Chapter 4

LMS: Light-weight Multicast Services

In this chapter we present the major contribution of this thesis, namely *Light-weight Multicast Services*. LMS is a set of services provided by routers to greatly simplify the solutions to basic multicast transport problems. Even though LMS was motivated by the problem of scalable reliable multicast, we stress that LMS itself is not tied to a specific problem. It is rather a set of services that *facilitate* efficient solutions to several problems. In this chapter we concentrate on how LMS can be applied to reliable multicast, which is just one application of LMS; we list other possible applications of LMS at the end of the chapter.

LMS changes the existing IP multicast model by enhancing it with new forwarding functionality. The decision to change the IP model did not (and should not) come lightly. The existing model has arguably [82, 83] served the Internet very well to date, and understandably any attempt to change it will be met with skepticism. The model, however, which has proven successful in a unicast Internet, has come under some criticism in multicast applications. Much effort has been expended to date attempting to devise solutions to basic multicast transport problems like error and congestion control within the confines of the existing model (see some examples in Chapter 3: Related Work). These solutions, however, have met with generally limited success, and appear to have lost a significant amount of the elegance and efficiency present in their unicast counterparts.

Why have solutions to basic multicast transport problems proven so elusive? We touched upon these issues in the previous chapter, where we described problems like implosion and exposure.

These problems make multicast error recovery far more difficult than unicast, and consequently much harder to solve within the existing model. Thus, it is perhaps time to explore enhancements of the traditional IP multicast model.

In this chapter, we begin by defining an imaginary but highly efficient scheme, which we believe to be close to optimal in terms of efficiency and scalability. We use this scheme as a guiding light in our effort to realize a practical solution. After failing to apply traditional techniques to satisfactorily implement our solution, we present Light-weight Multicast Services (LMS). We describe how with LMS we can finally realize our elusive solution. We then proceed to discuss in detail the issues raised by LMS, including enhancements, limitations, and applications to other problems. Finally, we conclude the chapter by summarizing the features and strengths of LMS.

4.1. Defining an Efficient Scheme

In the previous chapter we described two important scalability problems faced by reliable multicast, namely implosion and exposure. Recall that the magnitude of these problems varies, depending on the location of a packet drop, which is hard to predict, thus making these problems difficult to solve.

Before trying to address these problems individually, we first try to determine if we can devise an integrated solution, i.e., one that eliminates both implosion and exposure. For this exercise we set aside any concerns about implementation - we will return to these later. We adopt a top-down approach for a couple of reasons: first, defining such a solution helps us understand the problems better; second, by defining a "good" solution, we arm ourselves with a standard against which other solutions can be compared with. In this section, we define a solution, analyze its operation and justify why we believe it to be near-optimal and therefore a good standard.

4.1.1. A Near-Optimal Solution

Consider the example in Figure 4.1, which shows a multicast group experiencing loss. Assume that a packet is dropped on link L marked with "X". Also assume that the source is relatively far away from the point of loss and there is a closer receiver that has received the data correctly. In our example, two receivers become important: the one marked as *requestor* and the one marked as *replier*. What makes these receivers important is that the requestor is the closest of the receivers

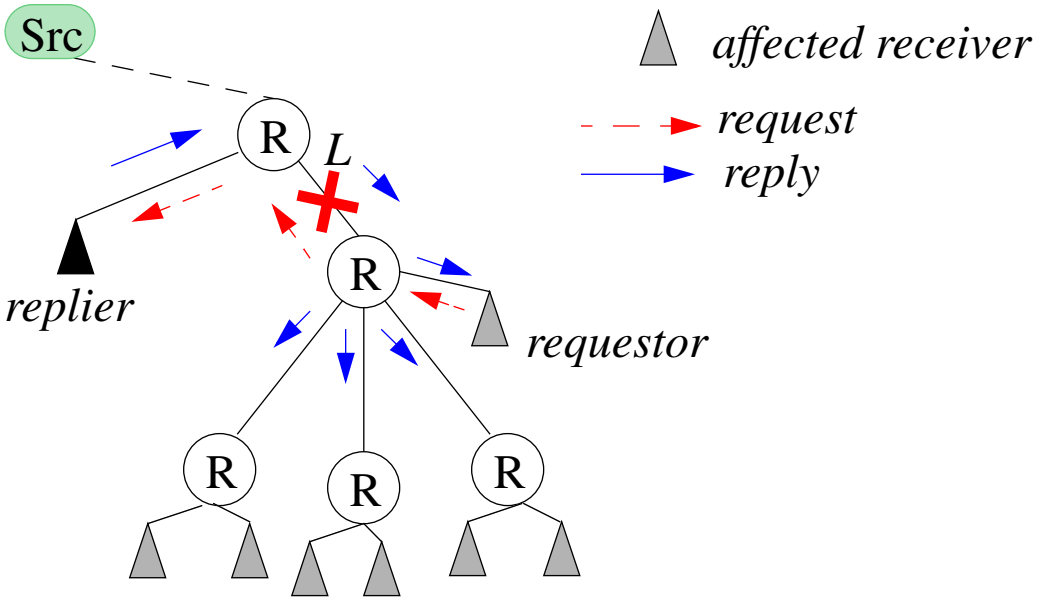

Figure 4.1: An imaginary, efficient recovery scenario

downstream of the loss that has missed the packet, and the replier is the closest receiver *upstream* of the loss that has the packet. As you might have guessed, we would like these receivers to initiate recovery, because their proximity to the loss will minimize *recovery latency*, another important performance metric which we will discuss in a moment.

So how can these two receivers collaborate to facilitate recovery? Assume that by some imaginary mechanism, the requestor knows that it is the closest receiver to the loss and sends a NACK upstream, while all other receivers refrain from sending NACKs; since only one request is generated, we have no NACK implosion. Then, using another imaginary mechanism, the NACK is forwarded to the replier only, without bothering any other receivers upstream. Finally, the replier using yet another imaginary mechanism initiates the multicast of a retransmission on link L; as a result, the retransmission rides on the existing multicast tree and is thus delivered to all receivers that missed the original packet.

We claim that the scheme we just presented is very efficient. Let us justify this claim:

- Only one NACK is generated by the receiver closest to the loss. A single NACK per lost packet is both necessary and sufficient to initiate recovery (assuming no NACK loss).

- Only one retransmission is generated by the receiver immediately above the loss. Again, assuming no further loss of retransmissions, a single retransmission is both necessary and sufficient to repair the loss.

- Initiating the retransmission at the point of loss ensures that only affected receivers receive the retransmission, which eliminates exposure. In addition, utilizing the underlying multicast tree results in good efficiency (in terms of latency and bandwidth) in delivering the retransmission.

- The NACK and the retransmission are initiated by the receivers closest to the point of loss. In addition, both messages are sent immediately upon error detection and reception of a request. These actions together ensure that requests and retransmissions travel the minimum distance and are sent as soon as possible, minimizing latency.

Let us study the general characteristics of our imaginary scheme a little further. We note that the scheme uses *local recovery*. By local recovery we mean that a loss may not necessarily be recovered from the original sender, as done in unicast, but may be recovered from a nearby receiver who has the lost packet. Local recovery is crucial to scalability because it does not require the participation of the source to recover each loss[1] (which may occur frequently in very large groups). Thus, by using receiver-reliable recovery *and* allowing receivers to send retransmissions, we significantly improve scalability by taking a substantial burden away from the sender. The use of local recovery implicitly assumes a *collaborative environment*, i.e., that receivers trust each other not only to retransmit lost data, but also to retransmit the *correct* data. This touches upon another thorny issue, that of *multicast security*, which is still under research. We discuss some security issues towards the end of this chapter. In addition to trust, local recovery assumes that receivers are willing and able to allocate some resources for recovery, namely processing cycles and buffer space.

In terms of performance, we noted that our scheme not only eliminates implosion and exposure, but also minimizes recovery latency. We discussed in the previous chapter the importance of limiting implosion and exposure in achieving scalability, but it may not be clear why minimizing latency is equally important. An obvious reason is that reduced latency may increase application utility: for

1. By loss here we mean the entire effect of dropping a single packet and the resulting loss observed by all receivers that missed it.

example, in multimedia applications data is only useful if delivered within certain time window, otherwise it is considered lost. However, another important reason is that latency affects the amount of buffering needed to serve retransmissions. Recall from Chapter 3, that with NACK-based error control an inexact decision must be made as to when to purge retransmission buffers. Thus, if recovery latency increases with the group size, buffers must also increase to limit the probability that a request may arrive after the buffers have been released.

Having sketched what we believe to be a near-optimal solution, we now turn to issues of implementation. Our multicast model is IP Multicast, which provides a simple, and therefore quite flexible model. IP Multicast uses a multicast-to-everyone model, i.e., a packet addressed to the group is received by all members, and provides receiver anonymity, i.e., the exact constituency of the group is not known to anyone.

When trying to implement our solution within this model it immediately becomes apparent that this will be a difficult task. Specifically, we encounter the following problems:

- Due to anonymity, it is impossible to identify the requestor/replier pair, and thus we are not able to forward requests to an appropriate replier.
- There is no mechanism to identify either the router or the link at the root of the loss subtree.
- There is no mechanism to deliver a message to a router for retransmission.
- There is no mechanism to allow routers to perform fine-grain multicast to restrict a retransmission within a particular subtree.

The topic of how to overcome these difficulties is the subject of the following sections. To summarize, in this section we have sketched the operation of a near-optimal but imaginary recovery mechanism, assuming a collaborative environment, which we claim is very scalable because it eliminates implosion and exposure, and minimizes latency. We also saw that implementing such a scheme within the current IP multicast model poses many challenges. In the next section we describe an existing approach that attempts to addresses the problems we have identified. We argue, however, that this approach fails to provide a general solution.

4.2. Implementing Reliable Multicast Using a Hierarchy

In the previous section we presented a scheme that although efficient, is hard to implement due to the existing multicast model's anonymity and lack of fine grain multicast. In this section we present two variations of an approach that attempts to solve these problems by organizing receivers into a hierarchy.

