

Improving Internet Multicast with Routing Labels

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

Abstract

The IP-multicast architecture is extended with addressing information along multicast routing trees that permits more efficient and sophisticated multicast routing options and encourages communication and cooperation between IP and higher-layer protocols. The Addressable Internet Multicast (AIM) architecture is introduced that enables sources to restrict the delivery of packets to a subset of the receivers in a multicast group on a per-packet basis, permits receivers to listen to subsets of sources on a subscription basis, provides nearest-host routing, and allows higher-layer protocols to place packets into application-defined logical streams, so that hosts may direct the multicast routing of packets based on application-defined contexts. In addition, the Reliable Multicast Architecture (RMA) is introduced to support end-to-end reliable multicasting using heterogeneous reliable multicast protocols and providing acknowledgment trees implicitly, thereby eliminating the ACK implosion problem and allowing NAK-avoidance algorithms to work within local groups.

1. Introduction

The design of *multicast* protocols for many-to-many communication over an internetwork was first explored by Deering [7] almost a decade ago. Deering's seminal work has become the foundation of the IP-multicast design and architecture [5]. The strength of the IP-multicast architecture is the anonymity of the sources and receivers involved in the session. Sources may send information to a known multicast address without explicitly knowing the constituency of the receiver set, and similarly, receivers are not required to announce their presence to any other member of the multicast session. This anonymity greatly reduces the complexity of the mechanisms needed to provide the multipoint distribution. On the other hand, the weakness of the IP-multicast architecture is the absence of addressing information within the context of multicast groups, which severely limits packet-delivery semantics: A source in a multicast group can send packets only to the entire group.

For routing purposes, IP-multicast uses multicast IP addresses of groups, which are really names because they are independent of the relative location of the group members in the multicast routing trees used for the groups. As a result, IP-multicast is unable to deliver packets to subsets of a

multicast group on a per-packet basis; a separate multicast group must be constructed for each subset of the multicast group, which is a lengthy and costly process. Furthermore, when protocols working over IP-multicast require some notion of relative position between two hosts, or when it is required that the scope of a packet to be restricted, several inadequate work-around solutions must be employed. Roundtrip delay is often used as a heuristic for determining the relative location of two hosts [8, 14], despite the fact that roundtrip delay in the Internet is very dynamic and is a poor measure of relative location. When attempting to restrict the scope of a multicast transmission on a per-packet basis, many protocols utilize the time-to-live field present in all IP packets to limit delivery to a locus of routers a limited hop count away [8, 15, 22, 14]. However, hop counts are a crude measure of locality of reference for most applications. Alternatively, some protocols may choose to create and tear down separate multicast groups, but this approach is efficient only when a series of packets justifies the overhead [16, 12].

In addition to the limitations in the IP-multicast architecture that are inherent to the absence of addressing information in multicast routing trees, there is no support for cooperation between the protocols providing the IP-multicast service and the applications or end-to-end transport protocols that use the service.

We propose a new multicast architecture that extends IP-multicast with group addressing information to permit sources to address subsets of the receiver set and receivers to listen to subsets of the sources within a given multicast group. We show how this can improve the efficiency, flexibility, and scalability of multicast routing in the Internet, and present a new architecture for end-to-end reliable multicasting in the Internet. Following the IP-multicast architecture, we assume that none of the multicast-group members needs to know the constituency of the group. Each Internet router can support more than one host, and routers are organized into multicast routing trees. Routers accept packets from their attached hosts and disseminate the packets over multicast routing trees maintained using any of the proposed multicast routing protocols [2, 6, 20, 18].

Section 2 introduces the Addressable Internet Multicast (AIM) architecture, which generalizes the IP-multicast architecture by introducing *group-relative addressing* information in multicast routing trees. This added information enables the provision of new sender- and receiver-initiated delivery services, and allows higher-layer protocols to place packets into application-defined logical streams, so that hosts may prune the routing of packets based on contexts meaningful to the applications. Section 3 describes how AIM can be implemented in existing multicast routing protocols. Section 4 presents two applications of AIM: scalable any-

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Grant F19628-96-C-0038 and by the UC MICRO Program.

casting and layered multicasting over a single multicast routing tree. Finally, Section 5 presents the Reliable Multicast Architecture (RMA), which builds implicit acknowledgment trees that eliminate the acknowledgment-implosion problem of large-scale reliable multicasting, allow NAK-avoidance schemes to operate solely within local groups, and support the efficient integration of heterogeneous end-to-end reliable multicast protocols.

2. AIM Definition

AIM (Addressable Internet Multicast architecture) generalizes existing Internet multicast services by introducing addressing information as part of the basic structure of multicast routing trees. AIM requires either a source-based or shared multicast routing tree (multicast tree for short), which can be constructed and maintained by CBT [2], OCBT [20], PIM [6], MIP [18], or other protocols. AIM defines an *addressing root* for each multicast tree, which is used as a reference point for addressing and routing along the tree.

AIM extends IP-multicast with three types of labels assigned to the routers of a multicast tree: *positional labels*, *distance labels*, and *stream labels*.

A router's positional label specifies its location relative to the addressing root of the multicast tree. At each router, each interface that belongs to a multicast tree is assigned a distance label specifying the number of hops along the tree needed to reach the nearest host or the nearest qualified type of destination in the same multicast group. The intent of these labels is to permit routers to multicast packets to specific subsets of the multicast group on a per-packet basis.

