Chapter 3

Background and Related Work

Reliable multicast has been a topic of great interest for many years, and therefore a large amount of work has been done in the area. Recently, however, researchers turned their attention to the problems of large scale multicast, where groups can have receiver populations ranging from a few to hundreds of thousands. Such applications require highly scalable multicast error control.

In this chapter, we begin by presenting a brief but essential background on error control. We then proceed to describe the problems encountered when we apply traditional error control techniques to large-scale multicast. Next, we describe related work, which we divide in two parts: the first is a general overview of the literature, where we categorize proposed solutions into two categories, one for solutions that do not require network assistance and another for solutions that do. In the second part of our overview we describe in more detail two prominent error control solutions, both of which we used in the simulation comparison with our work.

3.1. Background: Positive vs. Negative Acknowledgments

Traditionally, data recovery is done using positive acknowledgments. In general, recovery works as follows: imagine a sender and a receiver, wishing to share data reliably. The simplest method to achieve reliability is by using the *stop and wait protocol*. With this protocol, the sender transmits a packet, and sets a retransmission timer. The receiver, upon receiving the packet, returns a *positive acknowledgment* notifying the sender of the successful reception of a the packet. Upon reception of the acknowledgment, the sender cancels the retransmission timer and sends the next packet. If either the data packet or the acknowledgment is lost, the sender's timer expires, the

packet is retransmitted and a new timer is set. The process repeats until the sender transmits (and receives acknowledgments) for all its data. In practice, in order to improve efficiency, protocols like TCP use a window - which allows multiple packets to be in transit before receiving an acknowledgment - appropriately called a *window-based error control* mechanism.

This is a very simplified description of window error control (more details can be found in any of the numerous networking books, including [1, 2]). What is important, however, is that with this type of error control, the sender is responsible for detecting loss and sending retransmissions. The receiver's job is relatively much simpler: upon receiving of new packet, it acknowledges the sequence number of the highest consecutive packet it has received. Because the sender does most of the important work, this approach is known as a *sender-reliable* approach[36].

We saw that the sender-reliable approach, uses positive acknowledgments. Another approach, which uses *negative acknowledgments* and puts the burden of recovery on the receiver, is appropriately called a *receiver-reliable* approach[36]. In a receiver reliable approach, the receiver is responsible for detecting loss by keeping track of the sequence number of arriving packets. A gap in the sequence number indicates a packet loss; for example, the reception of packet *n* followed by packet *n+2* signifies that packet *n+1* may have been lost[1]. Upon detection of a gap, the receiver sends back to the sender a negative acknowledgment, i.e., a message requesting the retransmission of packet *n+1*. After sending the negative acknowledgment, the receiver sets a timer, waiting for the retransmission to arrive. If the retransmission fails to arrive before the timer expires, another negative acknowledgment is sent and a new timer is set; the process repeats until the packet is received successfully. In a receiver-reliable approach, it is the sender's job that is now much simpler: it responds to each negative acknowledgment by sending a retransmission.

Although it appears that both sender-reliable and receiver-reliable approaches have equivalent functionality (namely the reliable transmission of data), they have some rather significant differences. In addition to placing the burden of recovery at different ends, the two mechanisms differ in a more fundamental manner: with the sender-reliable approach, both the sender and the receiver are notified that a packet was successfully delivered (a process that happens continuously, as data

1. While we ignore re-ordering in our discussion, note that gap-detection can still be used in the presence of re-ordering if the receiver defines how many out-of-sequence packets it can receive before it declares loss.

is transmitted). With the receiver-reliable approach, however, the absence of negative acknowledgments is ambiguous: it either means that the receiver has received all packets successfully, or that negative acknowledgments are getting lost. Therefore, with a receiver-reliable protocol, the sender can never be certain that the receiver has received all packets. Among other things, this has significant implications on buffer allocation: the sender must either provide infinite buffers, which is not practical in most cases, or make an independent decision (and risk being wrong) about when to purge and reclaim its retransmission buffers. If buffers are purged early, some data may be lost forever. One way to overcome this limitation is to have the receiver periodically send positive acknowledgments.

