as a tree-based protocol that uses NAK-avoidance and periodic polling [17] in the local groups. NAKs alone are not sufficient to guarantee reliability with finite memory, so receivers send a periodic positive (hierarchical) acknowledgment to their parents to advance the cw . Note that messages sent for the setting of timers needed for NAK-avoidance are limited to the local group, which is scalable. The tree-based multicast transport protocol (TMTP) [20] is the only specification of a tree-NAPP protocol to date.

2.5. Ring-Based Protocols

Token-ring based protocols for reliable multicast were originally developed to provide support for applications that require an atomic and total ordering of transmissions at all receivers. One of the first proposals for reliable multicasting is the token ring protocol (TRP) [3]; its aim was to combine the throughput advantages of NAKs with the reliability of ACKs. The Reliable Multicast Protocol (RMP) [19] discussed an updated WAN version of TRP. Although multiple rings are used in a naming hierarchy, the same class of protocol is used for the actual rings. Therefore, RMP has the same throughput bounds as TRP.

We base our description of generic ring-based protocols on the LAN protocol TRP and the WAN protocol RMP. The basic premise is to have only one token site responsible for ACKing packets back to the source. The source times out and retransmits packets if it does not receive an ACK from the token site within a timeout period. The ACK also serves to timestamp packets, so that all receiver nodes have a global ordering of the packets for delivery to the application layer. The protocol does not allow receivers to deliver packets until the token site has multicast its ACK.

Receivers send NAKs to the token site for selective repeat of lost packets that were originally multicast from the source. The ACK sent back to the source also serves as a token passing mechanism. If no transmissions from the source are available to piggyback the token, then a separate unicast message is sent. Since we are interested in the maximum throughput, we will not consider the latter case in this paper. The token is not passed to the next member of the ring of receivers until the new site has correctly received all packets that the former site has received. Once the token is passed, a site may clear packets from memory; accordingly, the final deletion of packets from the collective memory of the receiver set is decided by the token site, and is conditional on passing the token. The source will only delete packets when an ACK/token is received. Note that both TRP and RMP specify that retransmissions are sent unicast from the token site. Because our analysis focuses on maximum attainable throughput of protocol classes, we will assume that the token is passed exactly once per message.

3. Protocol Correctness

To address the correctness of protocol classes, we assume that nodes never fail during the duration of a reliable multicast session and that a multicast session is established correctly. Therefore, our analysis of correctness focuses on the ability of the protocol classes to sustain packet losses or errors. We assume that there exists some non-zero probability that a packet is received error-free, and that all senders and receivers have *finite* memory. Extensions of the generic tree-based protocols that ensure liveness and safety when nodes can fail are discussed by Levine, Lavo, and Garcia-Luna-Aceves [11].

The proof of correctness for ring-based protocols is given by Chang and Maxemchuk [3]. The proof that sender-initiated unicast protocols are safe and live is available from many sources (e.g., see Bertsekas and Gallager [2]). The proof does not change significantly for the sender-initiated class of reliable multicast protocols and is omitted for brevity. The safety property at each receiver is not violated, because each node can store a counter of the sequence number of the next packet to be delivered to a higher layer. The liveness property proof is also essentially the same, because the source waits for ACKs from all members in the receiver set before sliding the cw forward. Theorems 1 and 2 below demonstrate that the generic tree-based reliable multicast protocol (TRMP for short) is correct, and that receiver-initiated reliable multicast protocols are not live.

Theorem 1: TRMP is safe and live.

Proof: Let R be the set of all the nodes that belong to the reliable multicast session, including a source s . The receivers in the set are organized into a B -ary tree of height h . The proof proceeds by induction on h .

For the case in which $h = 1$, TRMP reduces to a non-hierarchical sender-initiated scheme of $R = B + 1$ nodes, with each of the B receivers practicing a given retransmission strategy with the source. Therefore, the proof follows from the correctness proof of unicast retransmission protocols presented by Bertsekas and Gallager [2].

For $h > 1$, assume the theorem holds for any t such that $(1 \leq t < h)$. We must prove the theorem holds for some $t = h$.

Safety: We must prove that each receiver of a tree of height t delivers all data in order to a higher layer. Each one keeps a variable storing the sequence number of the packet to be delivered next. Only the first error-free packet of that sequence received that was transmitted by the source, or retransmitted by the group leader is delivered, and then the variable is incremented. This procedure is continued until the session has ended. Therefore, TRMP is safe.

Liveness: We must prove that each member of a tree of height t never reaches a deadlock. Consider a subset of the tree that starts at the source and includes all nodes of the tree up to a height of $(t-1)$; the leaves of this subtree are also hop nodes in the larger tree, i.e., group leaders of the nodes at the bottom of the larger tree. By the inductive hypothesis, the liveness property is true in this subtree. We must only show that TRMP is live for a second subset of nodes consisting of leaves of the larger tree and their hop node parents. Each group in this second subset follows the same protocol, and it suffices to prove that an arbitrary group is live.

The arbitrary group in the second subset of the tree constitutes a case of sender-initiated reliable multicast, with the only difference that the original transmission is sent from the source (external to the group), not the group leader. Since the availability of a retransmission from a group leader is guaranteed by the inductive hypothesis, each group is live; therefore, the entire tree is live. $\square \mathcal{E} \mathcal{D}$

Theorem 2: A receiver-initiated reliable protocol is not live.