A stream label specifies a subscription by one or multiple receivers to the traffic generated by a subset of sources in the multicast group. These labels permit receivers to listen to subsets of sources on a subscription basis.

2.1. Positional Labels

The positional label assigned to a router specifies the router's position in a multicast tree relative to the tree's addressing root. With such labels, each source and receiver of the corresponding multicast group has an address within the multicast tree. This address can then be used by sources to address subsets of the receiver set on a per-packet basis, and by receivers to direct traffic to specific sources.

Any prefix labeling scheme for implicit routing along a tree can be used in AIM (e.g., [1, 14]); we adopt a simple scheme here as an example. The addressing root of the tree is labeled "1". All children are assigned a positional label that consists of their parent's label as a prefix and, as a suffix, an integer unique in the set of the parent's children. Whenever a child changes its parent, it receives a new prefix and suffix. The router's new prefix must be passed down to its children, who in turn must change their own prefixes. The same applies recursively down the tree. Figure 1 illustrates a routing tree that has been assigned positional labels. Figure 2 describes the positional labeling algorithm, where "&" denotes concatenation.

With the specific labeling scheme we just described, the size of a label increases with the height of the routing tree, and it is likely that at least three bits are needed for each integer that composes a label, so the labels may become large. However, it is only necessary to increase the size of a label

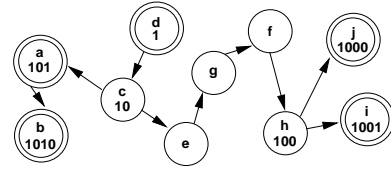


Figure 1. Positional labeling. Routers with attached hosts are double circled.

```

Assign-Pos-Label (routing-table)
  Local_var unique := 0
  If (this router does not have an attached host)
    AND (this router has 1 outgoing interface)
    Then interface.label := cur-label
  Foreach interface of session-routing-table
    If (interface does not lead to the tree-core)
      Then
        interface.label := cur-label & unique
        unique := unique + 1

```

Figure 2. Algorithm for assigning positional labels.

(i.e., add a suffix) at two types of routers: routers with directly attached hosts belonging to the multicast group, and routers that have at least three interfaces on the multicast routing tree. This has the added advantage that re-labeling does not occur after every topology change, but only when these two types of routers are involved.

2.2. Positional Routing

Routing along a tree that is positionally labeled is implicit, i.e., the labels of the source and the intended destination determines the chain that must be used along the tree. A router can determine the correct path for a destination node along the multicast routing tree by comparing the destination's positional label with its own label. This comparison determines whether the destination is an ancestor of the current router, or a descendant. If the destination is an ancestor of the current router on the routing tree, then the packet should be sent to the router's parent; otherwise, the destination is a descendant of the current router, and the packet should be sent to each child who heads the appropriate subtree. Figure 3 shows this simple routing algorithm. Throughout this paper, all algorithms assume that each router stores its current positional label in *cur-label*.

The proofs that algorithms equivalent to *Assign-Pos-Label()* and *Pos-Route()* perform loop-free routing on an existing tree of hosts are presented elsewhere [14]. The details

```

Pos-Route (dest-label)
  Local_var child-label
  If (sizeof(cur-label) > sizeof(dest-label))
    Then route the packet to parent.
  Else compare the first sizeof(cur-label)
    bits of cur-label and dest-label
    If (the numbers are not equal)
      Then route the packet to parent.
    Else child-label := the integer
      where the labels first differ.
      Route the packet to child-label.

```

Figure 3. Algorithm for positional routing.

of the proof for a multicast routing tree would be almost identical, and are omitted for brevity.

AIM relies on a simple language of masks to specify positional destinations of a packet. The packet header contains a list of one or more masks, and if a router's label satisfies the mask, the packet may be delivered to the attached hosts. Each mask is composed of a label and a two-bit flag. With both bits cleared, the router that matches the destination label can deliver the packet. If the "x" bit is set, then any router that has the label as a prefix can deliver the packet. A set "n" bit denotes a negation of the mask specified; in other words, only routers that do not satisfy the mask can deliver the packet. Some examples of this system follow.

- (100x) Deliver to hosts attached to routers with labels that have a prefix of 100.
- (n100x) Deliver to hosts attached to routers without a prefix of 100.

AIM utilizes a second time-to-live (TTL2) field. While the original time-to-time-live field from the existing IP-multicast model (TTL1) decrements starting from the source's router, TTL2, if specified, works in one of two ways: (a) TTL2 decrements from the router of the first destination, and when either TTL reaches 0 the packet cannot be forwarded; or (b) TTL2 decrements from the source's router, and the packet *cannot* be delivered until TTL2=0. For example, a delivery specification of (100x, TTL1=10, TTL2=3 type=a) would indicate the packet should be delivered to all hosts attached to routers with a prefix of 100, but would not reach any routers more than three hops away from the first router with a label prefix of 100, and only if the destinations are no more than 10 hops away from the source. Each mask listed in a packet's header requires its own TTL1, TTL2, and type fields. Additional definitions of TTL2 are possible. These fields could easily be defined as an extension header in either IP version 4 or IP version 6.

2.3. Distance labels

At each router of a multicast tree, distance labels are assigned to each of its interfaces in the tree (called tree interfaces). The distance label assigned to tree interface specifies the shortest distance to another router with attached hosts that are members of the group (called member hosts) and which also belong to a predefined type. The rest of this section considers the case in which a distance label simply denotes the hop count to the nearest router with an attached member host.

