Multimedia Systems 6: 334–348 (1998)

Multimedia Systems

A comparison of reliable multicast protocols*

Brian Neil Levine, J.J. Garcia-Luna-Aceves

Computer Engineering Department, School of Engineering, University of California, Santa Cruz, Santa Cruz, CA 95064 USA;
e-mail:{brian,jj}@cse.ucsc.edu

Abstract. We analyze the maximum throughput that known classes of reliable multicast transport protocols can attain. A new taxonomy of reliable multicast transport 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 acknowledgements (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. It is shown that tree-based protocols constitute the most scalable class of all reliable multicast protocols proposed to date.

Key words: Reliable multicast – Multicast transport protocols – ACK implosion – Tree-based protocols

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. Minimally, an end-to-end multicast service ensures that all packets from each source are delivered to each receiver in the session within a finite amount of time and free of errors and that packets are safely deleted within a finite time. Additionally, the service may ensure that each packet is delivered only once and in the order sent by the source. Although reliable broadcast protocols have existed for quite some time [3], viable approaches on the provision of end-to-end reliable multicasting over the Internet are just emerging. The end-to-end reliable multicast problem facing the future Internet is compounded by its current size and continuing growth, which makes the handling of acknowledgements a major challenge commonly referred to as the *acknowledgement (*ACK*) implosion problem.*

The two most popular approaches to end-to-end 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 acknowledgments (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 acknowledgement (NAK) when it detects an error or a lost packet.

The first comparative analysis of ideal sender-initiated and receiver-initiated reliable multicast protocols was presented by Pingali et al. [17, 18]. 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 becomes independent of the number of receivers as the probability of packet loss becomes negligible. However, as this paper demonstrates, the ideal receiver-initiated protocols cannot prevent deadlocks when they operate with finite memory, i.e., when the applications using the protocol services cannot retransmit any data themselves, and existing implementations of receiver-initiated protocols have inherent scaling limitations that stem from the use of messages multicast to all group members and used to set timers needed for NAK avoidance, the need to multicast NAKs to all hosts in a session, and to a lesser extent, the need to store all messages sent in a session.

* Supported in part by the Office of Naval Research under Grant N00014-94-1-0688, and by the Defense Advanced Research Projects Agency (DARPA) under grant F19628-96-C-0038
Correspondence to: B.N. Levine

This paper addresses the question of whether a reliable multicast transport protocol (reliable multicast protocol, for short) can be designed that enjoys all the scaling properties of the ideal receiver-initiated protocols, while still being able to operate correctly with finite memory. To address this question, the previous analysis by Pingali et al. [17, 18, 22] is extended to consider the maximum throughput of generic ring-based protocols, which organize receivers into a ring, and two classes of tree-based protocols, which organize receivers into ACK trees. 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, 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 (i.e., without requiring the application level to store all data sent in a session) 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. Section 4 extends the analysis by Pingali et al. [17, 18, 22] 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. 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 [8, 17, 18, 22] 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]).

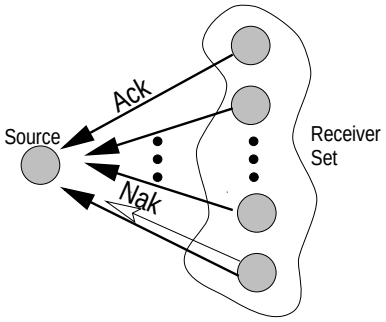


Fig. 1. A basic diagram of a sender-initiated protocol

Each reliable protocol assumes the existence of multicast routing trees 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], Ordered Core-Based Trees (OCBT) [20], Protocol-Independent Multicast (PIM) [7], or the Multicast Internet Protocol (MIP) [14].

2.1 Sender-initiated protocols

In the past [17, 18], 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 acknowledgements 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. Receivers are not restricted from directly contacting the source. 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 unicasting) 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. A simple illustration of a sender-initiated protocol is presented in Fig. 1.

Notice that, regardless of whether a sender-driven or receiver-driven 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. For a sender-initiated protocol, if a sender-driven retransmission strategy is used, the sender "polls" the receivers for ACKs by retransmitting after a timeout. If a receiver-driven retransmission strategy is used, the receivers "poll" the source (with an ACK) after they time out.[1]

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 can be 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 [19]. This protocol is a broadcast protocol for local area networks (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) [21], 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 [17, 18] 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. Receivers are not restricted from directly contacting the source. 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*). Figure 2 is a simple illustration of a receiver-initiated protocol.

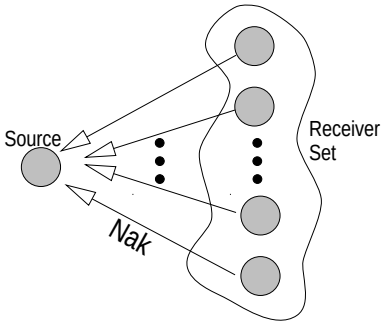


Fig. 2. A basic diagram of a receiver-initiated protocol

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 transmission errors. To remedy this problem, previous work on receiver-initiated protocols [8, 17, 18] 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 [17, 18, 22] to have better performance than the basic receiver-initiated protocol. The resulting generic RINA protocol is as follows [17, 18]. 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.

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 embedded into an internet collaborative whiteboard application called *wb*. SRM sets timers based on low-rate, periodic "session messages" multicast by every member of the group. The messages specify a time stamp used by the receivers to estimate the delay from the source, and the highest sequence number generated by the node as a source.[2] The average

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

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

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, or that the application layer store all relevant data.

We note that NAKs from receivers are used to advance the *cw*, which is controlled by the receivers, and the sequence number in each multicast session message is used to "poll" the receiver set, i.e., to ensure that each receiver is aware of missing packets. Although session messages implement a "polling" function [19], they cannot be used to advance the *mw*, as in a sender-initiated protocol, because a sender specifies its highest sequence number as a source, not the highest sequence number heard from the source.[3]

In practice, the persistence of session messages forces the source to process the same number of messages that would be needed for the source to know the receiver set over time (one periodic message from every receiver). Accordingly, as defined, the basic dissemination of session messages in 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 RINA protocols for reliable multicasting. First, as we show in the next section, a RINA protocol requires that data needed for retransmission be rebuilt from the application. This 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 scheme 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, described in the next section). 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! Furthermore, nodes that are on congested links, LANs or regions may constantly bother the rest of the multicast group by multicasting NAKs. Approaches to limit the scope of NAKs and retransmission are still evolving [8]. However, current proposals still rely on session messages that reach all group members.

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 ideal receiver-initiated protocol has three main advantages over sender-initiated protocols, namely: (a) the source does not know the receiver set, (b) the source does not have to process ACKs from each receiver, and (c) the receivers pace the source. The limitation of this protocol is that it has no mechanism for the source to know when it can safely release data from memory. Furthermore, as we have argued, the practical implementations of the receiver-initiated approach fail to provide advantages (a) and (b). The following two protocol 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*).