To summarize, receiver-reliable approaches (i.e., those using only negative ACKS) cannot provide 100% reliability; only sender-reliable approaches (i.e., those employing positive ACKS) can make such guarantees. However, receiver-reliable approaches, have merit: because they place the burden of recovery at the receivers, are better suited for reliable multicast, as we will see next.

3.2. Problems with Reliable Multicast

One could argue that the sender-reliable approach can easily be applied to small multicast groups of perhaps a dozen receivers or so. Modern hosts and networks have adequate capacity to accommodate the extra processing, state, and positive acknowledgments that would be required. However, for multicast groups that can scale to hundreds or thousands of receivers, it is clear that such solutions are hopelessly non-scalable. A sender serving thousands of receivers would be forced to maintain state, receive and process an acknowledgment from each receiver and for every packet (or window) it sends out. This not only would create a huge load at the sender, but may also create congestion in the network as the acknowledgments funnel back towards the sender. The problem became apparent very early, and thus the vast majority of reliable multicast protocols today have adopted receiver-reliability with negative acknowledgments, which reduces receiver feedback from one message per packet sent per receiver, to one message per packet lost per affected receiver. While initially this appears to significantly reduce feedback from receivers, it is still far from adequate to ensure scalability. The reasons are discussed next.

3.2.1. The Error Model

First, let us define the error model we will be using in our solution. We assume that if a packet gets lost, all receivers downstream of the loss miss the packet. The loss of one or more consecutive packets constitutes a loss event. Receivers detect these events when they see a gap. Typically, one request and one retransmission is required to recover from a loss event (assuming no further loss). If the same packet (or burst) is lost at different places independently, then we regard this as separate loss events and an independent process must be initiated to recover from such losses.

3.2.2. Implosion

We already discussed why reliable multicast protocols have adopted negative acknowledgments in an effort to achieve scalability. Here we show why this is not sufficient to make error control scalable. We begin with a problem known as *implosion*, caused by receiver feedback.

Recall that a negative acknowledgment is sent by each receiver that detects a gap in the sequence number of received packets. Also recall that with multicast, the sender sends a single packet addressed to the group, which is replicated only at the branching points in the multicast delivery tree. Figure 3.1 shows such a multicast tree with a large number of receivers. Now suppose that the “X” marks a link where a packet is dropped. Note that this link serves most all the receivers in the group. The result of this single drop is that every receiver downstream the link misses the packet. When the next packet gets through, the resulting gap causes each receiver to send a negative acknowledgment back to the sender. As one can imagine, the outcome is disastrous: a single packet loss has caused a negative acknowledgment from almost every receiver in the group, resulting in *NACK* implosion at the sender.

The problem is complicated because its magnitude depends on the location where the packet is dropped. In the best case, if the packet is dropped on a link serving a single receiver, only one NACK would be generated. However, at the worst case, if a packet is dropped near the source, all receivers will lose it. Since it is impossible to predict where a packet is dropped, an error control scheme must be capable of dealing with all possible loss scenarios, from loss at a single receiver to loss by the entire group.