Proof: The proof is by example focusing on the sender and an arbitrary member of the receiver set R (where $R \geq 1$).

- Sender node, X , has enough memory to store up to M packets.
- Each packet takes 1 unit of time to reach a receiver node Y . NAKs take a finite amount of time to reach the sender.
- Let p_i denote the i^{th} packet, i beginning from zero. p_0 is sent at start time 0, but it is lost in the network.
- X sends the next $(M - 1)$ packets to Y successfully.
- Y sends a NAK stating that p_0 was not received. The NAK is either lost or reaches the sender after time M when the sender decides to send out packet p_M .
- Since X can only store up to M packets, and it has not received any NAKs for p_0 by time M , it must clear p_0 assuming that it has been received correctly.
- X then receives the NAK for p_0 at time $M + \epsilon$ and becomes deadlocked, unable to retransmit p_0 . QED

The above indicates that the receiver-initiated protocols proposed to date require an infinite memory to work correctly. In practice, this requirement implies that the source must keep in memory every packet that it sends during the lifetime of a session. This becomes impractical in long-lived sessions or in sessions in which the likelihood of lost packets or NAKs is not negligible. Fortunately, the next section shows that tree-based protocols, which we have shown to work correctly with finite memory, provide all the scaling benefits of receiver-initiated protocols.

4. Maximum Throughput Analysis

To analyze the maximum throughput that each of the generic reliable multicast protocols introduced in Section 2 can achieve, we use the same model used by Pingali *et al.* [15, 16], which focuses on the processing requirements of generic reliable multicast protocols, rather than the communication bandwidth requirements. Accordingly, the maximum throughput of a generic protocol is a function of the per-packet processing rate at the sender and receivers, and the analysis focuses on obtaining the processing times per packet at a given node.

We assume a single sender, X , multicasting to R identical receivers. The probability of packet loss is p for any node. Figure 1 summarizes all the notation used in this section. For clarity, we assume a single ack tree rooted at the source in the analysis of tree-based protocols. A selective repeat retransmission strategy is assumed in all the protocol classes since it is well known to be the retransmission strategy with the highest throughput (e.g., see Bertsekas and Gallager [2]), and its requirement of keeping buffers at the receivers is a non-issue given the small of cost memory. Assumptions specific to each protocol are listed in Section 2, and are in the interest of modeling maximum throughput.

We make two additional assumptions: (1) all loss events at any node in the multicast of a packet are mutually independent, and (2) no acknowledgments are ever lost. Our assumptions clearly fail to model real systems accurately but greatly increase the tractability of the model.

Such multicast routing protocols as CBT, PIM, and DVMRP [1, 7, 5] organize routers into trees, which means that there is a correlation between packet loss at each receiver. Our first assumption is equivalent to a scenario in which there is no correlation among packet losses at receivers and the location of those receivers in the underlying multicast routing tree of the source. We argue that the results of our analysis constitute a lower bound on maximum throughput for any protocol class that can take advantage of the relative position of receivers in the multicast routing tree for the transmission of ACKs or NAKs. We have not given any class an advantage with this assumption.

Our second assumption benefits all classes, but especially favors protocols that multicast acknowledgments. For example, NAK-avoidance is most effective if all receivers are guaranteed to receive the first NAK multicast to the receiver set. As the number of nodes involved in NAK-avoidance increases, the task of successful delivery of a NAK to all receivers becomes less probable. Both RINA and tree-NAPP protocols are favored by the assumption, but RINA protocols much more so, because the probability of delivering NAKs successfully to all receivers is exaggerated. Tree-NAPP protocols benefit from the assumption, but the number of receivers involved in the exchange of NAKs is bounded by the size of the local groups, and therefore the advantages of assuming perfect NAK transmissions are limited. Even with this handicap, our analysis shows that tree-NAPP protocols are better than RINA protocols.

Following the notation introduced by Pingali *et al.* [15, 16], we place a superscript A on any variable related to the sender-initiated protocol, $N1$ and $N2$ on variables related to the receiver-initiated and RINA protocols, respectively, and $H2$ on tree-NAPP protocols. Table 1 summarizes the bounds on maximum throughput for all the known classes of reliable multicast protocols. The results for sender-initiated, receiver-initiated, and tree-NAPP protocols are taken from the analysis presented by these authors [11] and Pingali *et al.* [15, 16]. The rest of this section analyzes tree- and ring-based protocols.

4.1. Tree-Based Protocols

We denote this class of protocols simply by $H1$, and use that superscript in all variables related to the protocol class. In the following, we derive and bound the expected cost at each type of node and then consider the overall system throughput. To make use of symmetry, we assume, without loss of generality that there are enough receivers to form a full tree at each level.