2.3 Tree-based protocols

Tree-based protocols are characterized by dividing the receiver set into groups, distributing retransmission responsibility over an acknowledgement tree (ACK tree) structure built from the set of groups, with the source as the root of the tree. A simple illustration of a tree-based protocol is presented in Fig. 3. The ACK tree structure prevents receivers from directly contacting the source, in order to maintain scalability with a large receiver set.

The ACK tree consists of the receivers and the source organized into *local groups*, with each such group having a *group leader* in charge of retransmissions within the local group. The source is the group leader in charge of retransmissions to its own local group. Each group leader other than the source communicates with another local group (to either a child or the group leader) closer to the source to request retransmissions of packets that are not received correctly. Group leaders may be children of another local group, or minimally, may just be in contact with another local group. Each local group may have more than one group leader to handle multiple sources. Group leaders could also be chosen dynamically, e.g., through token passing within the local group.

Hosts that are only children are at the bottom of the ACK tree, and are termed *leaves*. Obviously, an ACK tree consisting of the source as the only leader and leaf nodes corresponds to the sender-initiated scheme.

Acknowledgments from children in a group, including the source's own group, are sent only to the group leader. The children of a group send their acknowledgements to the group leader as soon as they receive correct packets, advancing the *cw*; we refer to such acknowledgements as *local* ACKs or *local* NAKs, i.e., retransmissions are triggered by local ACKs and local NAKs unicast to group leaders by their children. Similar to sender-initiated schemes, the use of local NAKs is unnecessary for correct operation of the protocol.

Tree-based protocols can also delegate to leaders of subtrees the decision of when to delete packets from memory (i.e., advance the *mw*), which is conditional upon receipt of *aggregate* ACK*s* from the children of the group. Aggregate ACKs start from the leaves of the ACK tree, and propagate toward the source, one local group at a time. A group leader cannot send an aggregate ACK until all its children have sent an aggregate ACK. Using aggregate ACKs is necessary to ensure that the protocol operates correctly even if group leaders fail, or if the ACK tree is partitioned for long periods

[3] Our prior description of SRM [11, 12] incorrectly assumed that session messages contained the highest sequence number heard from the source. We thank Steve McCanne for pointing this out.

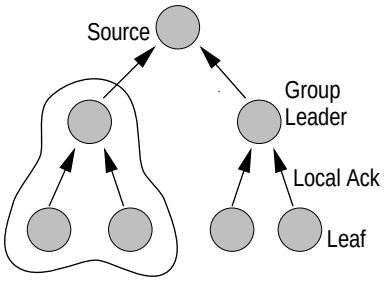


Fig. 3. A basic diagram of a tree-based protocol

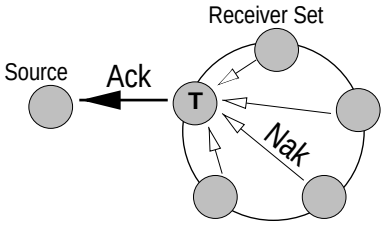


Fig. 4. A basic diagram of a ring-based protocol

of time [12]. If aggregate ACKs are not used, i.e., if a group leader only waits for all its children to send local ACKs before advancing the *mw*, then correct operation after group leaders fail can only be guaranteed by not allowing nodes to delete packets; this is the approach used in all tree-based protocols [13, 16, 24] other than Lorax [12]. The Lorax protocol [12] is the first tree-based protocol to build a single *shared* ACK *tree* for use by multiple sources in a single session, and to use aggregate ACKs to ensure correct operation after hosts in the ACK tree fail.

The use of local ACKs and local NAKs for requesting retransmissions is important for throughput. If the source scheduled retransmissions based on aggregate ACKs, it would have to be paced based on the slowest path in the ACK tree. Instead, retransmissions are scheduled independently in each local group.

Tree-based protocols eliminate the ACK-implosion problem, free the source from having to know the receiver set, and operate solely on messages exchanged in local groups (between a group leader and its children in the ACK tree). Furthermore, if aggregate ACKs are used, a tree-based protocol can work correctly with finite memory even in the presence of receiver failures and network partitions.

To simplify our analysis and description of this protocol, we assume that the 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 group leader sets a timer as it becomes aware of a new packet. If there is a timeout before all local ACKs have been received, the packet is assumed to be lost and is retransmitted by the source or group leader to its children.

The first application of tree-based protocols to reliable multicasting over an internet was reported by Paul et al. [15], who compared three basic schemes for reliable point-to-multipoint multicasting using hierarchical structures. Their results have been fully developed as the reliable multicast transport protocol (RMTP) [13, 16]. While our generic protocol sends a local ACK for every packet sent by the source, RMTP sends local ACKs only periodically, so as to conserve bandwidth and to reduce processing at each group leader, increasing attainable throughput.

We define a tree-NAPP protocol as a tree-based protocol that uses NAK avoidance and periodic polling [19] in the local groups. NAKS alone are not sufficient to guarantee reliability with finite memory, so receivers send a periodic positive local ACK 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) [24] is an example of a tree-NAPP protocol.

2.4 Ring-based protocols

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) [23] 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. A simple illustration of a ring-based protocol is presented in Fig. 4. 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 deletes packets only 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

A protocol is considered *correct* if it is shown to be both *safe* and *live* [2]. Given the minimum definition of reliable service we have assumed, for any reliable multicast protocol to be live, no deadlock should occur at any receiver or at the source. For the protocol to be safe, all data sent by the source must be delivered to a higher layer within a finite time. 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 and is permanent. Therefore, our analysis of correctness focuses only 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.

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., 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 liveness 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 safety property proof is also essentially the same, because the source waits for ACKs from all members in the receiver set before sliding the *cw* and *mw* 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 $t = h$.

Liveness. We must prove that each member of a tree of height t is live. 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 group leaders 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 group leader parents. Each group in this second subset follows the same protocol, and it suffices to prove that an arbitrary group are 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 leaves can only contact the group leader, we must prove this relationship is live. The inductive hypothesis guarantees that the group leader and its parent is live.

Assume the source transmits a packet i at time c_1, and that it is received correctly and delivered at all leaves of the arbitrary group at time c_2. Let c_3 be the time at which the group leader deletes the packet and advances the *mw*. The protocol is live and will not enter into deadlock if $c_1 < c_2 < c_3$, and c_3 is finite. The rest of the proof follows from the proof by Bertsekas and Gallager [2] for unicast ARQ protocols, where the group leader takes the place of the source. Therefore, TRMP is live.

Safety. The safety of TRMP follows directly, because our proof of liveness shows that any arbitrary packet i is delivered at each receiver within a finite time. QED

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 ideal receiver-initiated protocol requires 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.

Theorem 1 assumes that no node failures or network traffic occur. However, node failures do happen in practice, which changes the operational requirements of practical tree-based protocols. For tree-based protocols, it can be shown that deleting packets from memory after a node receives local ACKs from its children is not live. Aggregate ACKs are necessary to ensure correct operation of tree-based protocols in the presence of failures. Lorax [12] is the only tree-based protocol that uses aggregate ACKs and can operate with finite memory in the presence of node failures or network partitions.