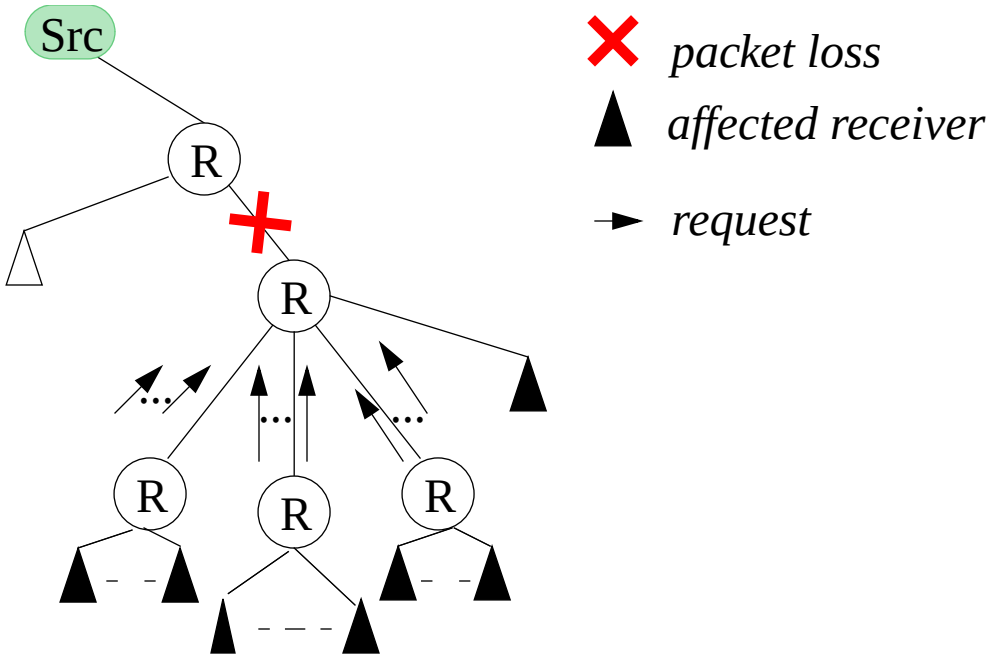

Figure 3.1: A single packet drop creates NACK Implosion.

3.2.3. Exposure

The problem of implosion is caused by receiver feedback to the sender. Implosion, however, is only one side of the problem. Even if we had the means to notify the sender of a packet loss in a scalable manner (i.e., without implosion), another problem remains, which pertains to the manner retransmissions are delivered to the receivers. This problem is *exposure*.

Let us examine the scenario in Figure 3.2. The topology is the same as in the previous figure, except that loss now affects only one receiver. The affected receiver sends a NACK back to the sender, the sender prepares a retransmission and is about to send it. The sender has two choices at this point:

- *Unicast*: send the retransmission via unicast to the affected receiver
- *Multicast*: multicast the retransmission to the entire group.

Although at first glance the first option seems to be the best, recall that in general a packet loss may affect any number of receivers, from a single receiver to the entire group. Therefore, in cases