Source node. To make use of symmetry, we will assume, without loss of generality that there are enough receivers to form a full tree at each level. We consider first X^{H1} , the processing costs required by the source to successfully multicast an arbitrarily chosen packet to all receivers using the $H1$ protocol. The processing requirement for an arbitrary packet can be expressed as a sum of costs:

$$\begin{aligned}
 X^{H1} &= (\text{initial transmission}) + (\text{retransmissions}) \\
 &\quad + (\text{receiving ACKs}) \\
 X^{H1} &= X_f + X_p(1) + \sum_{m=2}^M (X_t(m) + X_p(m)) \\
 &\quad + \sum_{i=1}^{L^{H1}} X_h(i), \tag{1}
 \end{aligned}$$

where X_f is the time to get a packet from a higher layer, $X_p(m)$ is the time taken on attempt m at successful transmission of the packet, $X_t(m)$ is the time to process a timeout interrupt for transmission attempt m , $X_h(i)$ is the time to process HACK i , M is the number of transmissions that the source will have to make for this packet, and L^{H1} is the number of HACKs received using the $H1$ protocol. Taking expectations, we have

$$\begin{aligned}
 E[X^{H1}] &= E[X_f] + E[M]E[X_p] + (E[M] - 1)E[X_t] \\
 &\quad + E[L^{H1}]E[X_h]. \tag{2}
 \end{aligned}$$

What we have derived so far is extremely similar to Equations (1) and (2) in the analysis by Pingali *et al.* [15, 16]. In fact, we can use all of that analysis, with the understanding that B is the size of the receiver subset from which the source collects HACKs. Therefore, the expected number of HACKs received at the sender is

$$E[L^{H1}] = E[M](B)(1 - p). \tag{3}$$

B	-	Branching factor of a tree, the group size.
R	-	Size of the receiver set.
X_f	-	Time to feed in new packet from the higher protocol layer.
X_p	-	Time to process the time to process the transmission of a packet.
X_a, X_n, X_h	-	Times to process transmission of a ACK, NAK, or HACK.
X_t, Y_t	-	Time to process a timeout at a sender or receiver node respectively.
Y_p	-	Time to process a newly received packet.
Y_f	-	Time to deliver a correctly received packet to a higher layer.
Y_a, Y_n, Y_h	-	Times to process and transmit an ACK, NAK, or HACK respectively.
p	-	Probability of loss at a receiver; losses at different receivers are assumed to be independent events.
L_r^{H1}	-	Number of HACKS sent by receiver r per packet using a tree-based protocol.
L_r^U	-	Number of ACKS sent by a receiver r per packet using a <i>unicast</i> protocol.
L^{H1}	-	Total number of HACKS received from all receivers per packet.
M_r	-	Number of transmissions necessary for receiver r to successfully receive a packet.
M	-	Number of transmissions for all receivers to receive the packet correctly; $M = \max_r \{M_r\}$
X^w, Y^w	-	Processing time per packet at sender and receiver respectively in protocol $w \in \{A, N1, N2, H1, H2, R\}$.
H^{H1}	-	Processing time per packet at a hop node in tree-based protocols.
T^R	-	Processing time per packet at the token-site in ring-based protocols.
Λ_x^w	-	Throughput for protocol $w \in \{A, N1, N2, H1, H2, R\}$ where x is one of the source s , receiver (leaf) r , hop-node h , or token-site t . No subscript denotes overall system throughput.

Figure 1. Notation.

<i>protocol</i>	<i>processor requirements</i>	<i>p as a constant</i>	<i>p → 0</i>
Sender-initiated [15, 16]	$O\left(R\left(1 + \frac{p \ln R}{1-p}\right)\right)$	$O(R \ln R)$	$O(R)$
Receiver-initiated NAK-avoidance [15, 16]	$O\left(1 + \frac{p \ln R}{1-p}\right)$	$O(\ln R)$	$O(1)$
Ring-based (<i>unicast retrans.</i>)	$O\left(1 + \frac{(R-1)p}{1-p}\right)$	$O(R)$	$O(1)$
Tree-based	$O(B(1-p) + pB \ln B)$	$O(1)$	$O(1)$
Tree-NAPP [11]	$O\left(1 + \frac{1-p+p \ln B+p^2(1-4p)}{1-p}\right)$	$O(1)$	$O(1)$

Table 1. Analytical bounds.

Substituting Eq. 3 into Eq. 2, we can rewrite the expected cost at the source node as

$$\mathbb{E}[X^{H1}] = \mathbb{E}[X_f] + \mathbb{E}[M] \mathbb{E}[X_p] + (\mathbb{E}[M] - 1) \mathbb{E}[X_t] + \mathbb{E}[M] B(1-p) \mathbb{E}[X_h]. \quad (4)$$

Because in $H1$ the number of receivers $R = B$, the expected number of transmissions per packet is [17, 15, 16]

$$\mathbb{E}[M] = \sum_{i=1}^B \binom{B}{i} (-1)^{i+1} \frac{1}{(1-p^i)}. \quad (5)$$

Pingali *et al.* [15, 16] show $\mathbb{E}[M]$ is bounded by

$$\mathbb{E}[M] \in O\left(1 + \frac{p}{1-p} \ln B\right). \quad (6)$$

Using Eq. 6, we can bound Eq. 4 as follows

$$\begin{aligned} \mathbb{E}[X^{H1}] &\in O\left(B\left(1 + \frac{p \ln B}{1-p}\right)(1-p)\right) \\ &\in O(B(1-p) + Bp \ln B). \end{aligned} \quad (7)$$

It then follows that when p is a constant $\mathbb{E}[X^{H1}] \in O(B \ln B)$.