4.2.1. A Logical Receiver Hierarchy

With a hierarchical approach, receivers are arranged in a logical structure, where each receiver is assigned a *parent*. The receivers assigned to a parent are its *children*, and the parent/children assignment forms a *hierarchy.* Parents and children know of each other by maintaining state. The hierarchy is typically created when the multicast group is formed and may remain *static* for the lifetime of the group, or may be *dynamic*, i.e., allow receivers to re-structure themselves as the group membership changes. Static hierarchies are simple to maintain, while dynamic hierarchies pose several challenging problems; we will examine these shortly. With a hierarchy, multicast error recovery is typically done as follows:

- when a receiver detects a loss, the receiver unicasts a request to its parent;
- if the parent has the requested data, it unicasts a reply back to the child; if not, it must also have sent a request to its parent, and thus it simply remembers that the child needs a retransmission;
- after the retransmission arrives, each parent forwards it to the children that requested it.

Note the differences of the hierarchical scheme compared to the efficient recovery scheme presented earlier. In the hierarchal scheme, every affected receiver generates a retransmission request; however, there is no implosion because the hierarchy limits the scope of requests between children and parents. Even though only one request actually triggers the retransmission, the remaining requests do not go to waste: they help eliminate exposure by establishing a return path for retransmissions. However, there are inefficiencies in this model compared to the earlier one. These are as follows:

- Retransmissions are not sent utilizing the existing multicast tree, but propagated hop-by-hop, which requires more work at the receivers and increases recovery latency.

- To achieve efficiency, we must be careful to maintain a hierarchy whose logical structure is *congruent* with the underlying multicast tree. By congruent we mean that the logical and physical structure should coincide. If congruency is not achieved a receiver may be assigned a parent that is downstream in the multicast tree. Such a parent will not be helpful in recovery because it will have lost the same packet as its child (assuming correlated loss). Maintaining congruency is hard to achieve because it requires knowledge of topology.

- If the group membership changes dynamically, the hierarchy needs be updated frequently: New receivers have to find suitable parents, and existing receivers whose parents leave the group must be re-assigned to new parents. Finding parents is hard due to the multicast model's anonymity.

- A hierarchy needs to be established for each sender. Thus, in multi-sender groups using source-based or bidirectional trees, the hierarchy overhead is proportional to the number of senders.

In summary, while a receiver-organized logical hierarchy appears to solve the problems of implosion and exposure quite elegantly, such an approach is more appropriate to relatively static groups. It is not well-suited for large, dynamic groups because it is hard to maintain a congruent logical and physical structure. For such groups we need a solution that can efficiently maintain congruency.

4.2.2. A Router Hierarchy

In the previous section we examined a fairly good approximation of our efficient solution using a logical receiver hierarchy. We argued that while such a solution is reasonably efficient, is not well suited for large, dynamic groups because, among other problems, it is hard to maintain congruency between the logical and physical structure. In this section, we address the problem of congruency by using help from the network, i.e., from routers.

The problem of maintaining congruency can be solved if we enlist help from the network. Assuming that routers are allowed to buffer packets, detect losses, send requests and retransmit lost packets, we could make each router a receiver in the multicast group. Then, by using only routers to form our hierarchy we can solve the congruency problem, as depicted in Figure 4.2. Here is how

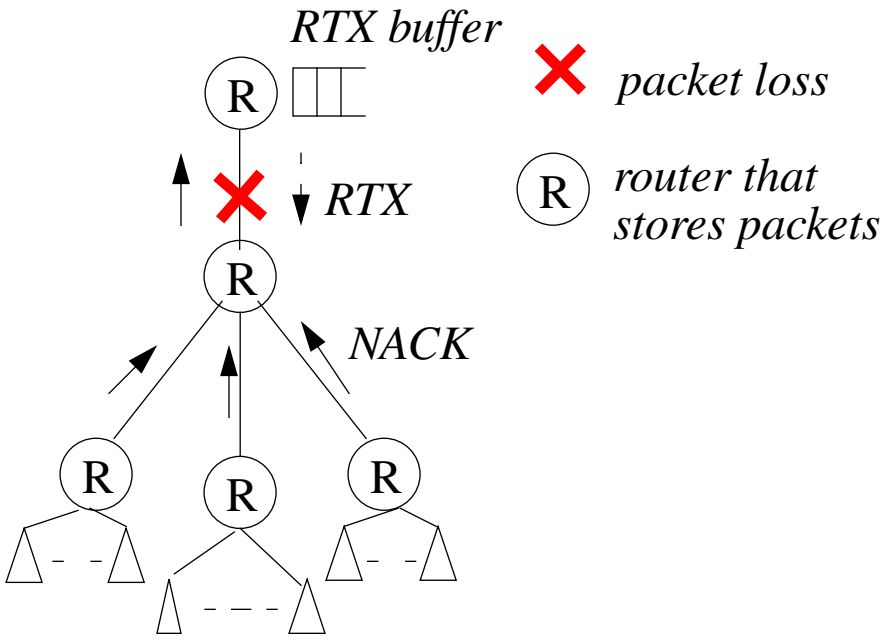

Figure 4.2: Reliable multicast using a router hierarchy

it would work:

- as soon as a packet is dropped on a link, the link's downstream router detects the loss and sends a request to the upstream router asking for the packet to be re-multicast on that link;
- routers downstream the loss send only one NACK upstream and suppress similar NACKs arriving on their downstream links; this eliminates implosion;
- the first router upstream of the loss re-multicasts the packet on the link where it received the NACK (which is the link on which the packet was dropped); this ensures that the retransmission follows the same path the original transmission would have followed, had it not been dropped, thus eliminating exposure.

This solution is quite simple and elegant; it not only retains the advantages of the previous solution (i.e., suffers from neither implosion nor exposure), but it actually improves upon them: it reduces recovery latency, because requests travel a shorter distance, and it eliminates the hop-by-hop delivery of retransmissions, replacing it with a single, efficient multicast. However, despite its simplicity and elegance, the scheme has several problems that make it impractical. These problems include:

- It is extremely heavy-weight. Such a solution requires that routers buffer all multicast packets that require reliability. This may potentially consume an enormous amount of resources at the router - resources that might be better spent on reducing loss, not recovering from it.

- Processing at routers would increase significantly. Routers are already struggling to keep up with increasing link speeds and high demand. Such a scheme would slow them down considerably because they would have to manage buffers and state, and make transport-level protocol decisions (e.g., when to purge buffers and how to handle late requests). Such protocol processing would most likely be done in software, bypassing the fast path.

- Such a solution not only violates layering, but goes against one of the fundamental Internet design principles that states that intelligence is best placed at the endpoints rather than inside the network.

In summary, our attempt to create an acceptable implementation of our efficient solution using traditional ways to create a hierarchy has failed. We considered a receiver hierarchy first. While this proved to be quite efficient, it has difficulties maintaining congruency between the logical and physical structure, and is thus unsuitable for applications involving dynamic groups. Then, we considered a router hierarchy, which not only solved the congruency problem, but actually improved the performance of our efficient recovery scheme. Unfortunately, we observed that in practice a router hierarchy is very hard to implement, because it adds too much weight and complexity to the routers and violates proven Internet design principles.

4.3. The LMS Concept

In this section we come to the main contribution of this thesis. Armed with the lessons learned in the previous sections, we take a fresh look at the problem, in an attempt to reconcile efficiency and practicality. We begin by taking a closer look at the router hierarchy solution, which despite its problems, is appealing due to its elegance, efficiency and simplicity.

We have already seen that the scheme's main strength is that because it lives at the routers, it is able to take advantage of its location to automatically build a congruent hierarchy; in other words, the scheme can instantly select points to (a) perform request suppression and (b) initiate fine-grain multicast of retransmissions. However, hidden in the above statement is the following key observation: it is not the router's processing power that the scheme exploits, but the router's *location*. In other words, the key advantage of enlisting routers to assist in error recovery, is not to utilize their processing cycles, but their knowledge of topology. This is important because our major objections with the router hierarchy scheme emanated from the fact that we had routers do processing; but as we just observed, processing is not what is important in this scheme. Would it then be possible to move the processing away from the routers, but not lose the location advantage? Or to summarize the question:

> *In the heavy-weight router hierarchy model, efficiency is not achieved due to harnessing the routers' processing power, but due to their location. Would it then be possible to move the processing away from the routers while retaining the location advantage and thus preserving the elegance and efficiency of the model?*

The above question lies at the heart of LMS; it also defines what LMS is: a set of services whose purpose is to allow the migration of processing away from the routers while retaining the location information. LMS achieves this by moving the processing from each[1] router to another entity, called a *surrogate*. The surrogate thus becomes responsible for performing recovery tasks on behalf of the router. The conceptual transformation from the router hierarchy to the surrogate model is depicted in Figure 4.3.

---

1. Here we are referring to routers that would have participated in recovery in the router hierarchy model.

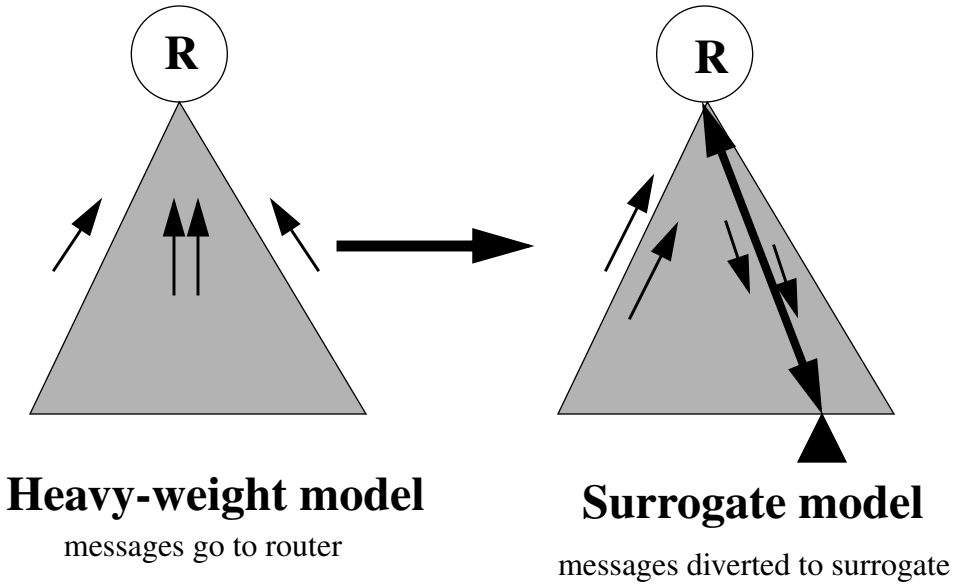

Figure 4.3: The LMS Concept

The next question is "who should the surrogate be?" While it is certainly possible to off-load router processing to a recovery server (e.g., a separate box attached to the router), this only sidesteps the problem. With such a server we introduce a new resource which needs to be managed and shared, just like the router before. Such a resource also constitutes a central point of failure. But more importantly, we have not removed the weight from the network - we simply introduced another network entity to carry it. Thus, we propose that the best candidates for surrogates are the *receivers*. While expecting a receiver to do some work towards recovery is not unreasonable (such a collaborative environment is at the heart of the Internet philosophy [21]), selecting receivers as surrogates has significant advantages: (a) it pushes the load of recovery at the ends rather than concentrating it at the routers, (b) has great flexibility because any future changes can be done at the ends without affecting the network, and (c) all recovery operations remain within the transport layer, which avoids the layer violation of the earlier model.