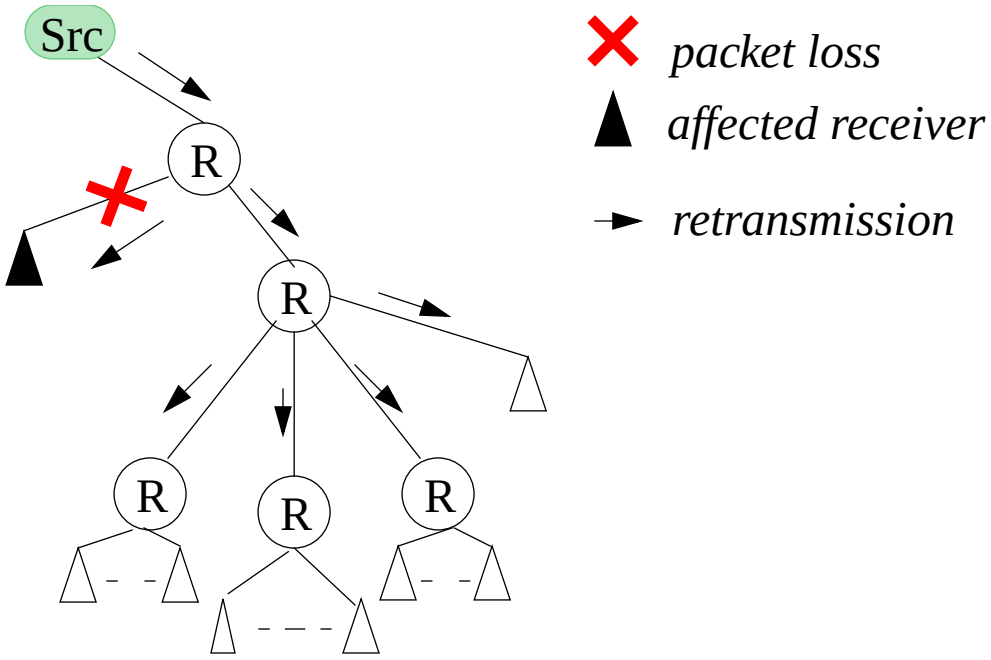

Figure 3.2: Loss at a single receiver causes exposure

where the entire group misses a packet, unicasting the retransmission to each receiver is impractical; the second approach (multicast to entire group) is definitely better. However, adopting the second approach means that when only one receiver loses a packet all receivers will be forced to receive a retransmission. This may lead to the "crying baby problem", where one receiver behind a lossy link causes excessive retransmissions to the entire group. Therefore, we conclude that neither approach is acceptable.

A hybrid approach has been suggested, that employs a threshold to select between unicast and multicast. With this approach, the sender waits for some amount of time (typically the maximum RTT in the group) while collecting NACKs. If the number of NACKs is below the threshold, then retransmissions are unicast to each receiver; if the number exceeds the threshold, the retransmission is multicast. Clearly, while this approach is an improvement, is still not scalable. As the number of receivers grows very large, selecting the threshold becomes increasingly difficult. A reasonable trade-off between implosion and exposure may be to select a threshold value as a function of the number of receivers. In practice, however, determining the number of receivers is difficult (recall that IP Multicast does not keep track of group membership). In addition, the threshold

may be limited to far less than half the receiver population because of sender's NACK processing capability may be low.

To summarize, exposure is the problem that occurs when recovery messages are delivered to receivers that do not need them. As the group gets very large, the probability that a packet is lost by one or more receivers (and hence retransmission is required) increases and a large number of recovery messages are generated. If these messages are not contained and exposure is high, scalability will suffer because unwanted packets will not only incur necessary processing at the receivers, but will also waste network bandwidth and possibly lead to congestion.

3.3. Overview of Related Work

There has been a significant amount of research on reliable multicast protocols. The early work has focused on distributed systems, providing primitives for constructing distributed applications, such as the ISIS system[67] and the V-kernel[68]. Other early work has focused on local area networks or broadcast links [69, 70, 71, 72, 73]. We will not cover the early work here; a good survey can be found in [64]. We will focus on recent work on reliable multicast that aims to provide scalability to very large groups.

As we described earlier, the vast majority of reliable multicast protocols use receiver-reliable recovery which was shown by Pingali, Towsley, and Kurose to be superior to sender-reliable recovery [36]. We begin our overview with reliable multicast schemes that require no assistance from routers. We we call these *non-assisted schemes*. Then, we proceed to list schemes that require network assistance, which we call *assisted schemes*. We will continue our overview of related work with a more detailed look at SRM and PGM, the two schemes that we use in our comparison with our scheme.

3.3.1. Non-Assisted Schemes

Many of the non-assisted schemes are hierarchical schemes, which organize receivers in a tree. Each receiver is assigned a parent and zero or more children. Request implosion is controlled by allowing requests from children to their parents only. Duplicate replies and exposure are reduced by either unicasting retransmissions from parents to children or multicasting after some threshold

of requests is exceeded. Parent discovery is a crucial step in hierarchical schemes. Some schemes are static, i.e., the parent/children allocation is fixed. Others are dynamic, and allow members to reorganize the tree as the group topology changes. Dynamic schemes are more flexible but require more complex parent discovery mechanisms.

The Reliable Multicast Transport Protocol (RMTP) [35] is an example of a static hierarchical scheme. The source multicasts data to all receivers, but only the *Designated Receivers* (DRs) return acknowledgments. Losses in RMTP are recovered from DRs. Retransmissions are either unicast or multicast depending on how many requests were received. This, however, is a crude solution because it performs well only at the extremes (if there are too many or very few losses). Otherwise, it incurs significant overhead, either in terms of network traffic or exposure. Although not implemented, RMTP was the first protocol to propose the use of *subcast*[1], a router service that allows a router to multicast a packet to all downstream links.