4 Maximum throughput analysis

4.1 Assumptions

To analyze the maximum throughput that each of the generic reliable multicast protocols introduced in Sect. 2 can achieve, we use the same model as Pingali et al. [17, 18], 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 5 summarizes all the notation used in this section. For clarity, we assume a single ACK tree rooted at a single 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 [2], and its requirement of keeping buffers at the receivers is a non-issue given the small cost of memory. Assumptions specific to each protocol are listed in Sect. 2, and are in the interest of modeling maximum throughput.

We make two additional assumptions: (1) no acknowledgements are ever lost, and (2) all loss events at any node in the multicast of a packet are mutually independent.

Such multicast routing protocols as CBT, OCBT, PIM, MIP, and DVMRP [1, 5, 7, 14, 20] organize routers into trees, which means that there is a correlation between packet loss at each receiver. Our first assumption benefits all classes, but especially favors protocols that multicast acknowledgements. In fact, this assumption is essential for RINA protocols, in order to analyze their maximum attainable throughput, because 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.

Our second 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. Protocols that can take advantage of the relative position of receivers in the multicast routing tree for the transmission of ACKs, NAKs, or retransmissions would possibly attain higher throughput than predicted by this model. However, no class is given any relative advantage with this assumption.

Table 1 summarizes the bounds on maximum throughput for all the known classes of reliable multicast protocols. Our results clearly show that tree-NAPP protocols constitute the most scalable alternative.

Table 1. Analytical bounds

Protocol	*Processor requirements*	p *as a constant*	$p \to 0$
Sender-initiated [17, 18]	$O\left(R(1+\frac{p\ln R}{1-p})\right)$	$O(R\ln R)$	$O(R)$
Receiver-initiated NAK avoidance [17, 18]	$O\left(1+\frac{p\ln R}{1-p}\right)$	$O(\ln R)$	$O(1)$
Ring-based (*unicast retrans.*)	$O\left(1+\frac{(R-1)p}{1-p}\right)$	$O(R)$	$O(1)$
Tree-based	$O(B(1-p)+pB\ln B)$	$O(1)$	$O(1)$
Tree-NAPP	$O\left(1+\frac{1-p+p\ln B+p^2(1-4p)}{1-p}\right)$	$O(1)$	$O(1)$

4.2 Sender- and receiver-initiated protocols

Following the notation introduced by Pingali et al. [17, 18], we place a superscript A on any variable related to the sender-initiated protocol, and $N1$ and $N2$ on variables related to the receiver-initiated and RINA protocols, respectively. The maximum throughput of the protocols for a constant stream of packets to R receivers is [17, 18]

$$1/\Lambda^A \in O\left(R(1+\frac{p\ln R}{1-p})\right), \tag{1}$$

$$1/\Lambda^{N1} \in O\left(1+\frac{pR}{1-p}\right), \tag{2}$$

$$1/\Lambda^{N2} \in O\left(1+\frac{p\ln R}{1-p}\right). \tag{3}$$

Even as the probability of packet loss goes to zero, the throughput of the sender-initiated protocol is inversely dependent on R, the size of the receiver set, because an ACK must be sent by every receiver to the source once a transmission is correctly received. In contrast, as p goes to zero, the throughput of receiver-initiated protocols becomes independent of the number of receivers. Notice, however, that the throughput of a receiver-initiated protocol is inversely dependent with R, the number of receivers, or with $\ln R$, when the probability of error is not negligible. We note that this result assumes *perfect* setting of the timers used in a RINA protocol without cost and that a single NAK reaches the source, because we are only interested in the maximum attainable throughput of protocols.

4.3 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.

Without loss of generality, we assume that each local group in the ACK tree consists of B children and a group leader. This allows us to make use of symmetry in our throughput calculations. We also assume that local ACKs advance the *mw* rather than aggregate ACKs, because by assumption no receiver fails in the system. We assume *perfect* setting of timers without cost and that a single NAK reaches the source, because we are only interested in the maximum attainable throughput of protocols.

4.3.1 Source node

We consider first X^{H1}, the processing costs required by the source to successfully multicast an arbitrarily chosen packet

B	–	Branching factor of a tree, the group size.
R	–	Size of the receiver set.
X_f	–	Time to feed in a new packet from the higher protocol layer.
X_p	–	Time to process the transmission of a packet.
X_a, X_n, X_h	–	Times to process transmission of an ACK, NAK, or local ACK, respectively.
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 local ACK, respectively.
Y_p	–	Probability of loss at a receiver; losses at different receivers are assumed to be independent events.
L_r^{H1}	–	Number of local acks 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 *unicast* protocol.
L^{H1}	–	Total number of local ACKs 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 (for protocols A, $N1$ and $N2$); $M = \max_r\{M_r\}$
M^{H1}, M^{H2}	–	Number of transmissions for all receivers to receive the packet correctly for protocols $H1$ and $H2$.
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}, H^{H2}	–	Processing time per packet at a group leader in tree-based and tree- NAPP protocols, respectively.
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, group leader h, or token-site t. No subscript denotes overall system throughput.
X_Φ, Y_Φ	–	Times to process the reception and transmission, respectively, of a periodic local ACK.

Fig. 5. Notation

to all receivers using the $H1$ protocol. The processing requirement for an arbitrary packet can be expressed as a sum of costs:

$$X^{H1} = \text{(initial transmission)} + \text{(retransmissions)} + \text{(receiving ACKs)}$$

$$X^{H1} = X_f + X_p(1) + \sum_{m=2}^{M^{H1}} (X_t(m) + X_p(m)) + \sum_{i=1}^{L^{H1}} X_h(i), \quad (4)$$

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 local ACK i, M^{H1} is the number of transmissions that the source will have to make for this packet using the $H1$ protocol, and L^{H1} is the number of local ACKs received using the $H1$ protocol. Taking expectations, we have

$$\begin{aligned}\mathrm{E}[X^{H1}] = {} & \mathrm{E}[X_f] + \mathrm{E}[M^{H1}]\mathrm{E}[X_p] \\ & + (\mathrm{E}[M^{H1}] - 1)\mathrm{E}[X_t] \\ & + \mathrm{E}[L^{H1}]\mathrm{E}[X_h]. \end{aligned} \quad (5)$$

What we have derived so far is extremely similar to Eqs. 1 and 2 in the analysis by Pingali et al. [17, 18]. In fact, we can use all of their analysis, with the understanding that B is the size of the receiver subset from which the source collects local ACKs. Therefore, the expected number of local ACKs received at the sender is

$$\mathrm{E}[L^{H1}] = \mathrm{E}[M^{H1}](B)(1-p). \quad (6)$$

Substituting Eq. 6 into Eq. 5, we can rewrite the expected cost at the source node as

$$\begin{aligned}\mathrm{E}[X^{H1}] = {} & \mathrm{E}[X_f] + \mathrm{E}[M^{H1}]\mathrm{E}[X_p] \\ & + (\mathrm{E}[M^{H1}] - 1)\mathrm{E}[X_t] \\ & + \mathrm{E}[M^{H1}]B(1-p)\mathrm{E}[X_h]. \end{aligned} \quad (7)$$

Pingali et al. [17, 18] have shown that the expected number of transmissions per packet in A, $N1$, and $N2$ equals

$$\mathrm{E}[M] = \sum_{m=1}^{\infty} \left(1 - \left(1 - p^{(m-1)}\right)^R\right). \quad (8)$$