4.4. LMS Core Ideas

Having decided that we will use receivers as surrogates, we need to address the following questions:

- how does a router select a surrogate?
- how does a router redirect messages to its surrogate?
- how does the surrogate send messages on behalf of the router?

Before addressing these questions, let first us explain our terminology. Since the main problem here is multicast error recovery, we will shift from the term "surrogate" to the terms "requestor" and "replier", as used earlier in the context of our efficient recovery solution. In most cases, we will be using the term "replier" to denote the router surrogate; however, both the requestor and the replier are surrogates. What distinguishes them in the context of error recovery is whether they are sending a request or a retransmission. In summary:

Router surrogate = Replier = selected receiver

Requestor = any receiver (plain or a surrogate) that sends a request

4.4.1. Selecting a Replier (surrogate)

Since we chose receivers as repliers, a router needs to select a separate replier for each multicast group. The process of selecting a replier is very simple:

- if the router has two or more downstream links, select one as the replier link;
- if the router has only one downstream link, select the downstream link;
- if the source is directly attached to the router, select the source.

Figure 4.4, shows a possible router-replier allocation. The links that lead to a replier are in bold. Note that a router only needs to know the replier *link*, not the actual receiver. For example, router R2 selects the right-most link as the replier link; it does not need to know that E5 is the receiver acting as a replier, which was selected by router R4. Therefore, a router only needs to identify the next hop of a path leading to a replier, which has some important advantages:

- If the replier changes, the change remains mostly local. For example, if R4 decides to switch to E4 as its replier (because E5 either left the group or crashed) R2 does not need to change its replier state.

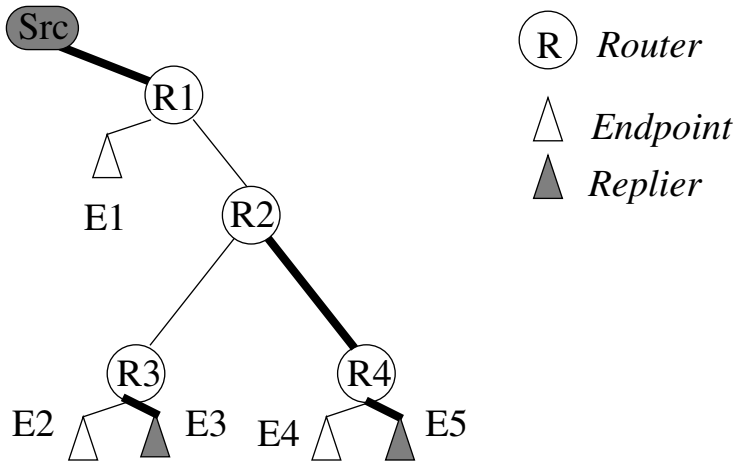

Figure 4.4: A possible replier allocation in LMS

- Receivers do not have to be notified that they have been selected as repliers. A receiver knows it has been selected if it receives a request. There is, however, no guarantee that the receiver will remain a replier for the next request.

- The replier state at the router is small. It is simply an identifier for the replier link.

In our discussion thus far, we did not specify which link the router selects as a replier link when there are two or more candidates. While the simplest solution would be to choose one at random, there are many reasons why we would prefer the receivers to influence the replier selection. For example, some receivers may be better suited to act as surrogates because they are located on fast machines, or machines with larger memory. Therefore, we introduce a replier *metric* to the receiver join procedure by which receivers control the replier selection at the routers, as described next.

Replier Selection Mechanism

To establish replier state, joining receivers express their desire to become repliers with the join request. We expect that all receivers will agree to become repliers (data recovery can be achieved even if this is not the case, but at reduced efficiency). Along with the join request, receivers communicate a metric of their appropriateness to be repliers. Routers pick repliers according to this metric. Receivers repeat their willingness to be repliers and their metric on every membership refresh.

Thus, through the combination of the join/refresh messages and the metric receivers can control the replier allocation at routers.

Replier Selection Metric

The metric used by repliers to communicate their appropriateness to the routers is a scalar value, whose meaning is determined by the receivers. The routers do not need to interpret the metric, but simply compare it with the metric provided by other receivers. Thus, receivers wishing to minimize latency for example, may choose a metric that represents the round-trip-time from repliers to the routers, so that routers select the nearest replier. Other receivers may use processing and memory capacity as a metric, so that hosts with more resources are preferred over those with less resources.

In summary, receivers are free to agree upon which metric is important, translate it into a scalar value and communicate it to the routers. It is a simple matter for the routers then to chose repliers according to the receivers' wishes.

4.4.2. Steering Messages to Repliers

After the replier selection is in place, the routers need mechanisms to steer messages from requestors to repliers. We describe these mechanisms next.

A request sent by a requestor needs to climb up the multicast tree until it finds a router which has a replier. It it thus important that the request follows the reverse multicast path. To achieve this the request is multicast, but contains a special IP option that forces routers to remove it from the fast path and examine it. That option is the IP Router Alert option [44]. The request contains the information shown in Figure 4.5.

The first two fields are part of the standard IP multicast header. The values in the IP option are new and are used as follows: The address of the original source is needed so that the router will select the correct replier (see discussion on source spoofing in section 4.9.) The request identifier simply identifies this message as a request; the request sequence number counts the number of similar requests sent; it is used to suppress duplicate retransmissions as discussed in section 4.5. The request bitmap specifies which packets are requested. The last field, the turning point information, is initially empty. We explain its purpose next.

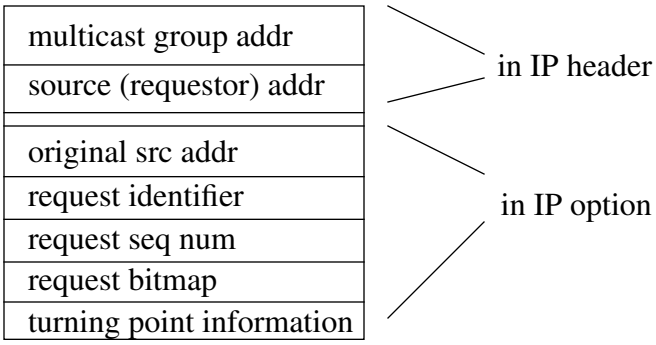

Figure 4.5: An LMS request

Request Handling at the Routers

As we saw before, even though a request is multicast to the group, it is immediately picked up by the first router because of the IP Router Alert option. A request may arrive at the router from three possible locations, as depicted in Figure 4.6:

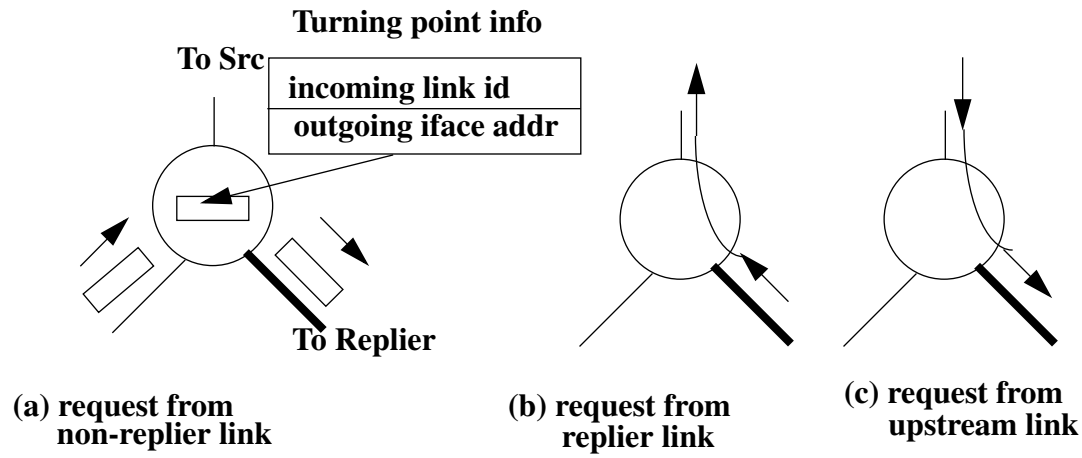

Figure 4.6: Request handling at a router

- **Request arrives on a non-replier link:** note that this case implies that the router has at least two downstream links. We call this point the *Turning Point* of the request. The reason is that the router will turn this request around (recall that the request was traveling upstream until this point) and send it out the replier link. The turning point is the only case

where some router processing is involved: before forwarding the request to the replier link, *and if the turning point field is empty*[1], the router fills in the turning point information. This information consists of an identifier for the link the request arrived on (the non-replier link) and the address of the interface the request if forwarded out on. Note that the turning point information globally identifies the point where the request was received. The reasons for carrying this identification in the request will become apparent when we describe how replies are sent.

- **Request arrives from the replier link:** when a request arrives from the replier link, then the router forwards the request to the upstream link, without making any changes to the packet.

- **Request arrives from the upstream link:** similar to the previous case, when a request arrives from the upstream link, the router forwards it to replier link unchanged.

Let us now examine the net result of request handling at the routers and see how it avoids implosion. Note that each router allows only one request to be sent upstream - the replier request. All other requests are funneled into the replier link, and follow replier links to eventually reach the replier. Therefore, if all routers behave in a similar fashion the maximum number of requests a router sends to the replier is equal to the number of downstream links at that router minus one. Assuming that routers do not have a large fan out, repliers will not experience implosion. Routers with a large fan out can be dealt with as described in section 4.13.

To summarize, we have added a simple steering mechanism at the routers which solved the implosion problem by allowing only one request to escape each router and propagate upstream; all other requests are forwarded to repliers.