The Log-Based Receiver-reliable Multicast (LBRM) [32] is another example of a static hierarchical scheme, aimed at distributed interactive simulation (DIS) applications. LBRM uses a primary logging server and a static hierarchy of secondary logging servers which log all transmitted data. Data is multicast from the source to all logging servers and all receivers; however, only the primary logging server returns acknowledgments to the source. The receivers request lost data from the secondary logging servers; in turn, the secondary logging servers request any lost data from the primary logging server. Similar to RMTP, retransmissions in LBRM are either unicast or multicast, or multicast based on a threshold. Both RMTP and LBRM are based on a static hierarchy and thus require explicit set-up of DRs or logging servers before new regions can be added to the group.

The Tree-based Multicast Transport Protocol (TMTP) [40] is an example of a scheme that uses a dynamic hierarchy. In TMTP, every region has a Domain Manager (DM). When a DM joins a group, it searches for a parent using an expanding ring search. During the search, the new DM repeatedly broadcasts a “SEARCH_FOR_PARENT” request by increasing the time-to-live (TTL) value. When one or more DMs respond, the new DM selects the closest DM as its parent. Thus, the DMs form a dynamic hierarchical control tree. Each endpoint maintains the hop distance to its

1. Term coined by Adam Costello.

DM, and each DM maintains the hop distance to its farthest child. These values are used to set the TTL field on requests and replies to limit their scope. To further limit request implosion at the DMs, TMTP uses randomized backoff for requests, which, however, increases latency.

LGMP [74] is a hierarchical, subgroup-based protocol, where receivers take the responsibility of dynamically organize themselves into subgroups. Subgroups select a Group Controller to coordinate local retransmissions and process feedback messages. LGMP subgroups are self-organizing and self-adaptive according to the current network load and group membership. In LGMP subgroups may not always achieve congruency. LGMP has been implemented and some of its testing was carried out on the MBONE.

TRAM [76] is another dynamic tree-based protocol designed to support bulk data transfer. TRAM uses TTL to form the receiver tree. The tree formation and maintenance algorithms borrow from other schemes like TMTP, but TRAM has a richer tree management framework, supporting member repair and monitoring, pruning of unsuitable members, and aggregation and propagation of protocol related information.

MFTP [75] is designed for reliable distribution of files to a large number of receivers. Data is transmitted in passes. After each pass, receivers unicast NACKs back to the sender using random back-off delay to avoid implosion. The sender collects all NACKs and transmits all missing packets in the next pass. The process repeats until all receivers receive the data and no NACKs are sent. It is clear that MFTP trades latency for reliability, a trade-off which is acceptable for file transfer, but may not be acceptable for other applications.

In summary, static hierarchical schemes like RMTP and LBRM do not adapt to rapid membership changes or changes in topology. Dynamic hierarchical schemes like TMTP, LGMP, and TRAM rely on approximate methods (e.g., expanding ring search) to discover parents and send replies. The use of expanding ring search for parent selection can lead to other forms of suboptimality, due to lack of congruency between the recovery tree and the underlying topology. Other schemes like MFTP are mostly suited for bulk data transfer.

3.3.2. Schemes Using FEC

We briefly touched on Forward Error Correction [8] in Chapter 2. FEC is attractive in a multicast environments with a high degree of uncorrelated loss, because such losses can be repaired efficiently. FEC typically increases the bandwidth required to transmit data, depending on the encoding method used. Recently, techniques have been proposed that reduce this overhead and increase the effectiveness of FEC [78, 79, 77]. We chose to investigate retransmission in our work because it offers very low cost in terms of bandwidth and does not require encoding/decoding of data at the ends. Some of the techniques we have devised in our work can be used with FEC solutions, for example in sending scoped parity packets.

3.3.3. Assisted Schemes

In the last few years, there have been several proposed schemes that use network assistance for reliable multicast, which we describe below. Most of these postdate our work.