Distance labels are computed in a distributed fashion along a multicast tree. For initialization, a router with an interface with member hosts attached assigns that interface a distance label of 1; it also assigns a label of infinity to its other tree interfaces. Each router on the multicast tree that has at least one interface with a distance label of 1 sends a distance-label update with this value to its neighbor routers in the tree. A router that receives a distance-label update over an interface increments the received label by one and assigns that distance label to the interface. The router then advertises its smallest distance label, if it has changed, to all its neighbor routers in the tree. Figure 4a illustrates a multicast tree that has been distance labeled; member hosts are indicated by squares and routers are shown as circles. The numbers inside the circles indicate the distance labels of the router interfaces; note that the router with four tree interfaces advertises a distance label of two to its neighbors in the tree. Figure 5 contains the pseudo-code for the distance labeling algorithm.

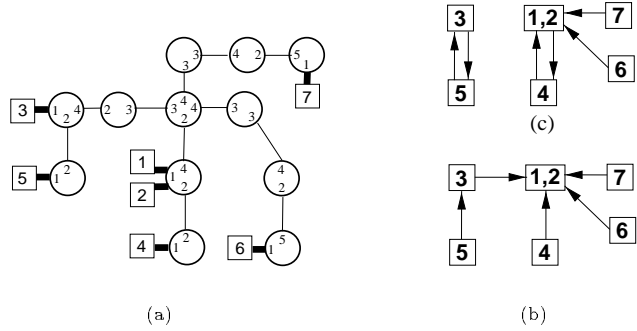


Figure 4. (a) Distance labeling of a routing tree. (b) Paths that can be taken with positional reachcasts toward hosts 1 and 2. (c) Paths that reachcast would take from each host.

Distance-Label (list of interfaces)

```

Set all interface.dist := ∞.
Foreach interface from list
  If (hosts are directly attached)
    Then interface.dist := 1
  Else set interface.dist equal to 1 plus
        incoming distance update from
        neighbor router.
Send minimum of distances reported
from other interfaces to neighbor router.

```

Figure 5. Distance labeling at each router.

The following theorem shows that *Distance-Label()* algorithm is correct for an arbitrary host and an arbitrary router.

Theorem 1: The *Distance-Label()* algorithm correctly labels each interface of a pre-existing multicast routing tree with the number of hops to the closest router with an attached host reachable through that interface.

Proof: Let x be the number of hops between an arbitrary host H and an arbitrary router R . The proof proceeds by induction on x . By definition of the algorithm, when H is attached directly to R , then $x = 1$. For this case, it is trivially true that the algorithm successfully labels that interface with the number of hops to the nearest host. For the case in which $x > 1$, we will assume the theorem is true for any router R within $1 \leq y < x$ hops from H . We must prove that the theorem is true for $x > y$.

Without loss of generality, we can assume that H is in fact the closest host to R on the interface that leads to H . Consider, router Q , which is the neighbor to R and is $x - 1$ hops away from H . Because $x > 1$, H is also the closest host to Q . Because $x - 1 < x$, the theorem is true for Q by the inductive hypothesis, and Q is able to correctly choose $x - 1$ as the minimum of all distances on all interfaces on the multicast routing tree. This value is forwarded to R , who must only increment the value, and assign the interface a distance label of x , which replaces any previous distance label greater than x . □

2.4. Reachcasting

Reachcasting consists of sending packets towards the nearest member host in any direction of a multicast routing tree.

```

Reachcast-Route ()
  If (any outgoing interfaces have dist==1)
  Then deliver packet to that interface.
  Else route a single copy of packet to outgoing interface
    with minimum distance label.

Pos-Reachcast-Route (dest-label)
  If (any outgoing interfaces have dist==1)
  Then deliver packet to that interface.
  Else If (number of outgoing interfaces  $\geq 2$  and
    the incoming interface has minimum distance)
  Then route once as Pos-Route(dest-label),
    and finish with Reachcast-Route().
  Else route as and finish as Reachcast-Route().

Rev-Pos-Reachcast-Route (dest-label)
  If (any outgoing interfaces have dist==1)
  Then deliver packet to that interface.
  Else If (number of outgoing interfaces  $\geq 2$  and
    the incoming interface has minimum distance)
  Then forward once as
    Rev-Pos-Route(dest-label),
    and finish as Rev-Pos-Reachcast-Route().
  Else route as and finish as Reachcast-Route().

```

Figure 6. Algorithms for reachcasting at routers.

In contrast to what occurs in multicasting, reachcast packets are not duplicated for each outgoing interface of each router. Instead, reachcast packets are forwarded at a given router only on the single interface that has the minimum distance label; if two or more interfaces have the same minimum distance label, the router selects one of them (e.g., by sorting interface numbers, as long the rule is consistent in all routers). Figure 4c illustrates reachcasting using the same example of Figure 4a; the figure indicates with the arrows the paths that reachcast packets would take from each host.

Figure 6 shows the algorithm used to route reachcast packets. The correctness of *Reachcast-Route()* follows directly from Theorem 1, because it must be true that the path taken by a reachcast packet must be the shortest path to the closest member host on the tree.

We define *positional reachcasting* as delivering packets to the nearest hosts in the direction of a specific (positionally described) destination. Such a routing service can be used to find the next closest host on the path to some other destination host, which we use in Section 5 to support end-to-end reliable multicasting.