The Request Turning Point

In the previous section we briefly touched upon the issue of the turning point information carried in a request. This is an important concept in LMS. The turning point is essentially the point where a request, during its search for a replier, arrives at a router through a non-replier link. Thus,

1. See later discussion on proxy directed multicast in section 4.14.1. as an example of where a router may find the turning point field non-empty.

a request has limited scope: it moves upstream until it finds the next closest path to a replier. A request arriving on a particular link signifies that all the receivers downstream that link require the data. Thus the location of the turning point specifies the root of the loss subtree that sent the request.

Note that because of the request handling, only one replier (who has the data) will receive a request; in addition, this replier will receive only one request for each loss event. This is a nice property because it eliminates ambiguity and the need to coordinate multiple potential repliers.

4.4.3. Directed Multicast (DMCAST)

If a replier receives a request but does not have the requested data, the replier ignores the request since it must have sent a similar request of its own. Otherwise, the replier retransmits the data using a *Directed Multicast* (DMCAST). This is the final service LMS adds to the routers; its purpose is to enable fine-grain multicast to eliminate exposure.

The operation of a DMCAST is shown in Figure 4.7. To perform a DMCAST, the replier creates

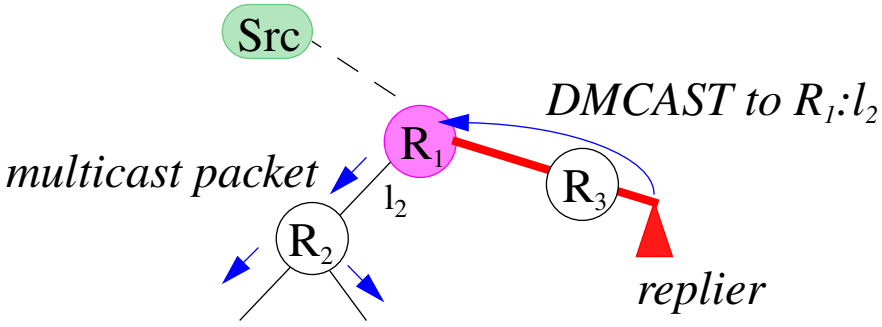

Figure 4.7: A Directed Multicast (DMCAST)

a multicast packet, addresses it to the group, and inserts the requested data. The packet contains an IP option with the turning point link identifier obtained from the request. The replier then encapsulates the multicast packet in a unicast packet and sends it to the turning point router, whose address was again obtained from the request. When the turning point router receives the unicast packet, it decapsulates the multicast packet, strips the IP option and multicasts it on the specified link. From that point on, the packet travels as a regular multicast packet originating from the source.

4.4.4. LMS Summary

In the previous sections we presented the operation of LMS from the router's point of view. Here we summarize the concepts and operation of LMS.

LMS employs three important concepts:

- Each router selects a receiver which acts as a surrogate. The surrogate performs processing on behalf of the router. We call this surrogate a replier.
- Routers are enhanced with steering mechanisms that allow the delivery of special messages to the surrogate. The router simply forwards these messages, and maintains no per-packet state.
- Finally, the routers are enhanced with the capability to receive encapsulated multicast messages and multicast them on specific links, thus delivering them to specific subtrees.

These concepts work together to enable receivers to construct an efficient recovery mechanism. These steps are depicted in Figure 4.8 and summarized below.

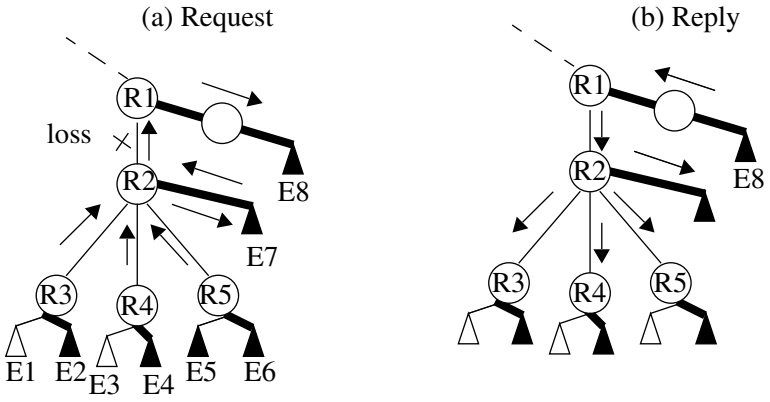

Figure 4.8: LMS Summary

Sending a request:

Replier links are in bold. Loss occurs on the link between R1 and R2. Endpoints E1 through E7 detect the loss. Then, the following events take place:

- E7 sends a request, which R2 forwards to R1 because E7 lies on R2's replier link.
- E1 sends a request which is forwarded by R3 to E2. Similarly, requests from E3 and E5 are forwarded to E4 and E6 by R4 and R5, respectively.
- The request from E2 is forwarded to R2, because E2 is on R3's replier link. Similarly, the requests from E4 and E6 are also forwarded to R2.
- R2 forwards requests from E2, E4 and E6, to E7, which ignores the requests since it does not have the data (but has requested it).
- The request from E7 reaches R1, which forwards it towards E8, which has the requested data.

Sending a Reply

Once E8 receives the request and determines that it has the requested data, it prepares a reply and sends it as follows:

- E8 creates a multicast message containing the reply. E8 encapsulates the message in a unicast message and sends it to R1 (the request's turning point).
- R1 decapsulates the multicast message and multicasts it on the link leading to R2.
- From that point on, all downstream routers and endpoints treat the reply as a regular multicast message coming from the source.

Note that LMS routers maintain no state other than the replier link identifier and cost. Routers need not remember anything about requests that passed through. Routers are not even aware that these messages are sent for data recovery; they simply forward messages according to the new forwarding rules, just as they forward regular multicast packets according to standard multicast rules. Thus, just like normal multicast, no per-packet state is needed at the routers.

4.5. Preventing Duplicate Retransmissions

Multicast error recovery protocols (including those using LMS) that allow receivers to send retransmissions, face an ambiguity problem when requests arrive at receivers asking for data just received through retransmission. We will call these requests *late requests*. The ambiguity problem is depicted in Figure 4.9: assume that the links between R3 and R2 and R4 and R2 are links with

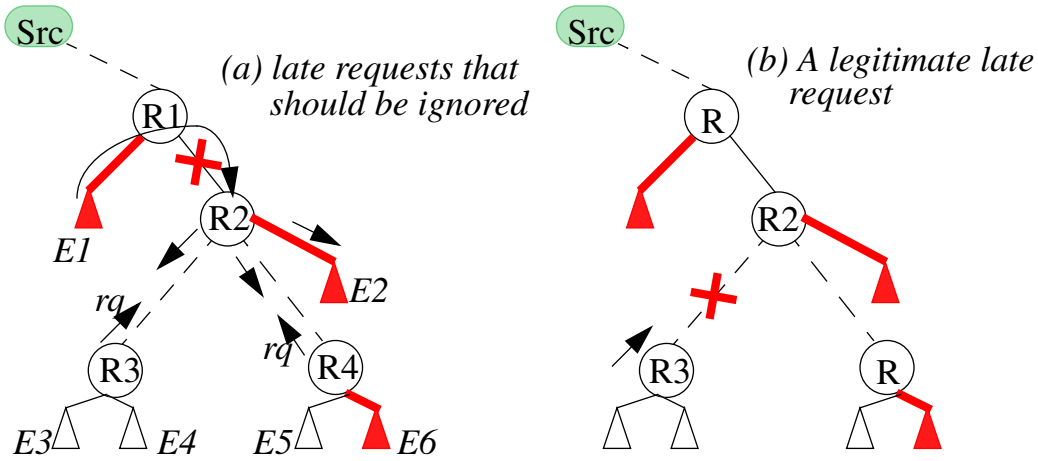

Figure 4.9: Late requests lead to ambiguity

long propagation delay, as shown on the left of the figure. Now suppose a packet is lost between R1 and R2. A request from E2 will reach E1 which will DMCAST the data to the subtree. Now suppose that the requests from R3 and R4 reach E2 after the reply; obviously E2 should ignore these requests. However, if we look at the right side of the figure, late requests may be legitimate if the retransmission was lost again. For example, if the previous retransmission was lost between R2 and R3, the request from R3 is obviously legitimate.

Clearly, the arrival of late requests leads to ambiguity: E2 does not know if the requests have crossed the retransmission, or if they are legitimate requests for a lost retransmission. To overcome this problem, we propose that repliers do not serve late requests, unless they receive a second request. An optimization is for receivers to mark their first request, which can always be safely ignored by repliers.

The above is a conservative approach, but one that ensures that eliminates ambiguity. If some duplicates are acceptable, repliers can serve requests immediately (except those marked as first). Another possible solution is to introduce an "ignore" period at repliers, where repliers ignore requests for some time after receiving a retransmission. We expect that different applications will employ their own method of dealing with late requests.

4.6. LMS Specification: Forwarding Services

In the previous sections we described how error control can be performed using the services provided by LMS. Our discussion intentionally blurred the distinction between the LMS forwarding services and the error control operations at the endpoints. In this section and the next, we clearly separate the two and give a clear description of the actions taken at routers and endpoints. In this section we describe the LMS forwarding services; in the next section we give the error control operations at the endpoints that use the LMS services.