Because in $H1$ the number of receivers $R = B$, the expected number of transmissions per packet in the $H1$ protocol is

$$\mathrm{E}[M^{H1}] = \sum_{m=1}^{\infty} \left(1 - \left(1 - p^{(m-1)}\right)^B\right), \quad (9)$$

which can be simplified to [17, 18, 19]

$$\mathrm{E}[M^{H1}] = \sum_{i=1}^{B} \binom{B}{i} (-1)^{i+1} \frac{1}{(1-p^i)}. \quad (10)$$

Pingali et al. [17, 18] provide a bound of $\mathrm{E}[M]$ that we apply to $\mathrm{E}[M^{H1}]$ with $R = B$ to obtain

$$\mathrm{E}[M^{H1}] \in O\left(1 + \frac{p}{1-p}\ln B\right). \quad (11)$$

Using Eq. 11, we can bound Eq. 7 as follows

$$\begin{aligned}\mathrm{E}[X^{H1}] & \in O\left(B(1 + \frac{p\ln B}{1-p})(1-p)\right) \\ & \in O(B(1-p) + Bp\ln B). \end{aligned} \quad (12)$$

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

4.3.2 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 acknowledgements than receivers in the A protocol. We can again use an analysis similar to the one by Pingali et al. [17, 18] for receivers using a sender-initiated protocol.

Y^{H1} = (receiving transmissions)
+ (sending local ACKs)

$$Y^{H1} = \sum_{i=1}^{L_h^{H1}} \Big(Y_p(i) + Y_h(i) \Big) + Y_f, \tag{13}$$

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

$$\mathrm{E}[L_r^{H1}] = \mathrm{E}[M^{H1}](1-p). \tag{14}$$

Taking expectations of Eq. 13 and substituting Eq. 14, we have

$$\begin{aligned}\mathrm{E}[Y^{H1}] &= \mathrm{E}[L_r^{H1}](\mathrm{E}[Y_p] + \mathrm{E}[Y_h]) + \mathrm{E}[Y_f] \\ &= \mathrm{E}[M^{H1}](1-p)\big(\mathrm{E}[Y_p] + \mathrm{E}[Y_h]\big) \\ &\quad + \mathrm{E}[Y_f]. \end{aligned} \tag{15}$$

Again, noting the bound of $\mathrm{E}[M^{H1}]$ given in Eq. 11,

$$\mathrm{E}[Y^{H1}] \in O(1 - p + p\ln B). \tag{16}$$

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

4.3.3 Group leaders

To evaluate the processing requirement at a group leader, 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 group leader is both a sender and receiver, we will express the costs in terms of X and Y. Our sum of costs is

H^{H1} = (receiving transmissions)
+ (sending local ACKs)
+ (collecting local ACKs)
+ (retransmissions)

$$\begin{aligned} H^{H1} &= \sum_{i=1}^{L_h^{H1}} \Big(Y_p(i) + Y_h(i) \Big) + Y_f + \sum_{k=1}^{L^{H1}} X_h(k) \\ &\quad + \sum_{m=2}^{M^{H1}} \Big(X_t(m) + X_p(m) \Big). \end{aligned} \tag{17}$$

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

$$\begin{aligned}\mathrm{E}[H^{H1}] &= \mathrm{E}[L_h^{H1}](\mathrm{E}[Y_p] + \mathrm{E}[Y_h]) + \mathrm{E}[Y_f] \\ &\quad + (\mathrm{E}[M^{H1}] - 1)(\mathrm{E}[X_p] + \mathrm{E}[X_t]) \\ &\quad + \mathrm{E}[L^{H1}]\mathrm{E}[X_h]. \end{aligned} \tag{18}$$

We can substitute Eqs. 6 and 14 into Eq. 18 to obtain

$$\begin{aligned}\mathrm{E}[H^{H1}] &= \mathrm{E}[M^{H1}](1-p)(\mathrm{E}[Y_p] + \mathrm{E}[Y_h]) + \mathrm{E}[Y_f] \\ &\quad + (\mathrm{E}[M^{H1}] - 1)(\mathrm{E}[X_p] + \mathrm{E}[X_t]) \\ &\quad + B\mathrm{E}[M^{H1}](1-p)\mathrm{E}[X_h]. \end{aligned} \tag{19}$$

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

$$\mathrm{E}[H^{H1}] = \mathrm{E}[Y^{H1}] + \mathrm{E}[X^{H1}] - \mathrm{E}[X_f] - \mathrm{E}[X_p]. \tag{20}$$

In other words, the cost on a group leader 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 Eqs. 12 and 16 into Eq. 20, we have

$$\begin{aligned}\mathrm{E}[H^{H1}] &\in O(1-p+p\ln B) \cup O(B(1-p) + Bp\ln B) \\ &\in O(B(1-p) + Bp\ln B). \end{aligned} \tag{21}$$

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

4.3.4 Overall system analysis

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

$$\Lambda^{H1} = min\{\Lambda_s^{H1}, \Lambda_h^{H1}, \Lambda_r^{H1}\}. \tag{22}$$

From Eqs. 12, 16, and 21 it follows that

$$1/\Lambda^{H1} \in O(B(1-p) + Bp\ln B). \tag{23}$$

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

$$1/\Lambda^{H1} \in O(B\ln B) = O(1)\ ;\ p \text{ constant}, \tag{24}$$

$$1/\Lambda^{H1} \in O(B) = O(1)\ ;\ p \to 0. \tag{25}$$

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.

4.4 Tree-based protocols with local NAK avoidance and periodic polling

To bound the overall system throughput in the generic Tree-NAPP protocol, we repeat the method used for the tree-based class; we first derive and bound the expected cost at the source, group leaders, and leaves. As we did for the case of tree-based protocols, we assume that there are enough receivers to form a full tree at each level. We place a superscript $H2$ on any variables relating to the generic Tree-NAPP protocol.

4.4.1 Source node

We consider first X^{H2}, the processing costs required by the source to successfully multicast an arbitrarily chosen packet to all receivers using the $H2$ protocol. The processing requirement for an arbitrary packet can be expressed as a sum of costs:

X^{H2} = (initial transmission) + (retransmissions)
+(receiving local NAKs)
+(receiving periodic local ACKs)

$$X^{H2} = X_f + \sum_{i=1}^{M^{H2}} X_p(i) + \sum_{m=2}^{M^{H2}} X_n(m) + BX_\phi, \quad (26)$$

where X_f is the time to get a packet from a higher layer, $X_p(i)$ is the time for (re)transmission attempt i, $X_n(m)$ is the time for receiving local NAK m from the receiver set, X_ϕ is the amortized time to process the periodic local ACK associated with the current congestion window, and M^{H2} is the number of transmission attempts the source will have to make for this packet. Taking expectations, where

$$\mathrm{E}[M^{H2}] = \mathrm{E}[M^{H1}], \quad (27)$$

we have

$$\mathrm{E}[X^{H2}] = \mathrm{E}[X_f] + \mathrm{E}[M^{H1}]\mathrm{E}[X_p] + (\mathrm{E}[M^{H1}] - 1)\mathrm{E}[X_n] + B\mathrm{E}[X_\phi]. \quad (28)$$

Using Eq. 11, the bound of $\mathrm{E}[M^{H1}]$, we can bound Eq. 28 as follows

$$\mathrm{E}[X^{H2}] \in O(1 + 1 + \frac{p}{1-p}\ln B) \in O(1 + \frac{p}{1-p}\ln B). \quad (29)$$

It then follows that, when p is a constant, $\mathrm{E}[X^{H2}] \in O(1)$.