Positional reachcasting can be accomplished by combining positional routing and reachcasting as follows: The first router with three or more interfaces in a multicast tree that receives a packet that requires positional reachcasting forwards the packet according to the positional route to the source for any packets coming in on the interface with the minimum distance label; all other routers follow the standard *Reachcast-Route()* algorithm. Figure 4b illustrates positional reachcasting from each host toward the router supporting hosts 1 and 2, forming a directed tree rooted at the router where the two hosts attach. Figure 6 shows the algorithm used to route positional reachcasts; note that positional reachcasting to the source of a source-based tree does not require positional labeling, because every router already knows the interface that leads to the source.

To prove that using *Pos-Reachcast-Route()* all hosts reach the closest host on the multicast routing tree in the direction of an arbitrary destination host, we make a stronger claim, and let the proof follow directly. The theorem be-

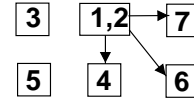


Figure 7. Reverse reachcasting from the router supporting hosts 1 and 2.

low proves that, when routers have correct labels, the union of all possible paths via a *Pos-Reachcast-Route()* between any two hosts attached to a multicast routing tree forms an in-tree rooted at the destination.

Theorem 2: The union of all paths from all hosts attached to a multicast routing tree toward an (arbitrary) destination host using *Pos-Reachcast-Route()* forms a tree.

Proof: Consider an arbitrary starting host A and an arbitrary destination host B . If A equals B , the proof is trivial. Let C be the host that a packet from A reaches using *Reachcast-Route()*. If no router in the path from A to C has more than one outgoing interface, there is only one possible path to C , and it is connected and loop free, and C must be able to reach B . Assume that there is a router in the path from A to C that has $n \geq 2$ outgoing interfaces; B necessarily lies through one of the outgoing interfaces. There are only two cases that must be considered.

Case 1: The incoming interface has the minimum distance label among all the router's interfaces in the tree. In this case, all reachcast packets arriving on the other interfaces must reach A using *Reachcast-Route()*. Furthermore, A must be the closest host to B , since it has the minimum distance label. Let D be the host reached by packets from A that are routed through that interface leading to B , and then routed via the *Reachcast-Route()* algorithm. Because D is reached through the interface leading to B , and because *Reachcast-Route()* is correct, D is necessarily closer to B than A , and D is the nearest host on that path. (Note that D does not necessarily equal C .) The path to D and B must be loop-free, because the theorem's assumption is that *Pos-Route()* is used to discover the correct interface, and this algorithm has been proven to be loop-free. Finally, because there can be only one host with a minimum distance, the path to B (thru D) is unique.

Case 2: Some other host has the minimum interface. From the proof of correctness of *Reachcast-Route()*, this other host must be C . The proof that there is a unique loop-free path from C to D to B (that does not go through A) is analogous to Case 1. Therefore, there is a unique loop-free path from A to B .

Since A and B are arbitrary nodes in the routing tree, we know there is a unique loop-free path between any two hosts. Therefore, the union of such paths forms a tree. \square

Reverse reachcasting multicasts a packet from one host, A , to all hosts that can reach A with a *Reachcast-Route()*. For example, in the case of Figure 4b, reverse reachcasting from hosts 1 and 2 would be delivered at hosts $\{3, 4, 6, 7\}$. With the help of positional routing, we can define a reverse positional reachcasting algorithm *Rev-Pos-Reachcast-Route()* (Figure 6) which forwards a packet to all hosts that can reach source host with *Pos-Reachcast-Route()*. Note that *Rev-Pos-Reachcast-Route()* uses a *Rev-Pos-Route()* algorithm that we have omitted for brevity, which simply forwards a packet out on all outgoing interfaces other than the one leading to some destination; in other words the opposite of *Pos-Route()*. Figure 7 shows the paths taken by a positional reverse reachcast from hosts 1 and 2, where 3 is the destination. Non-positional reverse reachcasting is a

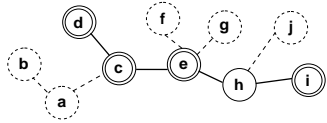


Figure 8. *c, d, e* and *i* are subscribed to a stream related to sources attached to routers *i* and *c*, packets must be routed through *h*.

special case of positional reachcasting that does not require positional routing since packets always travel away from the source.

2.5. Streams

AIM permits applications to group packets logically, and allows receivers to select the streams they wish to receive. The basic mechanism consist of subscriptions to streams. A stream can be defined in terms of a subset of sources in the multicast group, or a logical groupings of the data (e.g., audio or video) sent by the sources of the group, or both. Figure 8 illustrates how streams define subsets of the multicast tree over which receivers hear selected sources.

Establishing which routers should forward a particular stream is initiated by the receivers' subscriptions and is controlled by signaling from higher-level protocols. Each packet is associated with a particular stream by an application data unit (ADU) label assigned by a higher-level protocol to each packet. Thus, streams have a meaning and context defined by the application.

The signaling needed for subscribing to streams does not change with the multicast routing protocol, because the signaling messages needed for subscriptions are sent only over an existing multicast tree. Once a router has joined a multicast tree, it must specify which streams it chooses to receive from its neighbors. All routers are forced to receive the default global "0" stream at all times, so that there is always a method of contacting all hosts. To receive any other streams, member hosts must inform their attached routers, and routers must *subscribe* to the streams they wish to receive.

To manage streams, each router maintains a *stream table* listing the streams it currently receives, and a list of the interfaces on which packets are forwarded, for each stream. Subscribing to a stream mirrors the method used by CBT for joining a multicast group, except that subscription requests travel exclusively on the existing multicast tree.