4.6.1. Replier State

The replier state at the routers is created and maintained with *replier refresh* messages. A refresh message carries with it the following information: (a) the <*S, G*> entry this message is for; and (b) an associated *cost*. A refresh message may arrive from any of a router's downstream interfaces for a particular <S, G> tree. For the following discussion, *replier-iface* and *replier-cost* refer to the router's existing replier interface and cost; *new-iface* and *new-cost*, refer to the interface the incoming refresh message was received and the new cost carried by the message. The values replier-iface and replier-cost are initialized to values unattainable in reality (e.g., infinity). Periodically, the router checks all replier entries and expires those that have not been updated within a predefined interval. The following pseudocode describes the router actions after a refresh message is received:

```
if refresh arrived from upstream interface, then
     silently discard packet
else if the router has only 2 interfaces in <S, G>, then
     send refresh upstream
else if replier-iface = new-iface, then
     replier-cost = new-cost
     send refresh upstream
```

```
else if replier-cost > new-cost
          replier-cost = new-cost
          replier-iface = new-iface
          send refresh upstream
else silently discard packet /* replier-cost <= new-cost */
```

4.6.2. Handling Requests at the Router

As described earlier, requests are multicast but arrive at routers carrying a special IP option. The option causes the requests to be picked up by the router and examined more closely. Requests carry an <S, G> entry with them, where S is the address of the source that sent the data being requested. The following are the actions a router takes upon receiving a request. As before, the router's replier interface is stored in the variable *replier-iface*, and the interface the request arrived on is in *new-iface*.

```
if replier-iface not set, then
     send request upstream
else if replier-iface = new-iface, then
     send request upstream
else
     if turning point info is empty, then
          turning point address = replier-iface address
          turning point iface = new-iface
     forward request to replier-iface
```

4.6.3. Handling Directed Multicasts at the Router

A directed multicast arrives at a router in the form of a multicast packet encapsulated in a unicast packet. The multicast packet carries a special IP option with the identifier for the interface the packet should be multicast. Assume that the variable *dmcast-iface* contains the identifier of that interface. Upon reception of a dmcast the following actions take place:

```
decapsulate packet
dmcast-iface = iface carried in IP option
if dmcast-iface is valid, then
```

```
     multicast decapsulated packet on dmcast-iface
else silently drop packet
```

4.7. LMS Specification: Error Control

In the previous section we described the forwarding actions at a router for LMS messages. Note that no error control specific operations were done during these forwarding actions. In these section we specify the host specific actions that perform the actual error control.

4.7.1. Sending Requests

As we mentioned earlier, receivers detect losses using gap detection. When a receiver detects a gap G it performs the following actions:

```
create request(G) /* (see Figure 4.5) */
clear Turning Point information
set request seq num = 0
send request and start retransmission timer
while (reply not received)
     increment request seq num
     double wait interval
     send request
```

4.7.2. Receiving Requests

Assume that a replier receives a request *RQ* with sequence number for gap *G* and the turning point of the request is *TP*; The replier then performs the following actions:

```
if we do not have the requested data, then
     discard request
else if we received data as original transmission, then
     send dmcast (TP, G)
else /* a late request */
     deal with late requests /* application specific */
```

4.7.3. Receiving Retransmissions

As mentioned earlier, receiving a reply in LMS is very much like receiving normal data:

```
if gap exists for retransmission, then
     insert retransmission in the buffer
else discard the packet
```

A fine point in our protocol is that it is possible (as we have seen during our simulations) that a retransmission may arrive before the receiver even detected its loss. This can happen if another receiver experienced a shorter loss and initiated retransmission early. When this occurs, the receiver simply treats the retransmission as original data and initiates requests for any gaps detected by the reception of the retransmission.

4.8. Problem: Duplicate Data Packets

Since there is always one replier in our scheme, no duplicate replies are ever generated. It is possible, however, that a receiver may be exposed to replies generated as a result of recovery initiated by receivers in other regions. An example is shown in Figure 4.10. In this example, a packet is

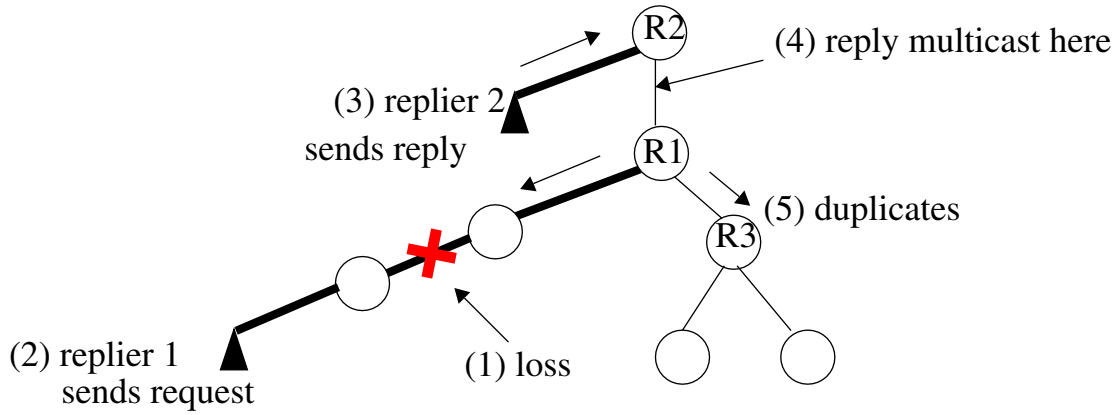

Figure 4.10: Exposure in LMS

lost on the path between R1 and replier 1. Replier 1 sends a request which reaches Replier 2. In response to the request, Replier 2 sends a directed multicast to R2, which multicasts the reply on

the downstream link leading to R1. The reply reaches all of R1's downstream links, causing duplicates on the subtree routed at R3.

Even though this problem does not inhibit recovery, it may lead to the "crying baby problem," where excessive loss experienced in one branch causes duplicates at a large number of other receivers. In LMS this problem is solved by using the cost field to select a replier that advertises the least loss. For example, R1 will select a replier from the right-hand-side branch if this branch experiences less loss, even though the replier on the left-hand-side branch may be closer.

4.9. Source Spoofing

Some routing protocols (e.g., DVMRP) create a separate multicast tree for each sender. With such protocols, the multicast reply resulting from a directed multicast must contain the original sender's address as the source address, otherwise it will not reach the appropriate receivers. To avoid this problem, we allow repliers to use the original source's address in the multicast packet (i.e., perform "source spoofing"). If desirable, to allow receivers to distinguish spoofed from real packets, routers ensure that spoofed packets are marked and include the replier's address in the message. Thus, source spoofing poses no additional security concerns since the real sender can always be identified by the recipient. Source spoofing is unnecessary with routing protocols that create shared trees.

4.10. Dealing with Shared Trees

While most of the multicast capable routers on the MBONE today use DVMRP, some new routing protocols like PIM-SM and CBT create shared multicast trees around a core or rendezvous point, and thus do not maintain per-source information. This enables them to scale to groups with many sources. In order to handle these protocols, we propose the following changes to our scheme, as depicted in Figure 4.11.

We calculate subtree repliers as before for the shared tree. A request is directed by the routers to the repliers as before. Requests from repliers are directed towards the core. In order to guarantee that the source will eventually receive the request, whenever the core receives a request it unicasts it to the source. The source in turn performs a directed multicast to the turning point as before. So

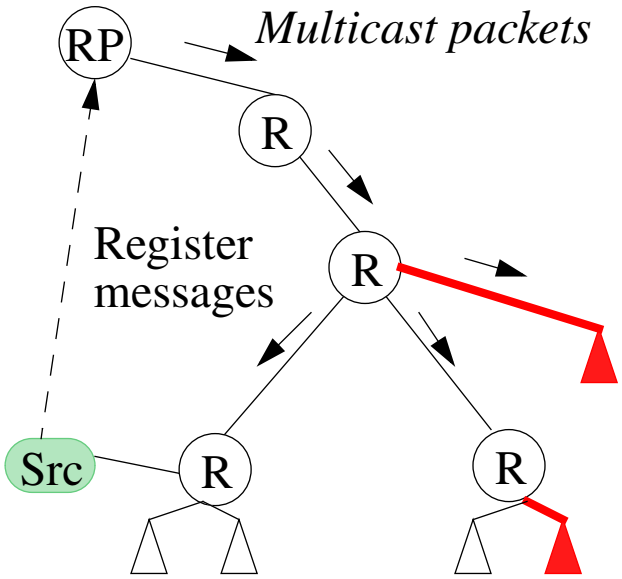

Figure 4.11: Dealing with unidirectional shared trees

rather than have the source directly connected to the root as in DVMRP schemes, the source is connected by a unicast path to the root.

The above modification works well for unidirectional shared trees like PIM-SM. Sources in PIM-SM initially send data to the core via register messages, which are then multicast by the core. Thus the core always lies upstream. If the core sends a join message to the source, then intermediate routers along the join path maintain per-source information, which allows correct routing of requests. However, in CBT, routers can no longer distinguish upstream and downstream links with respect to a source. Thus, LMS in its current state does not work with bidirectional shared trees, unless some per source information is added to the router state.

4.11. Replier Failure

The failure of a replier may disrupt recovery until the failed replier is detected. Note, however, that a failed replier will not always disrupt recovery; replier failure becomes a problem only if either the requestor below the loss or the replier above the loss fail. Thus if the break in the replier continuity does not coincide with the location of loss, recovery will proceed unaffected. Failed repliers will eventually be detected by routers when soft state expires. However, detection via soft state may

take too long; to enable fast detection of replier failure, we propose the following mechanism, depicted in Figure 4.12:

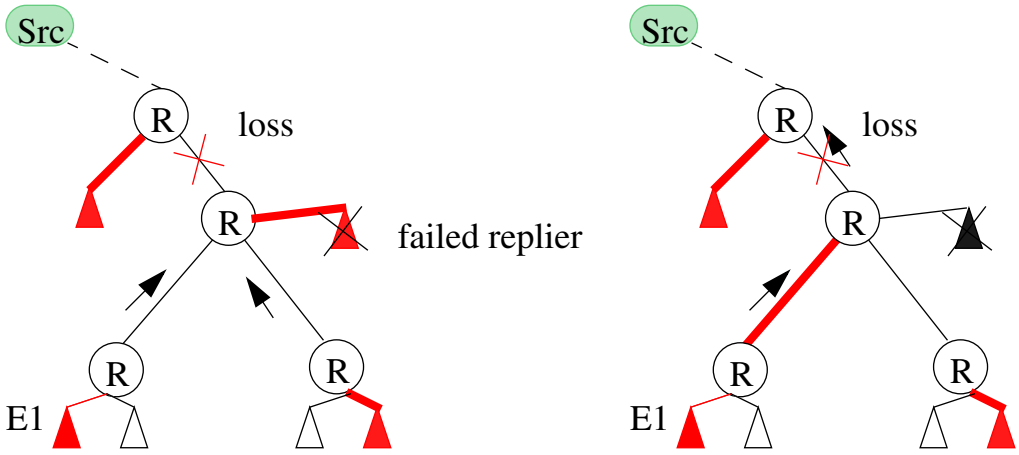

Figure 4.12: Detection of replier failure

- If after sending a request, the receiver's retransmission timer expires, the receiver sends another request; this process is allowed to repeat up to *N* times.