4.4.2 Leaf nodes

Let Y^{H2} denote the processing requirement on nodes that do not have to forward packets (leaves). The sum of cost can be expressed as

Y^{H2} = (receiving transmissions)
+ (sending periodic local ACKs)
+ (sending local NAKs)
+ (receiving local NAKs)

$$Y^{H2} = \sum_{i=1}^{M^{H1}} (1-p)Y_p(i) + Y_f + Y_\phi + \sum_{j=2}^{M^{H1}} \left(\frac{Y_n(j)}{B} + (B-1)\frac{X_n(j)}{B}\right) + Prob\{M_r > 2\} \sum_{k=2}^{M_r - 1} Y_t(i). \quad (30)$$

Let $Y_p(i)$ be the time it takes to process the (re)transmission i, M_r be the number of transmissions required for the packet to be received by receiver r, $Y_n(j)$ be the time it takes to send local NAK j, $X_n(j)$ be the time it takes to receiver local NAK j (from another receiver), $Y_t(k)$ be the time to set timer k, Y_f be the time to deliver a packet to a higher layer, and Y_ϕ be the amortized cost of sending a periodic local ACK for a group of packets of which this packet is a member. Taking expectations of Eq. 30,

$$\mathrm{E}[Y^{H2}] = \mathrm{E}[M^{H1}](1-p)\mathrm{E}[Y_p] + \mathrm{E}[Y_f] + \mathrm{E}[Y_\phi] + (\mathrm{E}[M^{H1}]-1)\left(\frac{\mathrm{E}[Y_n]}{B} + (B-1)\frac{\mathrm{E}[X_n]}{B}\right) + \mathrm{Prob}\{M_r > 2\}(\mathrm{E}[M_r | M_r > 2] - 2)\mathrm{E}[Y_t]. \quad (31)$$

It follows from the distribution of M_r that [17, 18]

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

Therefore, noting Eq. 32 and that $\mathrm{Prob}\{M_r > 2\} = p^2$, we derive from Eq. 31 the expected cost as

$$\mathrm{E}[Y^{H2}] = \mathrm{E}[M^{H1}](1-p)\mathrm{E}[Y_p] + \mathrm{E}[Y_f] + \mathrm{E}[Y_\phi] + (\mathrm{E}[M^{H1}]-1)\left(\frac{\mathrm{E}[Y_n]}{B} + (B-1)\frac{\mathrm{E}[X_n]}{B}\right) + p^2\left(\frac{3-2p}{1-p} - 2\right)\mathrm{E}[Y_t]. \quad (33)$$

Again, using the bound of $\mathrm{E}[M^{H1}]$ given in Eq. 11, we can bound Eq. 33 by

$$\mathrm{E}[Y^{H2}] \in O\left(1 + \frac{1 - p + p\ln B + p^2(1-4p)}{1-p}\right). \quad (34)$$

When p is treated as a constant, $\mathrm{E}[Y^{H2}] \in O(1)$.

4.4.3 Group leaders

The sum of costs for group leaders, which have the job of both sender and receiver, is

H^{H2} = (receiving transmissions)
+ (sending periodic local ACKs)
+ (receiving periodic local ACKs)
+ (receiving local NAKs)
+ (sending local NAKs)
+ (retransmissions to children)

$$H^{H2} = (1-p)\sum_{i=1}^{M^{H1}} Y_p(i) + Y_\phi + BX_\phi + Y_f + \sum_{j=2}^{M^{H1}} \left(\frac{Y_n(j)}{B} + (B-1)\frac{X_n(j)}{B}\right) + \mathrm{Prob}\{M_r > 2\} \sum_{k=2}^{M_r - 1} Y_t(k) + \sum_{m=2}^{M^{H1}} (X_n(m) + X_p(m)). \quad (35)$$

Taking expectations and substituting Eq. 32, we obtain

$$\mathrm{E}[H^{H2}] = (1-p)\mathrm{E}[M^{H1}]\mathrm{E}[Y_p] + \mathrm{E}[Y_\phi] + B\mathrm{E}[X_\phi] + \mathrm{E}[Y_f] + (\mathrm{E}[M^{H1}]-1)\left(\frac{\mathrm{E}[Y_n]}{B} + (B-1)\frac{\mathrm{E}[X_n]}{B}\right)$$

$$+ p^2\left(\frac{3-2p}{1-p} - 2\right)\mathrm{E}[Y_t] + (\mathrm{E}[M^{H1}] - 1)(\mathrm{E}[X_n] + \mathrm{E}[X_p]). \quad (36)$$

Similar to group leaders in the $H1$ protocol, the processing cost at a group leader is the same as a source and a leaf, without the cost of receiving the data from a higher layer and one less transmission. Substituting Eq. 28 and Eq. 33 into Eq. 36 and subtracting the difference, the expected cost can be expressed as

$$\mathrm{E}[H^{H2}] = \mathrm{E}[Y^{H2}] + \mathrm{E}[X^{H2}] - \mathrm{E}[X_f] - \mathrm{E}[X_p]. \quad (37)$$

Therefore, Eq. 36 can be bounded by

$$\mathrm{E}[H^{H2}] \in O(\mathrm{E}[Y^{H2}]) \cup O(\mathrm{E}[X^{H2}]) \in O\left(1 + \frac{1-p+p\ln B + p^2(1-4p)}{1-p}\right). \quad (38)$$

When p is a constant, $\mathrm{E}[H^{H2}] \in O(1)$. Therefore, all nodes in the Tree-NAPP protocol have a constant amount of work to do with regard to the number of receivers.

4.4.4 Overall system analysis

The overall system throughput for the $H2$ protocol is the minimum throughput attainable at each type of node in the tree, that is,

$$\Lambda^{H2} = min\{\Lambda_s^{H2}, \Lambda_h^{H2}, \Lambda_r^{H2}\}. \quad (39)$$

From Eqs. 29, 34, and 38, it follows that

$$1/\Lambda^{H2} \in O\left(1 + \frac{1-p+p\ln B + p^2(1-4p)}{1-p}\right). \quad (40)$$

Accordingly, if either p is constant or $p \to 0$, we obtain from Eq. 40 that

$$1/\Lambda^{H2} \in O(1). \quad (41)$$

Therefore, the maximum throughput of the Tree-NAPP protocol, as well as the throughput with non-negligible packet loss, is independent of the number of receivers.

4.5 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 Sects. 4.3 and 4.4. Because we are interested in the maximum attainable throughput, we are assuming a constant stream of packets, which means we can ignore the overhead that occurs when there are no ACKs on which to piggyback token-passing messages.