Each stream has its own associated *stream core*, which is advertised with the list of available streams. Subscribe requests are sent unicast towards the stream core, but necessarily routed along the multicast routing tree even though they are not multicast packets. The path to the stream core may be determined by positional routing (see Section 2.1). The stream core may be the routing tree core for simplicity, or it may be the router that is the primary source of the stream for efficient routing. By selecting the primary source of the stream as the stream core, the route is contained to the part of the tree that requires the packets. Any router can send packets on any stream; the stream core is not necessarily the only source, or a source at all.

As the request travels toward the stream core, each router along the path attempts to service the request. Each router on the path to the stream core that does not receive the

requested stream, notes the incoming link of the subscribe request, forwards the subscribe request on toward the stream core, and then waits for a *subscribe-ack*. If the router does currently receive the requested stream, then it replies with a *subscribe-ack* and updates its stream table. Upon receiving a *subscribe-ack*, routers update their respective stream tables and the ack travels the reverse path back to the original requesting router. Routers must carry a stream of packets if any router downstream has subscribed to the stream. Upon receiving a *subscribe-ack* from a neighbor router, the stream table stores the name of the neighbor so that *unsubscribe* requests can be sent along the same path. The corresponding *unsubscribe* messages prune back the distribution of a particular stream. Once an *unsubscribe* message is sent, if an adjacent router has no host that requires the stream, then the *unsubscribe* message is forwarded on to the stream core.

When the multicast routing tree topology changes, some routers will receive a new parent who does not necessarily subscribe to all the same streams. Thus, topology changes can result in an interruption of service as the streams adapt to the new routing tree.

Hosts not subscribed to the stream, or even the multicast group can transmit a packet to a specific stream: it is first unicast towards the multicast address and when the packet reaches an on-tree router, the packet is sent towards the stream core, unless the on-tree router is subscribed to the stream. In this case, the packet is distributed according to the router's stream table, as if it were the source of the packet. If a stream is specified that is unknown to the on-tree router, then the packet is dropped.

The IETF is currently reviewing version 3 of the Internet Group Management Protocol (IGMPv3) [3], which allows hosts to select specific sources from which to receive (or to specifically ignore) and relates to our definition of streams. Our model is based on sources transmitting over streams and hosts selecting streams to specifically receive; this approach has the advantage that each stream can have an associated meaning in the context of the application (e.g., "stage camera" or "low bandwidth audio"). Requiring the receivers to know the stream associations, rather than every source, allows more meaningful pruning. Using stream labels allows sources to send data over more than one stream, allows each stream to carry more than one source, and permits receivers to choose streams appropriately, rather than blocking an entire source, or constructing and managing several multicast groups.

3. Implementing AIM

This section explains how to include the functionality of AIM in CBT. This requires extending its join process to assign positional and distance labels, parsing masks in order to deliver packets correctly, and finer details of implementing receiver-defined Streams. These extensions also apply to OCBT, except where noted. Similar extensions would apply to PIM Sparse Mode (PIM-SM) [6].

CBT builds a shared tree. Routers wishing to join the tree send a "join-request" to the nearest core. All off-tree routers traversed on the way are forced to join the CBT tree. Once a suitable* core is reached, the core sends a "join-ack" message to the requesting router. The join-ack is forwarded back to the requesting router along the same path by which the join-ack reached the core.

*Readers interested in what "suitable" entails are directed to the original specification of CBT [2]

AIM requires that the primary core receive a positional label of “1”, and to send a new label (as determined by the *Assign-Pos-Label()* algorithm, Figure 2) with the join-ack in a reserved “label” field. As each off-tree router receives the join-ack, the *Assign-Pos-Label()* algorithm is run again, updating the label field for the next router before the join-ack is forwarded. A join-ack may be handled by any on-tree router, and the label is set appropriately.

OCBT organizes the set of cores into logical levels, and it is possible for a router to join a secondary or tertiary level core. OCBT allows a join-ack to be sent from any level core, even before the lower level core has received a join-ack from the primary core. In this case, the join-ack would be sent before a proper positional label could be determined. A join-ack with a positional label field that starts with a zero indicates that no label could be determined yet, and the attached hosts may not send any positional multicasts (remember that all labels must start with “1”). Traditional global multicasting would still be possible while the positional label is unknown. Once the parent knows its own label, the child is informed with a new packet type called a *label-ack*. This packet type is also used when the parent receives a new label, possibly in the event that the parent’s parent had changed.

All routers set the distance labels of each interface to infinity upon initialization. Once known, the correct distance labels are piggy-backed on CBT’s periodic “keep-alive” messages between routers. A separate *distance-update* message would also suffice.

Implementations of the Streams portion of AIM must consider subscription packet loss, advertisement of available streams, assignment of unique stream labels, and recovery from network partitions.

3.1. Subscription Packet Loss

Stream subscription requests are treated as a recursive query-response process in order to handle packet loss. As the subscribe message travels towards the stream core, an *association* is formed linking pairs of routers on the path. Each pair of routers sends periodic *keep-alive* messages to each other. If no keep-alive message is received within some timeout, T_k , then the association is broken: a *subscribe-cancel* message is sent along both partitions of the association, cancelling the stream. Each time a keep-alive message is received, T_k is reset. Keep-alive messages may stop if a router fails or if the multicast routing tree topology changes.

As the subscribe-ack returns to the original router, the correct reception of the subscribe-ack by each router from its pair is confirmed with an *end-assoc* message. If the end-assoc message is not received within some timeout, the subscribe-ack is sent again. Thus, the transmission of the ack is reliable. The end-assoc message also serves to shorten the association, eventually ending the process.