Leaf nodes. Let Y^{H1} denote the requirement on nodes that do not have to forward packets (leaves). Notice that leaf nodes in the $H1$

protocol will process fewer retransmissions and thus send fewer acknowledgments than receivers in the A protocol. We can again use an analysis similar to the one by Pingali *et al.* [15, 16] for receivers using a sender-initiated protocol.

$$Y^{H1} = (\text{receiving transmissions}) + (\text{sending HACKS})$$

$$Y^{H1} = \sum_{i=1}^{L_h^{H1}} \left(Y_p(i) + Y_h(i) \right) + Y_f, \quad (8)$$

where $Y_p(i)$ is the time it takes to process (re)transmission i , $Y_h(i)$ is the time it takes to send HACK i , Y_f is the time to deliver a packet to a higher layer, and L_h^{H1} is the number of HACKS generated by this node h (i.e., the number of transmissions correctly received). Since each receiver is sent M transmissions with probability p that a packet will be lost, we obtain

$$\mathbb{E}[L_r^{H1}] = \mathbb{E}[M](1-p). \quad (9)$$

Taking expectations of Eq. 8 and substituting Eq. 9 we have

$$\begin{aligned} \mathbb{E}[Y^{H1}] &= \mathbb{E}[L_r^{H1}](\mathbb{E}[Y_p] + \mathbb{E}[Y_h]) + \mathbb{E}[Y_f] \\ &= \mathbb{E}[M](1-p)(\mathbb{E}[Y_p] + \mathbb{E}[Y_h]) + \mathbb{E}[Y_f]. \end{aligned} \quad (10)$$

Again, noting the bound of $\mathbb{E}[M]$ given in Eq. 6,

$$\mathbb{E}[Y^{H1}] \in O(1-p + p \ln B). \quad (11)$$

When p is treated as a constant $\mathbb{E}[Y^{H1}] \in O(\ln B)$.

Hop nodes. To evaluate the processing requirement at a hop node, h , we note that a node caught between the source and a node with no children has a two jobs: to receive and to retransmit packets. Because it is convenient, and because a hop node is both a sender and receiver, we will express the costs in terms of X and Y . Our sum of costs is

$$\begin{aligned} H^{H1} &= (\text{receiving transmissions}) + (\text{sending HACKs}) \\ &\quad + (\text{collecting HACKs}) + (\text{retransmissions}) \\ H^{H1} &= \sum_{i=1}^{L_h^{H1}} (Y_p(i) + Y_h(i)) + Y_f + \sum_{k=1}^{L_h^{H1}} X_h(k) \\ &\quad + \sum_{m=2}^M (X_t(m) + X_p(m)). \end{aligned} \quad (12)$$

Just as in the case for the source node, L^{H1} is the expected number of HACKs received from node h 's children for this packet, and L_h^{H1} is the number of HACKs generated by node h .

$$\begin{aligned} E[H^{H1}] &= E[L_h^{H1}](E[Y_p] + E[Y_h]) + E[Y_f] \\ &\quad + (E[M] - 1)(E[X_p] + E[X_t]) \\ &\quad + E[L^{H1}]E[X_h]. \end{aligned} \quad (13)$$

We can substitute Equations 3 and 9 into Eq. 13 to obtain

$$\begin{aligned} E[H^{H1}] &= E[M](1-p)(E[Y_p] + E[Y_h]) + E[Y_f] \\ &\quad + (E[M] - 1)(E[X_p] + E[X_t]) \\ &\quad + B E[M](1-p) E[X_h]. \end{aligned} \quad (14)$$

The first two terms are equivalent to the processing requirements of a leaf node. The last two are almost the cost for a source node. Substituting and subtracting the difference yields

$$E[H^{H1}] = E[Y^{H1}] + E[X^{H1}] - E[X_f] - E[X_p]. \quad (15)$$

In other words, the cost on a hop node is the same as a source and a leaf, without the cost of receiving the data from higher layers and one less transmission (the original one). Substituting Equations 7 and 11 into 15 we have

$$\begin{aligned} E[H^{H1}] &\in O(1-p+p \ln B) \\ &\quad \cup O(B(1-p) + Bp \ln B) \\ &\in O(B(1-p) + Bp \ln B). \end{aligned} \quad (16)$$

When p is a constant $E[H^{H1}] \in O(B \ln B)$, which is the dominant term in the throughput analysis of the overall system.

Overall system analysis. Let the throughput at the sender Λ_s^{H1} be $1/E[X^{H1}]$, at the hop nodes Λ_h^{H1} be $1/E[H^{H1}]$, at the leaf nodes Λ_r^{H1} be $1/E[Y^{H1}]$. The throughput of the overall system is

$$\Lambda^{H1} = \min\{\Lambda_s^{H1}, \Lambda_h^{H1}, \Lambda_r^{H1}\}. \quad (17)$$

From Equations 7, 11, and 16 it follows that

$$1/\Lambda^{H1} \in O(B(1-p) + Bp \ln B). \quad (18)$$

If p is a constant and if $p \rightarrow 0$, we obtain

$$1/\Lambda^{H1} \in O(B \ln B) = O(1) \quad ; \quad p \text{ constant}, \quad (19)$$

$$1/\Lambda^{H1} \in O(B) = O(1) \quad ; \quad p \rightarrow 0. \quad (20)$$

Therefore, the maximum throughput of this protocol, as well as the throughput with non-negligible packet loss, is independent of the number of receivers. This is the only class of reliable multicast protocols that exhibits such degree of scalability with respect to the number of receivers.

4.2. Ring-Based Protocols

In this section we analyze the throughput of ring-based protocols, which we denote by a superscript R , using the same assumptions as in Section 4.1. Because we are assuming a constant stream of packets, we will ignore the overhead that occurs when there are no ACKs on which to piggyback token-passing messages.