4.5.1 Source

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

X^R = (initial transmission) + (processing ACKs) +(retransmissions)

$$X^R = X_f + X_p(1) + \sum_{i=1}^{L_r^U} X_a(i) + \sum_{m=1}^{M_r}\left(X_t(m) + X_p(m)\right), \quad (42)$$

where M_r is the number of transmissions required for the packet to be received by the token site, and has a mean of $\mathrm{E}[M_r] = 1/(1-p)$; 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:

$$L_r^U = \mathrm{E}[M_r](1-p) = 1. \quad (43)$$

Taking expectations of Eq. 42, we obtain

$$\mathrm{E}[X^R] = \mathrm{E}[X_f] + \mathrm{E}[M_r]\mathrm{E}[X_p] + (\mathrm{E}[M_r] - 1)\mathrm{E}[X_t] + \mathrm{E}[L_r^U]\mathrm{E}[X_a] = \mathrm{E}[X_f] + \frac{1}{1-p}\mathrm{E}[X_p] + \frac{p}{1-p}\mathrm{E}[X_t] + \mathrm{E}[X_a]. \quad (44)$$

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

$$\mathrm{E}[X^R] \in O\left(\frac{1}{1-p}\right), \quad (45)$$

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

4.5.2 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.)

T^R = (receiving transmission) + (multicasting ACK/token) + (processing NAKs) + (unicasting retransmissions)

$$T^R = Y_f + \sum_{i=1}^{L_r^U}\left(Y_p(i) + Y_a(i)\right) + \sum_{j=1}^{L^R} X_n(j) + (R-1)\mathrm{Prob}\{M_r > 1\}\sum_{m=1}^{M_r} X_p(m), \quad (46)$$

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

$$\mathrm{E}[L^R] = (R-1)(\mathrm{E}[M_r] - 1) = \frac{(R-1)p}{1-p}. \quad (47)$$

Therefore, the mean processing time at the token site is

$$\begin{aligned}\mathrm{E}[T^R] &= \mathrm{E}[Y_f] + \mathrm{E}[Y_p] + \mathrm{E}[Y_a] + \mathrm{E}[L^R]\mathrm{E}[X_n] \\ &\quad + (R-1)p\mathrm{E}[M_r]\mathrm{E}[X_p] \\ &= \mathrm{E}[Y_f] + \mathrm{E}[Y_p] + \mathrm{E}[Y_a] \\ &\quad + \frac{(R-1)p}{1-p}\left(\mathrm{E}[X_n] + \mathrm{E}[X_p]\right). \end{aligned} \tag{48}$$

The expected cost at the token site can be bounded by

$$\mathrm{E}[T^R] \in O\left(1 + \frac{(R-1)p}{1-p}\right), \tag{49}$$

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

4.5.3 Receivers

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

$$\begin{aligned} Y^R &= \text{(receiving ACK/token/time stamp)} \\ &\quad + \text{(receiving first transmission)} \\ &\quad + \text{(sending NAKs)} \\ &\quad + \text{(receiving retransmissions)} \\ Y^R &= Y_a + \mathrm{Prob}\{M_r = 1\}Y_p(1) + Y_f \\ &\quad + \mathrm{Prob}\{M_r > 1\}\sum_{i=1}^{L_r^U} Y_p(i) \\ &\quad + \mathrm{Prob}\{M_r > 1\}\sum_{m=2}^{M_r} Y_n(m) \\ &\quad + \mathrm{Prob}\{M_r > 2\}\sum_{n=3}^{M_r} Y_t(n). \end{aligned} \tag{50}$$

The first term in the above equation is the cost of receiving the ACK/token/time stamp 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

$$\begin{aligned}\mathrm{E}[Y_R] &= \mathrm{E}[Y_a] + (1-p)\mathrm{E}[Y_p] + \mathrm{E}[Y_f] \\ &\quad + p\mathrm{E}[L_r^U]\mathrm{E}[Y_p] \\ &\quad + p(\mathrm{E}[M_r|M_r > 1] - 1)\mathrm{E}[Y_n] \\ &\quad + p^2(\mathrm{E}[M_r|M_r > 2] - 2)\mathrm{E}[Y_t]. \end{aligned} \tag{51}$$

As shown previously [17, 18],

$$\mathrm{E}[M_r|M_r > 1] = \frac{2-p}{1-p}. \tag{52}$$

Substituting Eqs. 43, 52, and 32 into Eq. 51, we have

$$\begin{aligned}\mathrm{E}[Y^R] &= \mathrm{E}[X_a] + (1-p)\mathrm{E}[Y_p] + \mathrm{E}[Y_f] + p\mathrm{E}[Y_p] \\ &\quad + \frac{p}{1-p}\left(\mathrm{E}[Y_n] + p\mathrm{E}[Y_t]\right). \end{aligned} \tag{53}$$

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

$$\mathrm{E}[Y^R] \in O\left(\frac{1+p^2}{1-p}\right), \tag{54}$$

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

4.5.4 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:

$$\Lambda^R = min\{\Lambda_s^R, \Lambda_t^R, \Lambda_r^R\}. \tag{55}$$

From Eqs. 45, 49 and 54 it follows that, if p is a constant and for $p \to 0$, we obtain

$$1/\Lambda^R \in O\left(1 + \frac{(R-1)p}{1-p}\right) ;\ p \text{ constant}, \tag{56}$$

$$1/\Lambda^R \in O(1) ;\ p \to 0. \tag{57}$$

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 periodic costs X_ϕ and Y_ϕ which are set to 0.1. Figure 6 compares the relative throughputs of the protocols A, $N1$, $N2$, $H1$, $H2$, and R as defined in Sect. 2. The graph represents the inverse of Eqs. 19, 36, and 48, respectively, which are the throughputs for the tree-based, tree-NAPP, and ring-based protocols, as well as the inverse of the throughput equations derived previously [17, 18] for sender- and receiver-initiated protocols. The top, middle and bottom graphs correspond to increasing probabilities of packet loss, 1%, 10%, and 25%, respectively. Exact values of $\mathrm{E}[M^{H1}]$ were calculated using a finite version of Eq. 9; Exact values of $\mathrm{E}[M]$ were similarly calculated [22].

The performance of NAK avoidance protocols, especially tree-NAPP protocols, is clearly superior. However, our assumptions place these two subclasses at an advantage over their base classes. First, we assume that no acknowledgements 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 of the ACK tree, while 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.