- If after *N* attempts there still no response, the receiver declares the replier unavailable. The receiver then, sends another message to the router at the turning point, alerting the router of possible replier failure.

While detection of a failed replier is taking place, the replier closest to the loss uses *heartbeat DMCASTs* to assure downstream receivers that the failure is being dealt with. A heartbeat DMCAST is special in the sense that the router forwards it on *all* downstream links, including the replier link. The next replier downstream the failed replier takes the responsibility to perform the necessary actions to restore the replier continuity. In the figure, this replier is E1. E1 knows that it is the closest to the loss and should initiate the replier recovery process because it receives no heartbeat messages, except its own. Receivers closer to the loss start their heartbeat earlier. All remaining receivers who receive a heartbeat, defer the repair to E1. These receivers can in turn monitor E1 through its heartbeat. E1 seizes its heartbeat after a retransmission is received, signifying that the replier continuity has been restored.

4.12. Selecting Repliers in a LAN

For simplicity, the previous sections have assumed that only one receiver resides at each router link. This, of course, is not always true. In many cases routers are connected to a LAN where multiple receivers may belong to the same multicast group. A possible LAN/replier allocation is shown in Figure 4.13. In such cases, receivers on the LAN run a simple election algorithm to elect a replier.

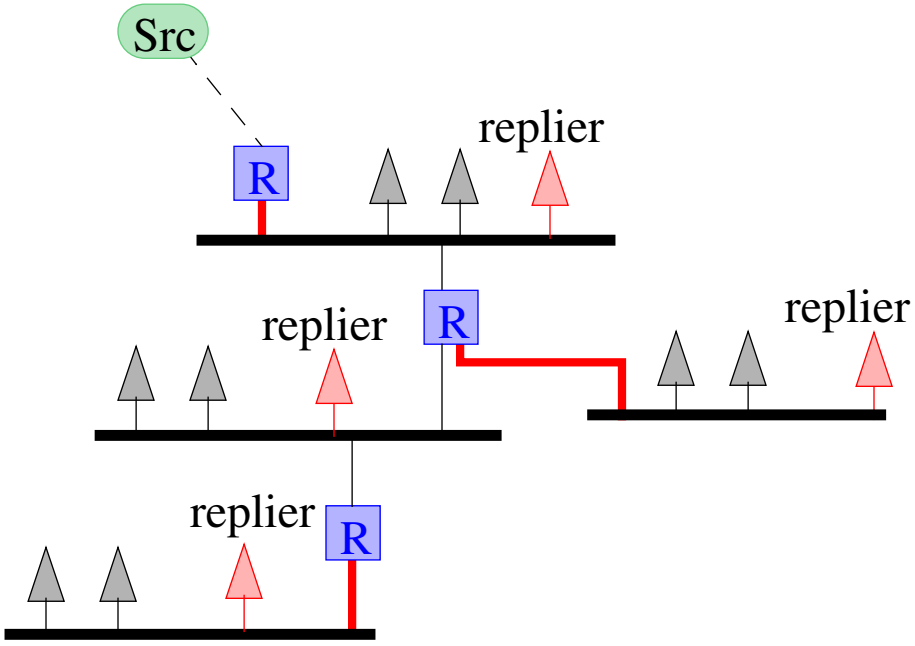

Figure 4.13: Repliers in a LAN

The election takes place without any involvement from routers. Receivers use local multicast (i.e., a multicast with TTL set to 1) for the election. Typically, the first receiver on the LAN becomes the replier; new receivers check for a replier by querying the LAN via a local multicast. If a replier exists, the replier responds with another local multicast. When the replier leaves the group, it sends a local multicast announcing its departure, which triggers an election by the remaining receivers to elect a new replier.

The method by a which a new member discovers existing repliers is depicted in Figure 4.14. Potential repliers upon joining, perform the following actions:

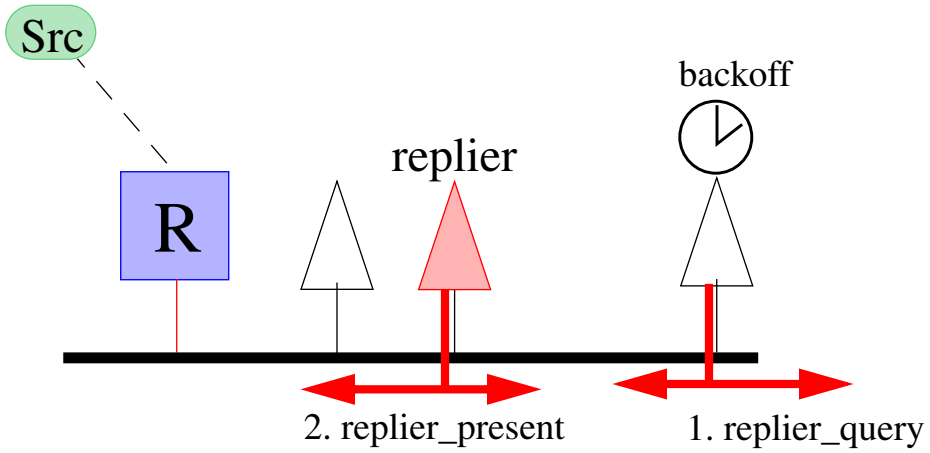

Figure 4.14: Electing Repliers on a LAN

- When a new member that is willing to be a replier joins a multicast group, the member multicasts a *replier_query* message on the LAN (TTL=1) and starts a *replier_query_timer*.
- If another replier exists, it multicasts a *replier_present* message on the LAN. The new member cancels its timer and sets a *replier_present* flag.
- If the replier_query timer expires, the new member declares itself as the replier by multicasting a *replier_present* msg on the LAN.
- Periodically the replier refreshes its status by multicasting a *replier_present* message to the LAN.

When a replier wants to leave the group, it takes the following actions:

- The replier multicasts a *replier_leaving* message on the LAN.
- The other members on the LAN start a new replier election. If multiple candidates are present, the election may be decided based on the advertised cost. Ties can be broken based on IP address.

- If the replier crashes without sending a *replier_leaving* message, the periodic *replier_present* messages will seize. Remaining members detect it and start a replier election.

The router does not participate in this election - it involves only receivers. The router does not need to know which receiver is currently acting as the replier, only that *some* receiver is acting as a replier. It is up to the receivers to ensure that a replier is present in the LAN.

Request Suppression on a LAN

When receivers on a LAN detect loss, they use back-off timers to delay sending requests. The replier, however, multicasts its request immediately, thus cancelling other receivers' requests. This is depicted in Figure 4.15. If loss was internal to the LAN, the replier repairs the loss with a local

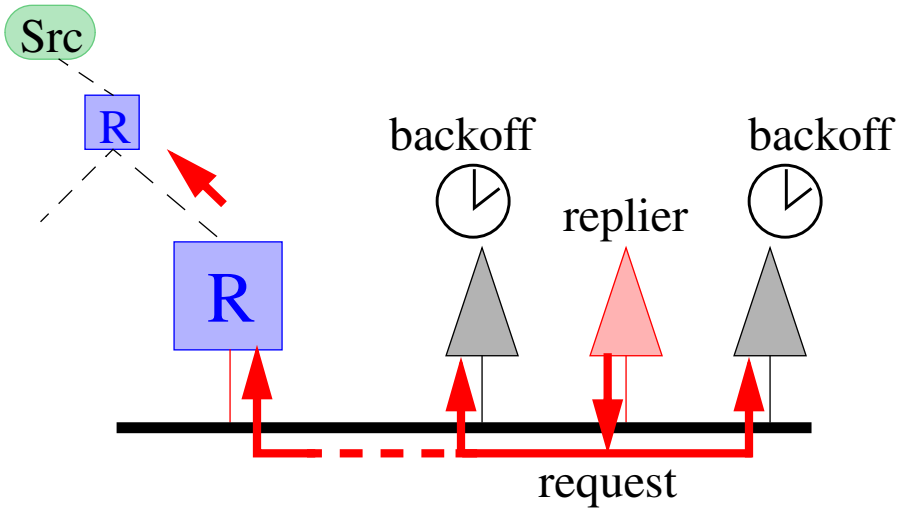

Figure 4.15: Suppressing requests on a LAN

multicast. However, if loss was specific to the replier, the replier's request will cause a duplicate to arrive on the LAN.

When a request arrives on a LAN, the router multicasts the request and all receivers receive it. However, only the replier responds to the request. Other receivers ignore it.

4.12.1. Coordinating Routers Interconnecting LANS

Routers interconnecting LANs need to maintain some extra state in order to forwards LMS messages correctly. An example is shown in the topology depicted in Figure 4.16, between LAN1

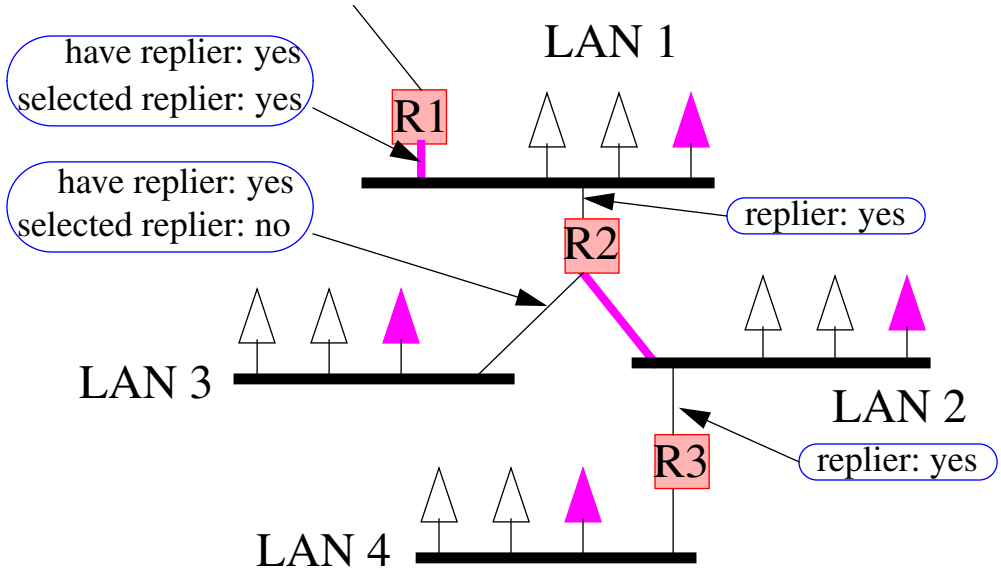

Figure 4.16: Routers connecting LANs

and R2. Suppose a request arrives at LAN via R1, destined for LAN1's replier. R1 multicasts the request on LAN1, where it is received by the replier as we described earlier. However, if R2 is not aware that LAN1 has a replier, then R2 may propagate the request to LAN2, to its own replier. In turn, R3 may do the same, and so on. This would result in multiple replies. To avoid this problem, R2 and R3 need to remember that the upstream LANs have a replier and thus if a request appears on these LANs these routers should not propagate them.