In Addressable Internet Multicast (AIM)[33], the authors propose to extend Internet routing by defining a rich set of services. These services require routers to assign per-multicast group labels to all routers participating in that group. There are three types of labels: positional, distance, and stream labels. Positional labels are used to route messages to individual members of the group. Distance labels are used to locate near-by members. Stream labels are used to subscribe to traffic generated by a subset of sources. AIM defines new routing mechanisms based on the presence of these labels. These mechanisms are:

- Positional routing: route a message to a particular destination router
- Reachcast: route to the closest router that has a member belonging to the group
- Positional reachcast: route to the closest router that has a member belonging to the group but towards some destination
- Reverse reachcast: route to the routers that can reach current router with a reachcast
- Reverse positional reachcast: route to the routers that can reach current router with a positional reachcast.

One application of the above mechanisms is the Reliable Multicast Architecture (RMA). In RMA, members requiring a retransmission ask their local router to send a request using a positional reachcast towards the source. A reachcast eventually reaches members that have the requested data, which respond by sending a retransmission via positional routing. The proposed labeling scheme has less overhead when used in shared trees. If used in source-based trees, each source tree requires its own labels. The overhead of distributing the labels after a membership change can be high if groups are highly dynamic: whenever a new branch is added to the multicast tree, all the routers below the new branch may have to change their labels.

Search Party [30] builds on our work by aiming to enhance robustness. In Search Party, requests are not routed deterministically, as in LMS, but randomly using a new mechanism called “randomcast”. This mechanism is used by routers to randomly route requests to either the parent or to one of the children. Search Party trades efficiency (in terms of increased latency and duplicates) for better robustness.

OTERS [80], uses a modified version of the mtrace[66] utility to construct a recovery tree that is congruent with the underlying multicast tree. OTERS builds the tree by incrementally identifying subroots in the multicast routing tree using back-tracing. For each subroot, OTERS selects a Designated Receiver (DR) which acts as the parent. OTERS solves the problem of maintaining congruency (in other words, ensuring that the recovery tree mirrors the underlying multicast tree), but receivers are still exposed to topology and have to keep track of changes in the structure of the underlying multicast group. In addition, the overhead of using mtrace probes may be high in highly dynamic groups.

Tracer [65] is similar to OTERS in that it also uses the mtrace utility to allow each receiver to discover its path to the source. Once the path is discovered, receivers advertise their paths to nearby receivers using expanding ring search. Once receivers discover nearby receivers, they use the data from tracing and their loss rate to select parents. Tracer can be used as a facility to create congruent trees for other tree-based protocols, such as RMTP. As with OTERS, Tracer exposes receivers to the underlying topology of the group and incurs overhead due to mtrace probes.

3.4. SRM

We now proceed to describe in detail the two schemes that as discussed in the following chapters, we have simulated and compared with our scheme. We start with Scalable Reliable Multicast (SRM) [17].

SRM employs two clever global mechanisms to limit the number of messages generated, namely duplicate suppression and back-off timers. In SRM, recovery messages (requests and replies) are multicast to the entire group; receivers listen for recovery messages from other receivers before sending their own, and suppress their recovery messages if they would duplicate one already seen. The intended goal is to allow the multicast of only one recovery message. In order to increase the effectiveness of the suppression mechanism, especially in densely packed groups, the round-trip-time between receivers is artificially enlarged (for recovery messages only) with the addition of back-off delay. To improve performance, the added delay consists of a fixed and a random component, calculated separately at each receiver. The fixed component is based on the distance of the receiver to each sender, and the random component is based on the density of the receivers in the neighborhood. However, these components have to be re-calculated when group membership, topology, or network conditions change, meaning that SRM needs time to adapt to improve performance.