Source. Source nodes practice a special form of unicast with a roaming token site. The sum of costs incurred is

$$\begin{aligned} X^R &= (\text{initial transmission}) + (\text{processing ACKs}) \\ &\quad + (\text{retransmissions}) \\ X^R &= X_f + X_p(1) + \sum_{i=1}^{L_r^U} X_a(i) \\ &\quad + \sum_{m=1}^{M_r} (X_t(m) + X_p(m)), \end{aligned} \quad (21)$$

where M_r is the number of transmissions required for the packet to be received by the token site, and has a mean of $E[M_r] = 1/(1-p)$; and let L_r^U be the number of ACKs from a receiver r (in this case the token site) sent *unicast*, i.e., the number of packets correctly received at r . This number is always 1, accordingly:

$$L_r^U = E[M_r](1-p) = 1. \quad (22)$$

Taking expectations of Eq. 21, we obtain

$$\begin{aligned} E[X^R] &= E[X_f] + E[M_r] E[X_p] + (E[M_r] - 1) E[X_t] \\ &\quad + E[L_r^U] E[X_a] \\ &= E[X_f] + \frac{1}{1-p} E[X_p] + \frac{p}{1-p} E[X_t] \\ &\quad + E[X_a]. \end{aligned} \quad (23)$$

If we again assume constant costs for all operations, it can be shown that

$$E[X^R] \in O\left(\frac{1}{1-p}\right), \quad (24)$$

which, when p is a constant, is $O(1)$ with regard to the size of the receiver set.

Token site. The current token site has the following costs: (Note both TRP and RMP specify that retransmissions are sent unicast to other $R-1$ receivers.)

$$\begin{aligned} T^R &= (\text{receiving transmission}) \\ &\quad + (\text{multicasting ACK/token}) \\ &\quad + (\text{processing NAKs}) \\ &\quad + (\text{unicasting retransmissions}) \\ T^R &= Y_f + \sum_{i=1}^{L_r^U} (Y_p(i) + Y_a(i)) + \sum_{j=1}^{L^R} X_n(j) \\ &\quad + (R-1) \text{Prob}\{M_r > 1\} \sum_{m=1}^{M_r} X_p(m), \end{aligned} \quad (25)$$

where L^R is the number of NAKs received at the token site when using a ring protocol. To derive L^R , consider M_r , the number of transmissions necessary for receiver r to successfully receive a packet. M_r has an expected value of $1/(1-p)$, and the last transmission is not NAKed. Because there are $(R-1)$ other receivers sending NAKs to the token site, we obtain

$$E[L^R] = (R-1)(E[M_r] - 1) = \frac{(R-1)p}{1-p}. \quad (26)$$

Therefore, the mean processing time at the token site is

$$\begin{aligned} E[T^R] &= E[Y_f] + E[Y_p] + E[Y_a] + E[L^R] E[X_n] \\ &\quad + (R-1)p E[M_r] E[X_p] \\ &= E[Y_f] + E[Y_p] + E[Y_a] \\ &\quad + \frac{(R-1)p}{1-p} \left(E[X_n] + E[X_p] \right). \end{aligned} \quad (27)$$

The expected cost at the token site can be bounded by

$$E[T^R] \in O\left(1 + \frac{(R-1)p}{1-p}\right), \quad (28)$$

with regard to the number of receivers. When p is a constant, $E[T^R] \in O(R)$.

Receivers. Receivers practice a receiver-initiated protocol with the current token site. We assume there is only one packet for the ACK, token, and timestamp multicast from the token site per data packet. The cost associated with an arbitrary packet are therefore

$$\begin{aligned} Y^R &= (\text{receiving ACK/token/timestamp}) \\ &\quad + (\text{receiving first transmission}) \\ &\quad + (\text{sending NAKs}) \\ &\quad + (\text{receiving retransmissions}) \\ Y^R &= Y_a + \text{Prob}\{M_r = 1\} Y_p(1) + Y_f \\ &\quad + \text{Prob}\{M_r > 1\} \sum_{i=1}^{L_r^U} Y_p(i) \\ &\quad + \text{Prob}\{M_r > 1\} \sum_{m=2}^{M_r} Y_n(m) \\ &\quad + \text{Prob}\{M_r > 2\} \sum_{n=3}^{M_r} Y_i(n). \end{aligned} \quad (29)$$

The above equation is complicated, and each term needs to be explained. The first term is the cost of receiving the ACK/token/timestamp packet from the token site; the second is the cost of receiving the first transmission sent from the sender, assuming it is received error free; the third is the cost of delivering an error-free transmission to a higher layer; the fourth is the cost of receiving the retransmissions from the token site, assuming that the first failed; and the last two terms consider that a NAK is sent only if the first transmission attempt fails and that an interrupt occurs only if a NAK was sent. Taking expectations, we obtain

$$\begin{aligned} E[Y_R] &= E[Y_a] + (1-p) E[Y_p] + E[Y_f] \\ &\quad + p E[L_r^U] E[Y_p] \\ &\quad + p(E[M_r | M_r > 1] - 1) E[Y_n] \\ &\quad + p^2(E[M_r | M_r > 2] - 2) E[Y_i]. \end{aligned} \quad (30)$$