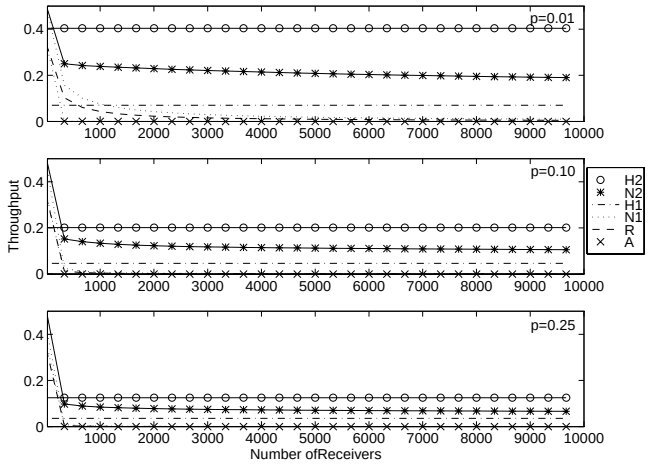


Fig. 6. The throughput graph from the exact equations for each protocol. The probability of packet loss is 1%, 10%, and 25%, respectively. The branching factor for trees is set at 10

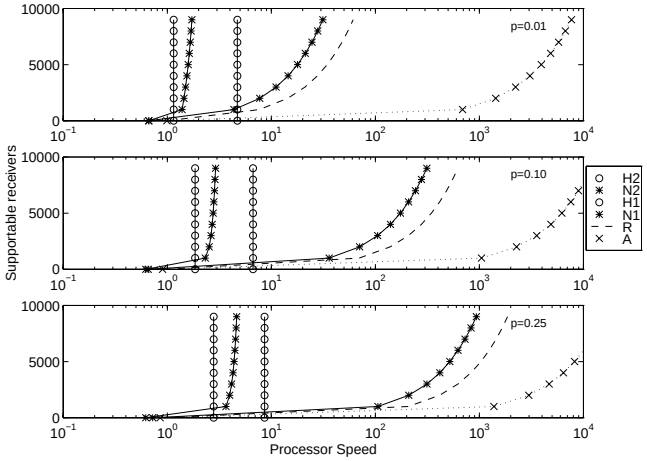


Fig. 7. Number of supportable receivers for each protocol. The probability of packet loss is 1%, 10%, and 25%, respectively. The branching factor for trees is set at 10

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* of NAK avoidance performance. 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 subclasses 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 even further. In addition, if NAK avoidance failed for a tree-NAPP protocol (e.g., due to incorrect setting of timers), 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 that some retransmissions must traverse a larger number of tree-hops towards 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. 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 et al. [12].

Figure 7 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. [17, 18]. 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 [17, 18]

$$\mathrm{E}[X^A]\Big|_{R=1} = \frac{1}{\mu^A[1]}\frac{3-p}{1-p} = \frac{3-p}{1-p}. \tag{58}$$

Using Eqs. 18, 36, 48, and 58, we can derive the following μs for tree-based, tree-NAPP, and ring-based protocols, respectively:

$$\begin{aligned}\mu^{H1}[R] &= \frac{1}{\mathrm{E}[X^A]}\mathrm{E}[H^{H1}] \\ &= \frac{1}{\mathrm{E}[X^A]}(\mathrm{E}[M^{H1}](1-p)(2)+1 \\ &\quad +(\mathrm{E}[M^{H1}]-1)(2)+B\mathrm{E}[M^{H1}](1-p)) \\ &= \frac{1}{\mathrm{E}[X^A]}(\mathrm{E}[M^{H1}](4+B-(2+B)p)-1)\,, \end{aligned} \tag{59}$$

$$\begin{aligned}\mu^{H2}[R] &= \frac{1}{\mathrm{E}[X^A]}\mathrm{E}[H^{H2}] \\ &= \frac{1}{\mathrm{E}[X^A]}((4-p)\mathrm{E}[M^{H1}]-1.9+0.1B \\ &\quad +p^2\left(\frac{3-2p}{1-p}-2\right), \end{aligned} \tag{60}$$

$$\begin{aligned}\mu^{R}[R] &= \frac{1}{\mathrm{E}[X^A]}\mathrm{E}[T^R] \\ &= \frac{1}{\mathrm{E}[X^A]}\left(1+1+1+\frac{(R-1)p}{(1-p)}(1+1)\right) \\ &= \frac{1}{\mathrm{E}[X^A]}\left(3+\frac{2(R-1)p}{(1-p)}\right)\,. \end{aligned} \tag{61}$$

The number of supportable receivers derived for sender- and receiver-initiated protocols are shown to be [17, 18]

$$\mu^{A}[R] = \frac{1}{\mathrm{E}[X^{A}]}\mathrm{E}[M](2 + R(1-p)), \quad (62)$$

$$\mu^{N1}[R] = \frac{1}{\mathrm{E}[X^{A}]}(1 + \mathrm{E}[M] + Rp/(1-p)), \quad (63)$$

$$\mu^{N2}[R] = \frac{1}{\mathrm{E}[X^{A}]}(2\mathrm{E}[M]). \quad (64)$$

From Fig. 6 and 7, it is clear that tree-based protocols can support any number of receivers for the same processor speed bound at each node, and that tree-NAPP protocols attain the highest maximum throughput. It is also important to note that the maximum throughput that RINA protocols can attain becomes more and more insensitive to the size of the receiver set as the probability of error decreases. Because we have assumed that a single NAK reaches the source, that NAKs are never lost, and that session messages incur no processing load, we implicitly assume the optimum behavior of RINA protocols. The simulation results reported for SRM by Floyd et al. [8] agree with our model and result from assuming no NAK losses and a single packet loss in the experiments. Figure 7 shows that tree-NAPP protocols can be made to perform better than the best possible RINA protocol by limiting the size of the local groups.

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. 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 effects of ACK or NAK failure, and the correlation of failures along the underlying multicast routing trees are accounted for, the same relative performance of protocols will be observed.

The results are summarized in Table 1. It is already known that sender-initiated protocols are not scalable because the source must account for every receiver listening. Receiver-initiated protocols are far more scalable, unless NAK avoidance 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 a limited set of applications. Furthermore, to set the timers needed for NAK avoidance, existing instantiations of RINA protocols require all group members to transmit session messages periodically, which makes them unscalable. 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 [19] were used.

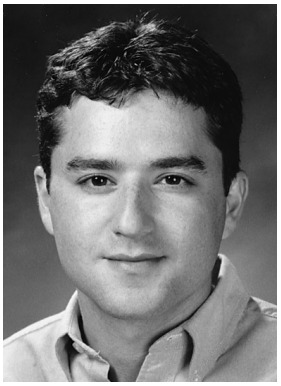
Our analysis shows that ACK trees are a good answer to the scalability problem for reliable multicasting. Practical implementations of tree-based protocols maintain the anonymity of the receiver set, and only the tree-based and tree-NAPP classes have throughputs that are constant with respect to the number of receivers, even when the probability of packet loss is not negligible (which would preclude accurate setting of NAK avoidance timers). 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 with all the receivers of a session 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, ACKs, and NAKs are received correctly.

On the other hand, while the scope of NAKs and retransmissions can be reduced without establishing a structure in the receiver set [8], limiting the scope of the session messages needed to set NAK avoidance timers and to contain the scope of NAKs and retransmissions require the aggregation of these messages. This leads to organizing receivers into local groups that must aggregate sessions messages sent to the source (and local groups). Doing this efficiently, however, leads to a hierarchical structure of local groups much like what tree-based protocols require. Hence, it appears that organizing the receivers hierarchically (in ACK trees or otherwise) is a necessity for the scaling of a reliable multicast protocol.