SRM performs well in suppressing requests but slightly worse in suppressing replies. However, SRM has the following disadvantages:

- The backoff delay for requests is set to some multiple of the unicast delay to the sender. Thus, on average, recovery delay will be higher than unicast.
- The randomization only ensures a unique requestor or replier with a certain probability. In topologies where the distance based tiebreaker is ineffective (e.g., a star), an unfortunate trade-off must be made. Using large random numbers can make the probability of a unique requestor or replier high but increase the recovery latency; using small random numbers can make latency small but increase the probability of duplicates.

- The “multicast to everyone” approach provides excellent fault tolerance, but also exposes recovery to *all* members of the multicast group. This situation is compounded if multiple requestors and repliers respond.
- A new receiver joining the group must measure the propagation delay to every existing receiver in the group in case the new receiver is elected as a replier. Also, if adaptive timers are used, several request-reply rounds are needed before timers stabilize.

Simulation results on random topologies with fixed timer values [17] show that SRM typically requires about 2-3 times the unicast round-trip delay to recover a lost packet and produces around 2 - 10 duplicates in the process. The SRM designers have proposed an algorithm to adapt timers to improve performance. Using adaptive timers reduces the number of duplicates after the timers are tuned. However, timer adaptation can be a slow process, and there may be transients when loss shifts from one location to another.

To avoid multicasting all messages to all members, SRM proposes the use of the TTL field in the IP header to limit the scope of recovery messages. However, this approach limits the scope of messages within a radius, while losses affect a subtree. Thus, it still allows duplicates to reach other regions, as shown in Figure 3.3.

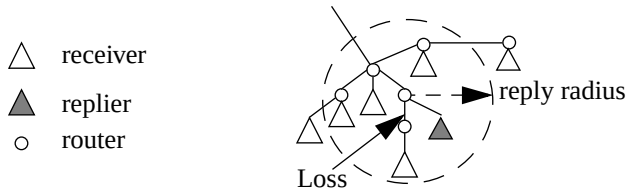

Figure 3.3: Scoping with TTL is not always effective

In [81] two approaches to enhance SRM with local recovery have been studied: one requires receivers to maintain the hop-count between themselves and all other receivers; the other suggests the use of multiple multicast groups. However, maintaining the hop-count to each receiver is a costly approach and prone to errors. Creating multicast groups for the sole purpose of recovery is a slow process, and multiplies the overhead associated with each multicast session, since creating and pruning a group is currently relatively expensive.

3.5. PGM

PGM [47] is a reliable multicast protocol marketed by the router company Cisco. PGM is a network-assisted scheme, that unlike the schemes we described earlier, peeks into the transport layer and requires per-lost-packet state at the routers.

The basic operation of PGM is depicted in Figure 3.4, and is as follows: Upon detection of

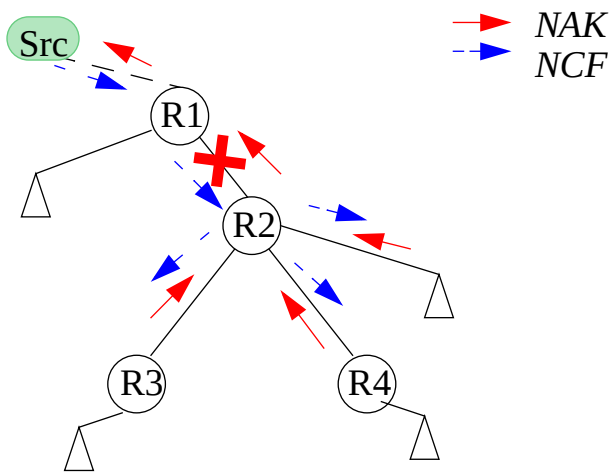

Figure 3.4: PGM operation

loss, every PGM receiver selects a short random back-off interval and then unicasts a negative acknowledgment (NAK in PGM terminology) to its upstream PGM router. Upon reception of a NAK, the upstream router performs the following steps:

- multicasts a NAK confirmation (NCF) on the link the NAK was received on to suppress other NAKs.
- creates retransmit state noting the sequence number of the requested data and the link the NAK was received on.
- repeatedly unicasts the NAK to its upstream router until a corresponding NCF is received.