3.2. Stream Advertisement

Advertisement of available streams and stream cores is based on hop-by-hop *update* messages. Each router periodically sends the messages to its neighbor routers on the multicast routing tree. For each neighbor, the update message contain a list of streams (and associated stream cores) that are upstream from that neighbor. Stream cores consider themselves upstream from all routers. Computing which streams are upstream in a source-based tree is trivial, since

only the source is allowed to be a stream core. In shared-trees, the positional label of the stream core determines which neighbor is on the path to the stream core.

Applications may choose to multicast advertisements of more important streams, however this is unscalable as the number of important streams increases. If, in building the path to a stream core, the subscribe-ack reaches the wrong router due to an old or incorrect label, then an error message is sent back, and the path is cancelled. The subscriber can then wait for an update of the correct label.

3.3. Uniqueness

Assignment of unique stream labels is administered by the multicast routing tree core, and is requested by the stream core with a *stream-request* message, which is sent unicast. Once the request message is received by the tree core, a unique number is assigned and returned as a *stream-label* message to the stream core. Eventually, the tree core receives an update stating that the stream is available. Once a label is assigned then the number cannot be used again for that multicast session until the stream is closed. When the stream is closed, this information is reflected in advertisement updates, and the tree core may recycle the number.

Since stream-label messages can be lost, some assigned numbers may be never be reclaimed. In order to gather lost numbers, the tree core sets a timer, T_{stream} , after the stream-label message is sent. T_{stream} is based on the period of update messages, a constant, and the number of hops between itself and the stream core. If the update does not contain an assigned stream when the timer expires, the number may be reclaimed.

3.4. Partitions

When there is a partition in the multicast routing tree and the tree core is unreachable for assigning unique stream labels, then stream cores can contact secondary cores. Fortunately, shared routing protocols, like CBT and PIM, already consider support of redundant tree cores. CBT and PIM-SM maintain a set of alternate “candidate cores” and “rendezvous-points”, respectively. AIM needs only enforce that labels assigned by alternate cores are always unique.

Because PIM-SM maintains an ordered list of alternate cores, ensuring uniqueness is straight-forward. Each core prefixes its assigned stream label with its index on the list. A one byte prefix would reserve enough space for a set of 256 alternate cores or rendezvous-points.

OCBT is slightly more complicated because no alternate core has a complete ordered list of cores. Instead OCBT employs a system of N levels of alternate cores. Therefore, each alternate core sends a *unique-label-req* request to the primary core at the start of the session. The primary core assigns unique sequential prefixes to each alternate core, returned in a *unique-label* message. If the primary core fails before an alternate core receives a unique prefix, then the alternate core cannot assign stream labels.

Note that we have not considered what happens when a stream core fails; choosing stream cores *and* replacing them when they fail is the responsibility of higher-layer protocols. Since stream cores are associated with hosts, higher-layers will have to deal with the host failure whether it is a stream core or not, especially for a source-based tree.

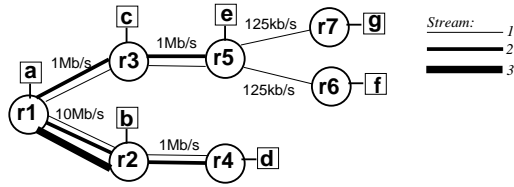


Figure 9. Layered multicast implemented as multicast streams.

4. AIM Applications

4.1. Scalable Anycasting

The reachcast service we have described is for routers already on the multicast tree, but it is easily extended to include access to routers off the multicast tree. As an example of the usefulness of reachcast, we show how such a design could function as an *anycast* service, defined by [19]. Packets transmitted on an anycast address are provided with unreliable delivery to at least one, and preferably only one, of the receivers that are members of the anycast address.

A reachcast version of anycast starts from routers that are not members of the multicast address. Packets are forwarded unicast to the multicast group as any other packet destined for a multicast address. Reach/anycast addresses appear to be standard multicast addresses to routers not on the multicast tree. A special flag indicates to on-tree routers that the packet is a reachcast, and the packet is then handled as an on-tree reachcast.

Previous designs for an anycast service [9] route packets toward the nearest host belonging to the anycast group according to the unicast routing tables. Once the packet reaches the nearest host, the packet to not forwarded any further. Because reachcast addresses appear to be normal multicast-group addresses, global anycast addresses based on reachcasts are possible. In contrast, the approach in [9] cannot support global anycast addresses.

4.2. Single-Tree Layered Multicasting

The Internet includes very different links, ranging from 28.8 Kbps modems to gigabits per second lines. Sending a uniform bandwidth stream of data may overwhelm under-resourced receivers, and penalize over-resourced receivers. One approach to this problem is to transmit the data as cumulative layered encodings rather than a single uniform encoding (e.g., [11]). Each layer of the cumulative encoding is a refinement of the previous layer's information. Hosts can then choose to receive only as many layers as their available bandwidth allows.

By separating each layer into IP-multicast groups, the responsibility of adapting to an appropriate encoding can be placed at the receivers. To this end, many adaptive protocols have been proposed that determine when the user should join and leave multicast groups (e.g., [10, 16]).

Streams can be used to support a layered multicast protocol. In this paper, we present only the framework for supporting layered multicasting within Streams. The mechanisms used by receiver hosts to decide when to adapt the fidelity of their reception can be the same as those proposed elsewhere (e.g., [10, 16]). The ADU label of each packet is

based on the level of data encoding. Once a host makes a decision to adapt, it can increase or decrease the level of its received encoding simply by subscribing or unsubscribing to the Streams.