4.13. Routers with a Large Fan-out

If a router has a large fan-out for a specific group, the router's replier may receive a large number of requests from downstream repliers. To avoid this problem, the router may partition its links into smaller groups and select a replier for every group, as shown in the example in Figure 4.17. In this example, requests from links in group *D* go to replier *d*, but requests from replier *d* go to replier *c*, requests from replier *c* go to replier *b* and so on. Requests from replier *a* are forwarded upstream.

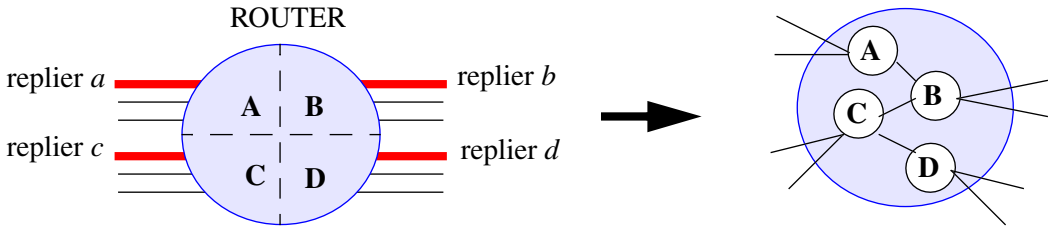

Figure 4.17: dealing with routers with large number of links

By partitioning links this way, the maximum number of requests a replier can receive is significantly reduced.

4.14. Improvements

There are some possible options receivers can utilize to improve the performance of LMS services. These improvements do not change the functionality of the services, only the manner in which receivers use these services. We describe these improvements next.

4.14.1. Proxy Directed Multicast

In the previous examples we assumed that the first replier who receives the request has the data and services the retransmission. This however, may not always be true. For example, since we employ a NACK-based protocol, the request may arrive after the buffers have been purged and even though it identifies the correct turning point, it cannot be served by this replier; or perhaps for security reasons, the retransmissions are only allowed to come from the original source. Even in such cases, the DMCAST service in LMS can still be used almost as effectively as before. We call this proxy directed multicast.

Once a request passes the turning point it contains enough information to uniquely identify the subtree that requires the retransmission. Thus, if a replier receives a request after its buffers have been purged, the replier can either forward the request upstream, or send it to the sender. In either case the request will eventually arrive at a replier which has the data (assuming there is one); this replier can then initiate a DMCAST to the original turning point specified in the request, thus reaching the correct subtree.

In order to preserve the original turning point, routers forwarding a request to a replier first check if the turning point field is empty. If it is not employ, the router does not overwrite the existing information, but leaves it untouched.

4.14.2. A DMCAST that Eliminates Exposure

As we have seen earlier, loss at a replier link may cause duplicate messages to reach some receivers. In this section, we describe a method to eliminate these duplicate messages, at the cost of adding an extra hop to the retransmission. This approach is shown in Figure 4.18. To eliminate

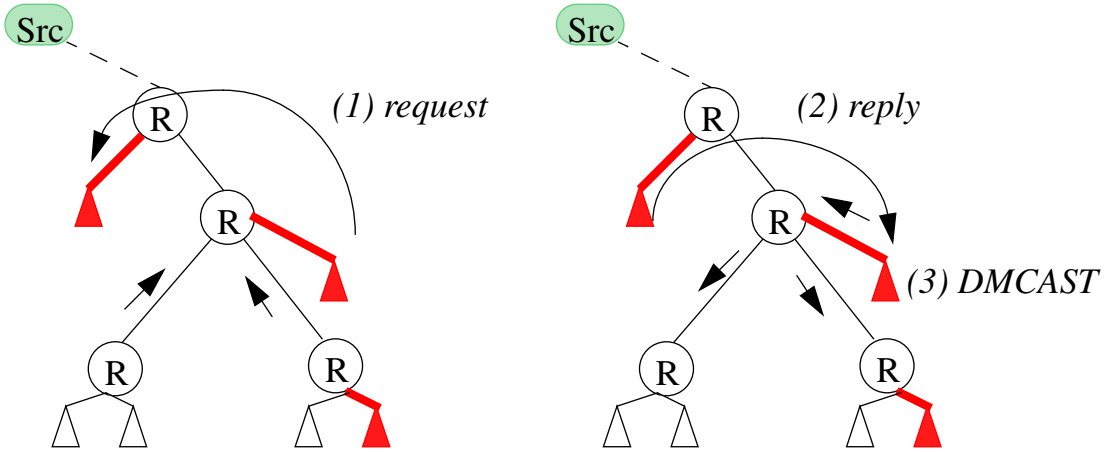

Figure 4.18: Eliminating Exposure

exposure, each request specifies that the reply should be unicast to the requestor rather than sent to the turning point with a DMCAST. If the requestor receives any other requests, either while waiting for the reply or soon after it receives it, then the requestor knows that this is a loss that affected more receivers than just itself. Therefore, it initiates a DMCAST to the remaining receivers. While it would be possible for the requestor to cover the loss by responding to each request with a separate DMCAST, another, perhaps simpler solution is to initiate a single DMACAST to all downstream links at the turning point.

4.15. Some Pathological Topologies

Some topologies may create pathological situations that may reduce the efficiency of LMS. In this section, we describe two such cases and propose possible solutions to overcome the problem. It is not clear how likely these topologies are to occur in real-life. The topologies used for our simulation experiments did not exhibit any such pathologies.

4.15.1. Long and Skinny Branches

These topologies may cause increased recovery latency in LMS. An example is shown in Figure

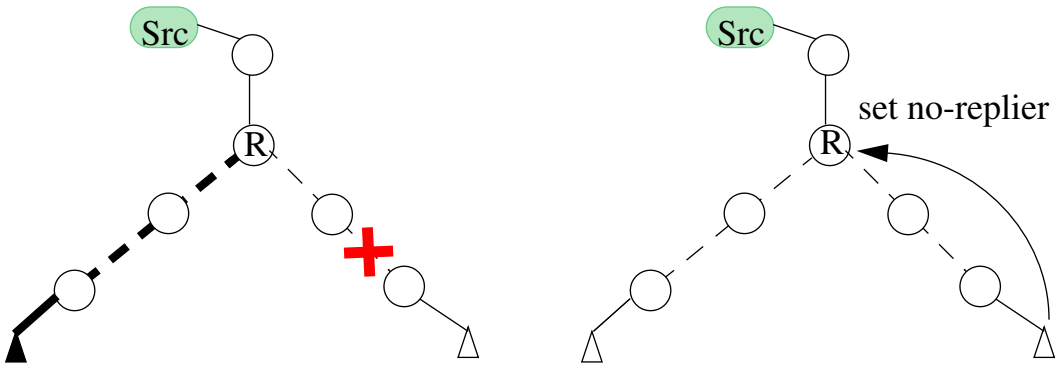

Figure 4.19: Long, skinny branches may increase recovery latency

4.19. These are essentially topologies composed entirely of long, skinny branches with no repliers in the middle. Latency may be increased in such topologies because router R has no choice but to select one of the long branches as the replier link. Thus, requests from a non-replier branch travel all the way back (possibly near the source) only to be diverted down another long branch to the replier. The same applies to retransmissions: they must follow the long path through the turning point.

A possible solution to this problem is the following: receivers that experience long recovery latency (compared to their RTT to the source) due to such topologies, advise the turning point router to release its replier and propagate requests from all downstream branches upstream, in the hope that another replier may be located closer to the turning point. The router may or may not follow

this advice: if the number of downstream links is small thus not risking implosion at the upstream replier, then this is a viable solution. Otherwise, receivers will have to deal with the increased latency. Note that if the router follows this advice, it should continue to insert the turning point information to requests so that dmcasts will be efficient. A disadvantage of this approach is that if loss happens upstream of the turning point, then one DMCAST per downstream link will be required at router R.

4.15.2. Topologies that cause Request Implosion

A pathological topology that may cause request implosion is shown in Figure 4.20. Recall that

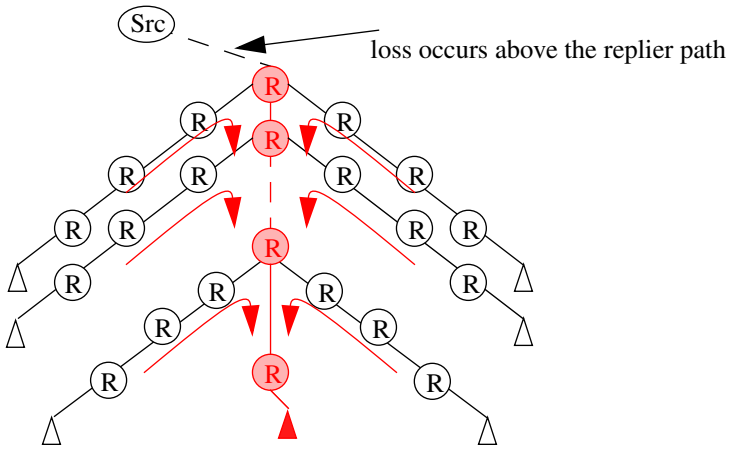

Figure 4.20: A Pathological topology that may cause Implosion

a router forwards at most one request on its upstream link. Thus, the maximum number of requests a replier can receive is typically bounded by the number of downstream links at the turning point. However, it is conceivable that in such pathological cases where many routers have selected the same replier, the replier may receive a potentially large number of requests. In the figure, a large number of neighboring routers (shaded) have selected the same replier (also shaded), forming a long replier path. Every request reaching a router on the replier path is now forwarded to the same replier, making the number of requests at the replier proportional to the sum of the downstream links of all the routers on the replier path.