It follows from the distribution of M_r that [15, 16]

$$E[M_r | M_r > 1] = \frac{2-p}{1-p}, \quad (31)$$

$$E[M_r | M_r > 2] = \frac{3-2p}{1-p}. \quad (32)$$

Substituting Equations 22, 31, and 32 into Eq. 30 we have

$$\begin{aligned} E[Y^R] &= E[X_a] + (1-p) E[Y_p] + E[Y_f] + p E[Y_p] \\ &\quad + \frac{p}{1-p} \left(E[Y_n] + p E[Y_i] \right). \end{aligned} \quad (33)$$

Assuming all operations have constant costs, it can be shown that

$$E[Y^R] \in O\left(\frac{1+p^2}{1-p}\right), \quad (34)$$

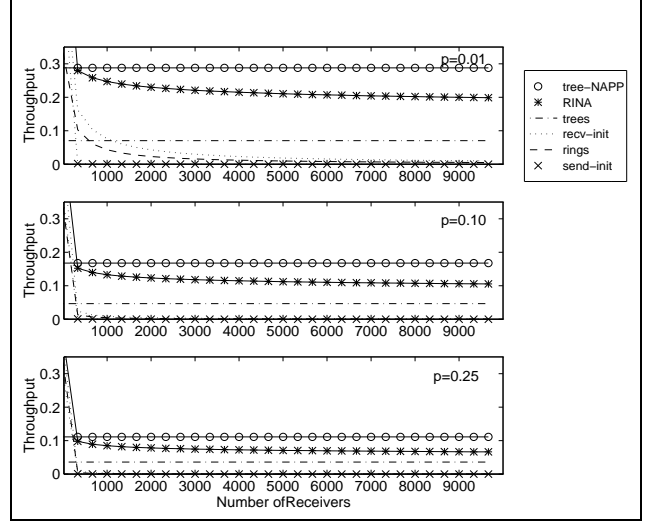


Figure 2. The throughput graph from the exact equations for each protocol. The probability of packet loss is 1%, 10%, and 25% respectively. The branching factor for trees is set at 10.

with regard to the size of the receiver set. If we consider p as a constant, then $E[Y^R] \in O(1)$.

Overall system analysis. The overall system throughput of R , the generic token ring protocol, is equal to the minimum attainable throughput at each of its parts:

$$\Lambda^R = \min\{\Lambda_s^R, \Lambda_t^R, \Lambda_r^R\}. \quad (35)$$

From Equations 24, 28 and 34 it follows that if p is a constant and for $p \rightarrow 0$, we obtain

$$1/\Lambda^R \in O\left(1 + \frac{(R-1)p}{1-p}\right) ; \quad p \text{ constant}, \quad (36)$$

$$1/\Lambda^R \in O(1) ; \quad p \rightarrow 0. \quad (37)$$

When $p \rightarrow 0$, the maximum throughput of this class of protocols is $O(1)$ and not dependent of the number of receivers.

5. Numerical Results

To compare the relative performance of the various classes of protocols, all mean processing times are set equal to 1, except for the costs of sending or receiving periodic HACKS in tree-NAPP protocols which are set to 0.1. Figure 2 compares the relative throughputs of the protocols A , $N1$, $N2$, $H1$, $H2$, and R as defined in Section 2. The graph represents the inverse of

Equations 14 and 27, respectively, which are the throughputs for the tree-based and ring-based protocols, as well as the inverse of the throughput equations derived previously [15, 16, 11] for sender-initiated, receiver-initiated, and tree-NAPP protocols.

The top, middle and bottom graphs correspond to increasing probabilities of packet loss, 1%, 10%, and 25%, respectively. The performance of NAK-avoidance protocols, especially tree-NAPP protocols, is clearly superior. However, our assumptions place these two sub-classes at an advantage over their base classes. First, we assume that no acknowledgments are lost or are received in error. The effectiveness of NAK-avoidance is dependent on the

probability of NAKs reaching all receivers, and thus, without our assumption, the effectiveness of NAK-avoidance decreases as the number of receivers involved increases. Accordingly, tree-NAPP protocols have an advantage that is limited by the branching factor, and RINA protocols have an advantage that increases with the size of the entire receiver set. Second, we assume that the timers used for NAK-avoidance are set perfectly. In reality the messages used to set timers would be subject to end-to-end delays that exhibit no regularity and can become arbitrarily large.

We conjecture that the relative performance of NAK-avoidance subclasses would actually lie closer to their respective base classes, depending on the effectiveness of the NAK-avoidance scheme; in other words, the curves shown are upper bounds. Our results show that when considering only the base classes (since not one has an advantage over another) the tree-based class performs better than all the other classes. When considering only the sub-classes that use NAK-avoidance, tree-NAPP protocols perform better than RINA protocols, even though our model provides an unfair advantage to RINA protocols.

It is the hierarchical structure organization of the receiver set in tree-based protocols that guarantees scalability and improves performance over other protocols. Using NAK-avoidance on a small scale increases performance further. In addition, if NAK-avoidance failed for a tree-NAPP protocol, the performance would still be independent of the size of the receiver set. RINA protocols do not have this property. Failure of the NAK-avoidance for RINA protocols would result in unscalable performance like that of a receiver-initiated protocol, which degrades quickly with increasing packet loss.