Figure 9 illustrates using multicast streams to implement layered multicast over one multicast routing tree. For simplicity, we concentrate on a single source, so that both shared-tree routing protocols and source-based protocols would build one tree. The streams are a cumulative layering; stream 2 refines the information of stream 1, and stream 3 refines the information of stream 2. In this example, we are assuming that hosts need a 10 Mb/s connection to subscribe all three streams, a 1 Mb/s connection to subscribe to the first two streams, and a 125 Kb/s connection to subscribe to just the first stream. In the figure, host *a* is the source, host *b* has subscribed to all streams, hosts *c*, *d*, and *e* have subscribed to the first two streams, and hosts *f* and *g* have subscribed to the first stream. Notice that routers subscribed to streams 2 and 3 are a subset of the routers subscribed to stream 1, and therefore one tree is sufficient.

An important characteristic of layered multicasting is that the set of *hosts* requiring encoding layer L_i contains the set of *hosts* requiring a refined encoding layer L_{i+1} . Yet, if the two layers are transmitted over two separate IP-multicast addresses, nothing guarantees that the multicast routing trees of each address will be a subset of each other. However, if each layer is simply treated as a multicast stream, then Streams guarantees that the set of *routers* serving layer L_i contains the set of *routers* serving layer L_{i+1} . In other words, Streams constructs and maintains a single routing tree for all layers. Since all streams travel the same route, they will experience the same network conditions, which may simplify the design of flow control and reliability protocols that work over all the streams of a single session.

The savings from using a single tree come not only from simplicity, but from reduced network traffic and state. Every routing protocol incurs a certain amount of overhead. For example, PIM-SM sends periodic messages to each neighbor to capture state, topology, and membership changes; CBT uses a keep-alive mechanism between routers in the tree. Furthermore, there is certain amount of overhead to setup each multicast tree. Layered multicast, as implemented in a AIM-extended CBT, PIM, or DVMRP will require only a single tree to be maintained, which is $1/n$ of the overhead of the n separate multicast groups required in previous approaches based on standard IP-multicast (e.g., [10, 16]). The logic is similar to the savings from using one shared tree versus several source-based multicast routing trees.

5. Reliable Multicast Architecture

Reliable multicast protocols provide end-to-end reliable many-to-many communication between a number of hosts on an internetwork, over the services of an underlying unreliable multicast routing protocol. A reliable service ensures that every packet from each source is delivered correctly to each host in the *receiver set* within a finite time. If applications do not implement their own deletion scheme, this service also informs the source when packets can be deleted from memory safely.

The fundamental problem in providing a reliable multicast service is *ack implosion*: as the receiver set increases, it becomes increasingly difficult for a source to efficiently process a growing number of *repair-requests* for lost data (i.e., negative acknowledgments, or NAKs), *retransmissions*

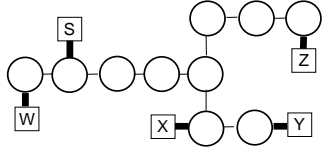


Figure 10. All three cases of packet loss correlation between routers (shown as circles) and attached hosts (shown as squares) can be found in this multicast routing tree.

of lost data to receivers, and *positive acknowledgments*, or ACKs, from receivers (stating that packet has been received correctly and may be deleted). Our approach to handling ack-implosion is *tree-based*: the receiver set is placed in a logical tree structure in order to distribute the processing of repair-requests, retransmissions, and ACKs. Tree-based reliable protocols have been shown to be the most scalable method [13].

This section introduces RMA (Reliable Multicast Architecture) which routes reliable protocol messages based on the implicit construction of an ack tree by routers. Just as with AIM and Streams, RMA does not interfere with the design or operation of multicast routing tree protocols. The ack tree RMA builds is based on the notion that the ack tree should mirror the underlying multicast routing tree in order to preserve packet loss correlation between the hosts in the session.

There are three possible relationships between any two routers, A and B , with respect to a source and the multicast routing tree connecting them:

- *Direct*: packets are routed through A before reaching B .
- *Indirect*: packets that reach A and B share some common path from the source.
- No relationship, where packets that reach A and B never travel on a common path from the source.

Generally, in direct relationships, there is high packet-loss correlation between A and B ; there is some packet loss correlation between A and B in indirect relationships; and there is no packet loss correlation (ignoring coincidence) when there is no relationship.

Figure 10 illustrates all three relationships between the routers for a source S , and four receivers W , X , Y , and Z . The routers for hosts X and Y have a direct relationship. The routers for hosts X and Z have an indirect relationship. Finally, the routers for W and X have no relationship.

When a packets are multicast, receivers closer to the source receive packets before receivers down the tree do, and there is a correlation of packet loss at nodes hanging from the multicast routing subtree of a router. The more these relationships are preserved in the ACK tree, the better the protocol performs, because latencies and retransmissions within each local group of the ACK tree have a direct correspondence with delays, congestion, and errors that occur in the routing tree.

If every parent-child relationship in an ACK tree retains the packet loss correlation present in a multicast routing tree, then we say the ACK tree is *acceptable*.

Definition 1: Let A and B be receivers, where A is on the path to B from the source S on the underlying multicast

routing tree. The relationship between A and B on the ACK tree is direct-acceptable, with respect to S , if A is not downstream from B relative to S on an ACK tree.