The problem can be solved by modifying the replier cost to take into consideration the sum of the children of all routers along the replier path. At each hop, a router modifies the cost to reflect the number of its children (minus the replier link). Thus as it moves upstream, the cost becomes high and the next upstream router is forced to select a different replier link, thus shortening the replier path.

4.16. Other LMS Applications

One of the important advantages of LMS is that its mechanisms are general enough to be used for other purposes besides error recovery. For example, the services used for implosion control can be used by many other applications where receivers need to implement a *scalable collect* or *voting* service. Such a service enables a large number of receivers to efficiently convey some value to the sender without risking implosion. The fine-grain multicast capability provided by the DMCAST service can be used for targeting specific parts of the multicast tree, for example to announce the presence of a server in that region. We expand on these applications next.

4.16.1. Simple ANYCAST

ANYCAST [45] is a service used when a client wishes to discover one server (typically the nearest one) out of a group of servers providing a service the client is interested in. Examples are replicated file systems, or the DNS service. LMS can be used to implement a simple ANYCAST service by grouping servers in a well-known multicast group. Servers use LMS to notify routers that they want to be repliers, choosing perhaps distance as the replier metric. Clients searching for servers send requests to the multicast group, which are directed to the nearest replier (as with retransmission requests) thus establishing contact with the server. These steps are shown in Figure 4.21.

One small change required in the replier selection method to implement ANYCAST. Servers tell the routers to advertise the existence of a replier (server) in all links rather than just the upstream link as before. which ensures that routers find the nearest server in any direction.

4.16.2. Positive Acknowledgment-Based Reliable Multicast

Recall from the previous chapter, that NACK-based protocols cannot guarantee 100% reliability due to the lack of positive ACKs. To achieve complete reliability, one should either employ a

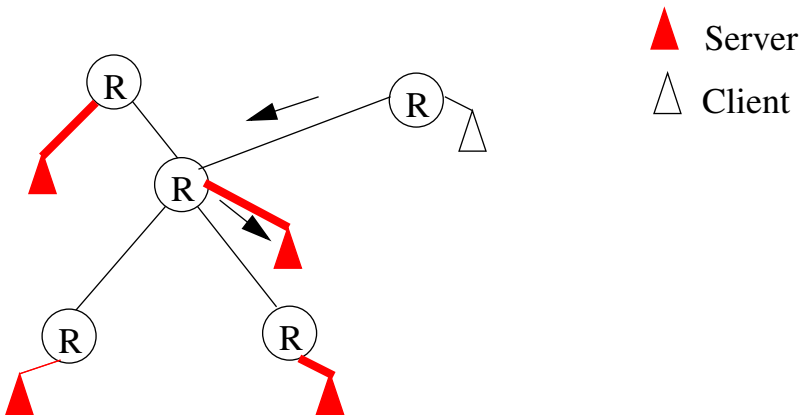

Figure 4.21: LMS implements a simple ANYCAST service

positive ACK-based protocol, or supplement NACKs with periodic synchronization via ACKs. The mechanisms for both approaches are similar; the main difference is the frequency of synchronization, which happens on every packet (or window) in an ACK-based protocol, instead of at a larger interval with a NACK/ACK combination.

LMS can be used to implement either a pure ACK-based protocol, or NACK/ACK protocol. In either case, repliers take the responsibility to collect ACKs from downstream receivers, fuse them, and send a cumulative ACK upstream. Thus the sender finally receives a single ACK containing the highest consecutive sequence number from all receivers.

LMS can help simplify the implementation of such protocols. For example, additional mechanisms that would be needed to assign children to parents (as done in existing protocols using static hierarchy [35]), have been eliminated; tree congruency, a sticking point with other protocols, is now automatic; a joining receiver searching for a parent can easily find one upstream by sending a request; switching to a new parent after the current leaves the group is again easy: it simply requires that downstream receivers send a probe message searching for a new parent; finally, the DMCAST service is still available to send retransmissions efficiently.

4.17. Security Issues

Current multicast security schemes employ encryption techniques to ensure security in multicast groups [46]. With encryption techniques, the receivers are given keys, which they use to decrypt packets sent by the sender. There are two possible problems that may arise with secure applications in LMS. These are the following:

- Since in LMS retransmissions may come from receivers, how can the requestors ensure that the retransmission sent by the replier is the same as the data sent by the source?
- How can requestors ensure that repliers will respond to their requests?

To address the first problem, in secure applications we require that repliers buffer copies of packets in their encrypted state, i.e., as they are received from the sender. Thus, when a request arrives, the entire packet is DMCAST in its original form. In addition, we require that DMCASTs contain the address of the replier in their body (or at least some indication that this is a retransmission) to inform receivers. This prevents a replier from faking original data from the source. This can be done by having routers check that retransmissions not coming from the source contain an IP option that clearly identifies it as a retransmission.

The second problem is addressed with the same mechanisms used to detect replier failure which we described earlier.

4.18. Incremental Deployment

Incremental deployment is an important issue in the deployment of any new scheme affecting routers in the Internet. The Internet has become so big, that a clear incremental deployment plan is essential.

The easiest way to deploy LMS is with the next generation of IP, namely IPv6[62, 42]. IPv6 is currently running on an experimental overlay known as the 6-Bone [63]. LMS can also be deployed following a similar approach as the one proposed for the incremental deployment of PGM [47]. PGM uses *Source Path Messages* (SPMs) to create an overlay of PGM aware routers. PGM signalling then occurs between these routers. The SPM approach works for LMS; it is depicted in Figure 4.22 and described below.

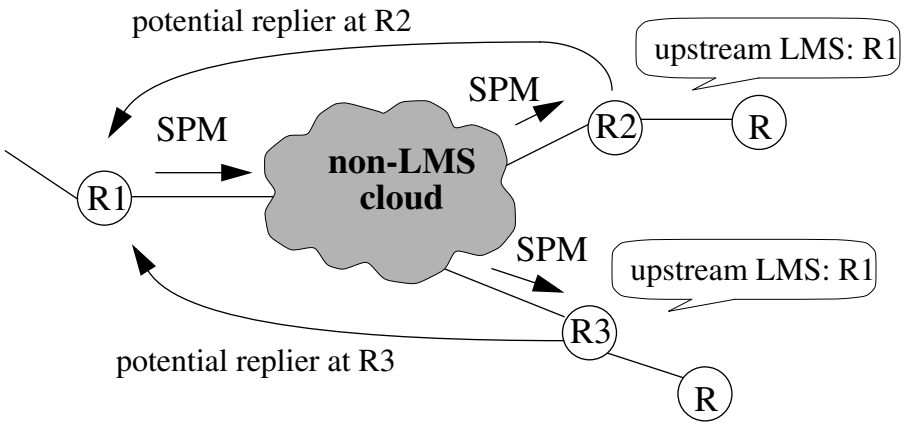

Figure 4.22: LMS Incremental Deployment

Periodically, the source multicasts SPMs to the group. These also carry the IP Router Alert option, so that all routers examine them. Non-LMS routers receiving these packets will simply ignore the option and forward the packets in the normal fashion. When an LMS router receives an SPM, it notes the address of the sender. If the sender is not its upstream neighbor on that link, then it notes the address of the upstream LMS element. It then writes its own address in the SPM and forwards it as usual. This process will eventually create an LMS overlay in the reverse direction. Messages destined for the upstream link in LMS will simply be sent to the upstream LMS element instead.

Note, however, that unlike PGM, LMS requires an additional step. LMS needs to send messages to repliers, so a router needs to know which of the possibly many downstream elements leads to the replier. To achieve this, replier updates are sent upstream through the LMS overlay as described above; when a router selects a replier that lies across a non-LMS cloud, instead of noting just the link it selects the address of the next downstream element. In our example in the figure, R1 receives solicitations from both R2 and R3 for replier selection. R1 will chose one of them, but will save the address of the chosen router as the next replier hop.

Incremental deployment has an effect on exposure. For example, a retransmission initiated at the turning point may reach receivers in a non-LMS cloud. If loss occurred upstream the non-LMS cloud, the retransmission will be useful to all receivers; however, if loss occurred downstream the cloud, receivers in the cloud will receive a duplicate.

4.19. Summary

In this chapter we presented the major contribution of this thesis, namely Light-weight Multicast Services (LMS) and how they can be used to implement a scalable reliable multicast protocol. We discussed how LMS addresses implosion and exposure, which are the main problems with reliable multicast, and showed how LMS solves these problems efficiently and with very little weight. We also identified some limitations of LMS and showed ways to overcome them. Then. we argued that services offered by LMS are general and can be used by other applications besides reliable multicast; such applications include ANYCAST and building a positive acknowledgment-based reliable multicast protocol. LMS offers mechanisms to scalably collect feedback from receivers in large multicast groups which are useful in several types of applications.

We conclude with a summary of the main strengths of LMS:

1. **LMS eliminates implosion:** we have shown that the maximum number of requests a replier will receive, is bounded by the number of downstream links at the turning point. While we expect this number to be generally small, we have shown how a router with a large number of downstream links can organize them in a logical hierarchy to ensure there is no implosion.

2. **LMS maintains low exposure:** while it is possible that some receivers may receive duplicate messages, LMS has the capability to reduce this occurrence by selecting appropriate repliers. However, as we will see in the next chapter, this problem is actually quite small to begin with.

3. **LMS maintains low recovery latency:** receivers in LMS will typically experience recovery latencies smaller than their unicast RTT to the source. Certain topologies, however, may cause latency higher than the RTT to the source. We have shown methods of dealing with such problems when they occur.

4. **LMS simplifies applications:** with LMS receivers do not need to maintain any state related to topology, except for their RTT to the source for purposes of recovering from lost retransmissions (similar to unicast). This is a great asset of LMS. It means that applications need not be concerned with discovering parents, maintaining RTTs to other receivers, or other similar complexity. The work required by applications is comparable to the work done in unicast.

5. **LMS is self-configuring:** this is yet another important asset of LMS: the replier hierarchy is built instantly as receivers join and leave the group, and with the exception of refreshing replier state (which can be done with the refreshing of group membership) is fully transparent to the receivers. Thus the replier hierarchy always mirrors the underlying topology.

6. **LMS requires very little state at routers:** the added state at the routers is minimal: it consists of an identifier for the replier link, and the cost value of the current replier. These can be as small as two bytes each. This is a minimal addition to the existing routing state.

In the next chapter, we use simulation to evaluate the performance of LMS and compare it with two other reliable multicast schemes.