Any increase in processor speed, or a smaller branching factor would also increase throughput for all tree-based protocols. However, for the same number of receivers, a smaller branching factor implies a larger number of tree-hops some retransmissions must traverse to receivers expecting them further down the tree. For example, if a packet is lost immediately at the source, the retransmission is multicast only to its children and all other nodes in the tree must wait until the retransmission trickles down the tree-structure. This poses a latency problem that can be addressed by taking advantage of the dependencies in the underlying multicast routing tree. Retransmissions could be multicast only toward all receivers attached to routers on the subtree of the router attached to the receiver which has requested the missing data. However, to date there is no proposed scheme which accomplishes this task. The number of tree hops from the receiver to the source is also a factor in how quickly the source can release data from memory in the presence of node failures, as discussed by Levine, Lavo, and Garcia-Luna-Aceves [11].

Figure 3 shows the number of supportable receivers by each of the different classes, relative to processor speed requirements. This number is obtained by normalizing all classes to a baseline processor, as described by Pingali *et al.* [15, 16]. The baseline uses protocol A and can support exactly one receiver; if $\mu^\omega[R]$, $\omega \in \{A, N1, N2, H1, H2, R\}$ is the speed of the processor that can support at most R receivers under protocol ω , we set $\mu^A[1] = 1$. The baseline cost is equal to [15, 16]

$$E[X^A] \Big|_{R=1} = \frac{1}{\mu^A[1]} \frac{3-p}{1-p} = \frac{3-p}{1-p}. \quad (38)$$

Using Equations 38, 13, and 27 we can derive the following μ s for tree-based and ring-based protocols, respectively:

$$\begin{aligned} \mu^{H1}[R] &= \frac{1}{E[X^A]} E[H^{H1}] \\ &= \frac{1}{E[X^A]} (E[M](1-p)(2) + 1 \\ &\quad + (E[M] - 1)(2) + B E[M](1-p)) \end{aligned}$$

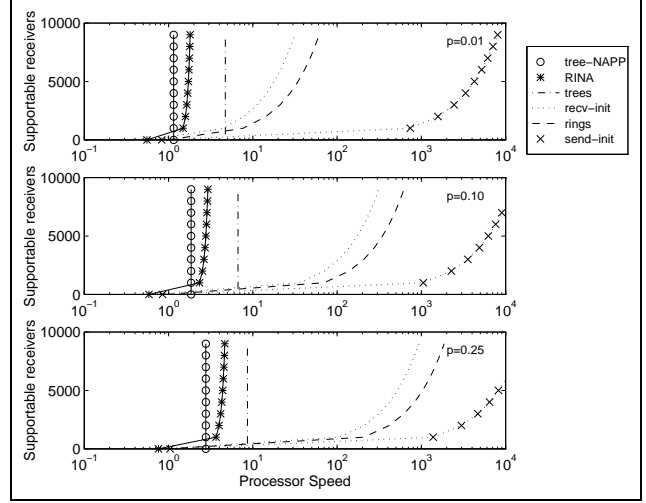


Figure 3. Number of supportable receivers for each protocol. The probability of packet loss is 1%, 10%, and 25% respectively. The branching factor for trees is set at 10.

$$= \frac{1}{E[X^A]} (E[M](4+B-(2+B)p) - 1), \quad (39)$$

$$\begin{aligned} \mu^R[R] &= \frac{1}{E[X^A]} E[T^R] \\ &= \frac{1}{E[X^A]} \left(1 + 1 + 1 + \frac{(R-1)p}{(1-p)}(1+1) \right) \\ &= \frac{1}{E[X^A]} \left(3 + \frac{2(R-1)p}{(1-p)} \right). \quad (40) \end{aligned}$$

The number of supportable receivers derived for sender-initiated, receiver-initiated, and tree-NAPP protocols are shown to be [15, 16, 11],

$$\begin{aligned} \mu^A[R] &= \frac{1}{E[X^A]} E[M](2 + R(1-p)), \\ \mu^{N1}[R] &= \frac{1}{E[X^A]} (1 + E[M] + Rp/(1-p)), \\ \mu^{N2}[R] &= \frac{1}{E[X^A]} (2 E[M]), \\ \mu^{H2}[R] &= \frac{1}{E[X^A]} ((4-p) E[M] - 1.9 + 0.1B \\ &\quad + p^2 \left(\frac{3-2p}{1-p} - 2 \right)). \end{aligned}$$

Because the exact value of $E[M]$ is difficult to compute for large values of R , we use the approximation [15, 16],

$$E[M] \approx a + \frac{(H_{35} - H_R)}{\ln(p)}, \quad (41)$$

where a is the value of $E[M]$ for $R = 35$ and H_k is the harmonic series. When evaluating $\mu^{H1}[R]$ and $\mu^{H2}[R]$, an exact value for $E[M]$ is used because the number of receivers is always $R = B = 10$.

From Figure 3, it is clear that only the tree-based classes can support any number of receivers for the same processor speed

bound at each node. It is also clear that, in terms of performance, tree-NAPP protocols are superior to other classes.

Because of the unicast nature of retransmissions in ring-based protocols, these protocols approach sender-initiated protocols; this indicates that allowing only multicast retransmissions would improve performance greatly.

6. Conclusions

We have compared and analyzed the four known classes of reliable multicast protocols. The results are summarized in Table 1. It is already known that sender-initiated protocols are not scalable at all since the source must account for every receiver listening. Receiver-initiated protocols are more scalable, especially when NAK-suppression schemes are used to avoid overloading the source with retransmission requests. However, because of the unbounded-memory requirement, this protocol class can only be used efficiently with application-layer support, and only for limited applications. Ring-based protocols were designed for atomic and total ordering of packets. TRP and RMP limit their throughput by requiring retransmissions to be unicast. It would be possible to reduce the cost bound to $O(\ln R)$, assuming p to be a constant, if the NAK-avoidance techniques presented by Ramakrishnan and Jain [17] were used.