Both NAKs and NCFs are examined hop-by-hop by the PGM routers. If similar NAKs arrive from other downstream links, these links are also added to the list. However, no additional NAKs

are propagated upstream. This effectively controls implosion by allowing only one NAK per lost packet to reach the source.

When the source receives a NAK, it responds with a retransmission (RDATA). As with NAKs and NCFs, RDATA is examined hop-by-hop by the PGM routers. Each router forwards RDATA on links for which is has previously established state (i.e., links where a NAK was received for that retransmission). Thus, RDATA packets follow the path laid out by previous NAKs, and consequently reach only those receivers that have sent a NAK. This completely eliminates exposure. After a router forwards RDATA, it discards the corresponding NAK state.

In PGM, all retransmissions originate from the source. Provision is made for suitable receivers to act as Designated Local Retransmitters (DLRs) but in order for a receiver to become a DLR it must lie directly on the path towards the source. This severely limits the deployment of DLRs. When a DLR is deployed, its operation is similar to the source.

PGM in its current specification, faces the following problems:

The dangling NAK state: if a NAK or RDATA is lost, previous NAK state is not discarded at the routers until the NAK state expires. Thus, when receivers that fail to receive RDATA timeout and send a NAK again, these NAKs are blocked by the routers. The reason is that PGM routers block duplicate NAKs from being propagated upstream while NAK state is present. Since NAK state is normally erased by passing RDATA which did not arrive, the state is still present until it times out. The NAK state expiration interval, however, is just a soft-state safeguard to eliminate stale state, and thus is typically large (several seconds). The solution proposed by the PGM designers is to make the NAK state permeable to one NAK after 1 second, thus limiting the amount of time a receiver's NAKs can be blocked.

Repeated retransmissions: While PGM in its current specification guarantees that a retransmission will not reach a receiver that has not requested it, it does not guarantee that a single retransmission will cover all receivers that have requested a retransmission. In some topologies PGM may have to send the same retransmission multiple times to cover all receivers. The reason is that a receiver close to the loss may send a NAK and trigger a retransmission before NAKs from distant receivers have a chance to establish NAK state in downstream routers. Since NAK state is wiped out by RDATA, a NAK arriving at a router after RDATA went through will re-establish

NAK state back to the source. This problem is depicted in Figure 3.5, which shows the same

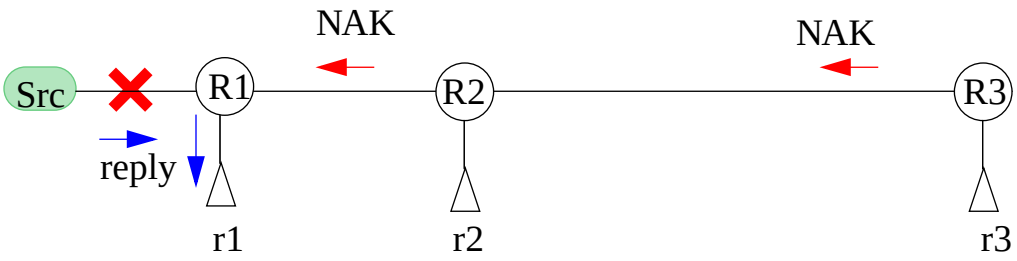

Figure 3.5: The repeated retransmissions problem in PGM

packet being retransmitted three times in response to a single loss.The RTT between receivers is such that the retransmission initiated by r1 passes through R1 to r1 before the NAK from r2 arrives at R1. Thus the NAK from r2 goes to the source and triggers another retransmission. If r3 is sufficiently distant, its NAK arrives at R2 after the second retransmission to r2, triggering a third retransmission of the same packet.

Retransmitting the same packet multiple times may create congestion near the source, which may lead to more loss and, in turn, more retransmissions, eventually leading to congestion collapse. A solution to this problem was sketched by the PGM designers and involves adding a back-off delay at the source before RDATA is sent. This allows NAKs to establish the appropriate state at the routers. This reduces multiple retransmissions at the expense of increasing recovery latency.