References

1. Ballardie A, Francis P, Crowcroft J (1993) Core based trees (CBT): An architecture for scalable inter-domain multicast routing. In: Proc. ACM SIGCOMM'93 (October 1993), pp 85–95. San Francisco, CA, USA, 13–17 September 1993
2. Bertsekas D, Gallager R (1992) Data Networks, second ed. Prentice Hall, Englewood cliffs, New Jersey
3. Chang J-M, Maxemchuk NF (1984) Reliable broadcast protocols. ACM Trans Comput Syst 2(3): 251–273
4. Clark DD, Lambert ML, Zhang L (1987) NETBLT: A high-throughput transport protocol. In: Proc ACM SIGCOMM'93 (August 1987), pp 353–359. San Francisco, CA, USA, 13–17 September 1993
5. Deering S (1991) Multicast routing in a datagram internetwork. PhD thesis, Stanford University, Palo Alto, Calif.
6. Deering S, Cheriton D (1990) Multicast routing in datagram internetworks and extended lans. ACM Trans Comput Syst 8(2): 85–110
7. Deering S, et al. (1994) An architecture for wide-area multicast routing. In: Proc. ACM SIGCOMM'94, pp 126–135, London, UK, 31 August–2 September 1994
8. Floyd S, et al. (1995) A reliable multicast framework for light-weight sessions and application level framing. In: Proc. ACM SIGCOMM'95, pp 342–356. Cambridge, MA, USA, 28 August–1 September 1995
9. Holbrook H, Singhal S, Cheriton D (1995) Log-based receiver-reliable multicast for distributed interactive simulation. In: ACM SIG-

COMM'95, pp 328–341. Cambridge, MA, USA, 28 August–1 September 1995
10. Postel JB (ed) (1981) Transmission control protocol. Request for Comments 793
11. Levine BN, Garcia-Luna-Aceves JJ (1996) A comparison of known classes of reliable multicast protocols. In: Proc. IEEE International Conference on Network Protocols (October 1996). Columbus, OH, USA, 29 October–1 November 1996. Ural H (ed) Los Alamitos, CA, USA. IEEE Comput Soc Press, pp 112–121
12. Levine BN, Lavo D, Garcia-Luna-Aceves JJ (1996) The case for reliable concurrent multicasting using shared ack trees. In: Proc. ACM Multimedia, pp 365–376, 18–22 November 1996. Boston, MA, USA
13. Lin J, Paul S (1996) RMTP: A reliable multicast transport protocol. In: Proc. IEEE Infocom, pp 1414–1425. San Francisco, CA, USA, 24–28 March 1996. Los Alamitos, CA, USA. IEEE Comput Soc Press 3: 1414–1424
14. Parsa M, Garcia-Luna-Aceves JJ (1997) A protocol for scalable loop-free multicast routing. IEEE J Sel Areas Commun 15(3): 316–331
15. Paul S, Sabnani K, Kristol D (1994) Multicast transport protocols for high-speed networks. In: International Conference on Network Protocols, pp 4–14. Boston, MA, USA, 25–28 October 1994. Los Alamitos, CA, USA. IEEE Comput Soc Press
16. Paul S, Sabnani KK, Lin JC, Bhattacharyya S (1997) Reliable multicast transport protocol (RMTP). IEEE J Sel Areas Commun 15(3): 407–421
17. Pingali S (1994) Protocol and Real-Time Scheduling Issues for Multimedia Applications. PhD thesis, University of Massachusetts, Amherst, Mass.
18. Pingali S, Towsley D, Kurose J (1994) A comparison of sender-initiated and receiver-initiated reliable multicast protocols. Perform Evaluation Rev 22: 221–230
19. Ramakrishnan S, Jain BN (1987) A negative acknowledgement with periodic polling protocol for multicast over lan. In: Proc. IEEE Infocom, pp 502–511. San Francisco, CA, USA, 31 March–2 April 1997. Washington, DC, USA. IEEE Comput Soc Press
20. Shields C, Garcia-Luna-Aceves JJ (1997) The ordered core-based tree protocol. In: IEEE Infocom'97, pp 884–891
21. Strayer T, Dempsey B, Weaver A (1992) XTP: The Xpress Transfer Protocol. Addison-Wesley, Reading, Mass.
22. Towsley D, Kurose J, Pingal S (1997) A comparison of sender-initiated and receiver-initiated reliable multicast protocols IEEE J Sel Areas Commun 15(3): 398–406
23. Whetten B, Kaplan S, Montgomery T (1994) A high-performance, totally ordered multicast protocol. In: Theory and Practice in Distributed Systems, International Workshop, LNCS 938 (September 1994). Dagstuhl Castle, Germany, 5–9 September 1994. Birman KP, Mattern F, Schiper A (eds) Berlin, Springer 1995, pp 33–57
24. Yavatkar R, Griffioen J, Sudan M (1995) A reliable dissemination protocol for interactive collaborative applications. In: Proc. ACM Multimedia, pp 333-344. San Francisco, CA, November 5–9



BRIAN NEIL LEVINE is a PhD candidate in Computer Engineering at the University of California, Santa Cruz (UCSC). In 1994, he received his B.S. in Applied Mathematics and Computer Science from the State University of New York at Albany. In 1996, he received his M.S. in Computer Engineering from the University of California at Santa Cruz. His current research interests include multicast routing and reliable multicast protocols.

J.J. GARCIA-LUNA-ACEVES was born in Mexico City, Mexico on October 20, 1955. He received the B.S. degree in electrical engineering from the Universidad Iberoamericana, Mexico City, Mexico, in 1977, and the M.S. and Ph.D. degrees in electrical engineering from the University of Hawaii, Honolulu, HI, in 1980 and 1983, respectively. He is Professor of Computer Engineering at the University of California, Santa Cruz (UCSC). Prior to joining UCSC in 1993 as an Associate Professor, he was a Center Director at SRI International (SRI) in Menlo Park, California. He first joined SRI as an SRI International Fellow in 1982. His current research interest is the analysis and design of algorithms and protocols for computer communication. At UCSC, he leads a number of research projects sponsored by DARPA and industry that focus on wireless networks and internetworking. Dr. Garcia-Luna-Aceves has coauthored the book Multimedia Communications: Protocols and Applications (Prentice-Hall), and has published more than 120 referred papers on computer communication in journals and conferences. He is on the editorial boards of the IEEE/ACM Transactions on Networking, the ACM Multimedia Systems Journals, and the Journal of High Speed Networks. Dr. Garcia-Luna-Aceves has been Chair of the ACM special interest group on multimedia, General Chair of the first ACM conference on multimedia: ACM MULTIMEDIA '93, Program Chair of the IEEE MULTIMEDIA '92 Workshop, General Chair of the ACM SIGCOMM '88 Symposium, and Program Chair of the ACM SIGCOMM '87 Workshop and the ACM SIGCOMM '86 Symposium. He has also been program committee member for numerous IFIP 6.5, ACM, IEEE, and SPIE conferences on computer communication. He received the SRI International Exceptional-Achievement Award in 1985 for his work on multimedia communications, and again in 1989 for his work on adaptive routing algorithms. He is a member of the ACM and the IEEE.