Our analysis shows that trees are the answer to the scalability problem for reliable multicasting. Only tree-based and tree-NAPP classes have a throughput that is constant with respect to the number of receivers even when the probability of packet loss is not negligible. Furthermore, our model predicts tree-NAPP protocols as the best method for supporting reliable multicast.

Of course, our model constitutes only a crude approximation of the actual behavior of reliable multicast protocols. In the Internet, an ACK or a NAK is simply another packet, and the probability of an ACK or NAK being lost or received in error is much the same as the error probability of a data packet. This assumption gives protocols that use NAK-avoidance an advantage over other classes. Therefore, it is more reasonable to compare them separately: our results show that tree-based protocols without NAK-avoidance perform better than other classes that do not use NAK-avoidance, and that tree-NAPP protocols perform better than RINA protocols even though RINA protocols have an artificial advantage over every other class.

We conjecture that, once the effect of ACK or NAK failure is accounted for, the same relative performance of protocols that do not use NAK-avoidance will be seen. Furthermore, we believe the true performance of tree-NAPP and RINA protocols will lie closer to their respective base classes, depending on the effectiveness of the NAK-avoidance scheme; in other words, the curves shown for NAK-avoidance protocols are upper bounds.

The fact that packet failures are correlated along a multicast routing trees setup by CBT or PIM means that our model's assumption of independent packet failures leads to lower bounds on the maximum throughput, because reliable multicast protocols can take advantage of the structure of the underlying multicast routing tree. Our analysis provides no advantage to any class however, and we believe the relative performances would not change if we did not make this assumption.

Because tree-based protocols delegate responsibility for retransmission to receivers and because they employ techniques applicable to either sender- or receiver-initiated protocols within local groups (i.e., a node and its children in the tree) of the ack tree only, any mechanism that can be used in a receiver-initiated protocol can be adopted in a tree-based protocol, with the added benefit that the throughput and number of supportable receivers is completely independent of the size of the receiver set, regardless of the likelihood with which packets are received correctly at the receivers. Based on

these results, future work on reliable multicasting for the internet should focus on developing new tree-based protocols.

References

- [1] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT): An architecture for scalable inter-domain multicast routing. In *Proc. ACM SIGCOMM'93*, pages 85–95, October 1993.
- [2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, Englewood Cliffs, New Jersey, second edition, 1992.
- [3] J.-M. Chang and N. F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.
- [4] D. D. Clark, M. L. Lambert, and L. Zhang. NETBLT: A high throughput transport protocol. In *Proc. ACM SIGCOMM. ACM Computer Communication Review*, pages 353–359, Aug. 1987.
- [5] S. Deering. *Multicast routing in a datagram internetwork*. PhD thesis, Stanford University, Palo Alto, California, Dec. 1991.
- [6] S. Deering and D. Cheriton. Multicast routing in datagram inter-networks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.
- [7] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, and others. An architecture for wide-area multicast routing. In *ACM SIGCOMM'94*, pages 126–135, 1994.
- [8] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proc. ACM SIGCOMM'95. ACM Computer Communication Review*, pages 342–356, August 1995.
- [9] H. Holbrook, S. K. Singhal, and D. R. Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. In *Proc. ACM SIGCOMM'95*, pages 328–341, August 1995.
- [10] Jon B. Postel, ed. Transmission control protocol. RFC 793, September 1981.
- [11] B. N. Levine, D. Lavo, and J. Garcia-Luna-Aceves. The case for concurrent reliable multicasting using shared ack trees. In *Proc. ACM Multimedia*, November 1996.
- [12] J. C. Lin and S. Paul. RMTP: A reliable multicast transport protocol. In *IEEE Infocom*, pages 1414–1425, March 1996.
- [13] S. Paul, R. Buskens, K. Sabnani, M. Siddiqui, J. Lin, and S. Bhattacharya. Reliable multicasting. Slides presented at IEEE Computer Communications Workshop, Orcas Island, Washington, Sept 1995.
- [14] S. Paul, K. K. Sabnani, and D. K. Kristol. Multicast transport protocols for high speed networks. In *International Conference on Network Protocols*, pages 4–14, 1994.
- [15] S. Pingali. *Protocol and Real-Time Scheduling Issues for Multimedia Applications*. PhD thesis, University of Massachusetts Amherst, September 1994.
- [16] S. Pingali, D. Towsley, and J. F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Performance Evaluation Review*, volume 22, pages 221–230, May 1994.
- [17] S. Ramakrishnan and B. N. Jain. A negative acknowledgment with periodic polling protocol for multicast over lan. In *IEEE Infocom*, pages 502–511, March 1987.
- [18] W. T. Strayer, B. Dempsey, and A. Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley Publishing Company, 1992.
- [19] B. Whetten, S. Kaplan, and T. Montgomery. A high performance totally ordered multicast protocol. Available from research.ivv.nasa.gov by `ftp/pub/doc/RMP/RMP_dagstuhl.ps`, August 1994.
- [20] R. Yavatkar, J. Griffioen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proc. ACM Multimedia*, pages 333–44, 1995.