This definition of acceptability can only be satisfied by hosts that have a direct relationship; for example, in Figure 10, host X should be the parent of host Y on the ACK tree. The following definition applies to indirect relationships.

Definition 2: Let A and B be receivers, where A and B share a common path from a source S on the multicast routing tree, and A is closer than B is to S . The relationship between A and B on the ACK tree is indirect-acceptable, with respect to S , if B is not upstream from A relative to S on the ACK tree.

In Figure 10, if host Z were an ancestor of host X on the ACK tree, the relationship would not be indirect-acceptable.

5.1. Signaling

RMA works between routers, treating the set of hosts attached to the same router—called a *covey*—as if it consisted of a single host. RMA provides special routing for repair-requests, retransmissions, and ACKs between coveys; original transmissions are disseminated to all coveys via the multicast routing tree. Any proposed concurrent multicast protocol can be used within each covey as long as the set of hosts presents itself to RMA as one host, which is not difficult to do.

Hosts using existing reliable multicast protocols over the standard IP-multicast must consider how to handle the entire receiver set, for example, by using NAK-avoidance, or by building token-rings or trees. With the addition of RMA, each host must only consider the other hosts in its covey and the set of coveys that can send messages to the host via a positional reachcast service (see Section 2.4), which we expect to be much smaller than the actual number of hosts in the session. Our suggestion for signaling in RMA is straightforward and simple, as this design is intended for groups where n will be small. A discussion of more sophisticated mechanisms is presented subsequently.

Since any reliable multicast protocol may be used within the covey, RMA really acts as an *interface* between protocols. Similarly, the Reliable Multicast Framework (RMF) [4] was designed to act as an interface between reliable multicast protocols. The difference between RMF and our work is that RMA provides a structural support so that any end-to-end reliable protocol can operate with a large receiver set without loss in performance. RMF focuses more on defining a common packet framework within which various reliable multicast protocols can be defined. Certainly, RMA could be defined as an extension to existing work on RMF.

5.2. Retransmissions

Coveys requiring retransmissions, detected by a skip in sequence number or a bit error, use positional reachcasting to send a *repair-request* message to the nearest covey in the direction of the source of the transmission. Generally, hosts closer to the source have a better chance of receiving a packet correctly as there is less chance for failure. For example, if some covey $C1$ attached to a router $R1$ requires a retransmission, then $R1$ reachcasts a *repair-request* packet (if the source is on a shared tree, then a positional reachcast must be used). Figure 11 illustrates this example. Some covey $C2$

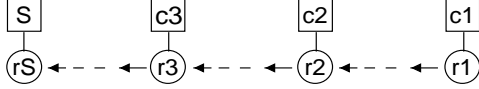


Figure 11. Covey c_1 reachcasts a repair-request towards source S . Covey c_2 receives the request.

attached to router R_2 , on the path to the source S will receive the repair-request. Notice that R_2 is necessarily closer to the source than R_1 . If C_2 is also missing the requested packet, then C_2 uses RMA to reachcast the repair-request toward the source (if it has not already done so). Some covey C_3 attached to a router R_3 will receive the repair-request, where R_3 is necessarily closer to the source than R_2 . If C_2 (or C_3) has received the requested packet correctly, then it will send the retransmission to C_1 via positional routing.

5.3. Deletion

RMA utilizes a system of *registration* and *positive acknowledgments* (ACKs) to handle safe deletion of packets at the source covey and intermediate coveys in the session. Sessions that do not wish to enforce direct deletion semantics can ignore this part of the RMA definition (e.g., a non-deleting service is provided by SRM [8]).

We define the *children* of a covey G to be the set of coveys that contacts G using a positional reachcast service. Likewise, G is the *parent* of every covey in the set of children. Deletion is not allowed at any covey until all children of that covey have stated that they have correctly received the packet with a *positive-ack* message. RMA has two deletion semantics: strong deletion means that a covey can only ACK with the minimum of sequence numbers ACKed by its children; weak deletion means that a covey may send ACKs based on the sequence number it expects next, though its children may not have ACKed the data yet. Weak deletion can fail under topology changes (see [14]), but has smaller memory requirements. Therefore, every covey must maintain a list of its current children; but it is the responsibility of children to inform parents of their existence. This is accomplished with a *register* message positionally reachcast from the child to the parent in the direction of the source. Parent's respond with a *register-ack* message, which confirms the association. The register-ack includes the IP address of the parent's router.

If a covey's router receives a new positional label, this signifies that a topology change has occurred, and possibly the covey's parents have changed. To confirm the identity of the current parents, a *parent-query* message is reachcasted in the direction of the source, and a *parent-response* message is sent in reply which includes the parent's router's IP address. If the IP addresses do not match, then an *unregister* message is sent to the old parent (via the IP address), and the child reachcasts a new register message. When a covey leaves the session, an *unregister* message is sent to every parent and child.

If there is more than one source on the shared routing tree, then it is possible that a child will have more than one parent. A subtle point is that a separate register message is not needed for every source. A covey can compare the positional label of the parent's router with the positional label of any source's router to determine if parent lies on the path to that source. If the parent does lie on the path, then it is *responsible* to that covey for storing that source's packets.

```

RMA-Route (parent-label,source-label)
  If (any outgoing interfaces have dist==1)
  Then deliver packet to that interface.
  If (incoming interface leads toward parent-label)
  Then route as and finish as
    Rev-Pos-Reachcast-Route(source-label).
  Else If (number of outgoing interfaces  $\geq 2$  and
    the incoming interface has minimum distance)
  Then route once as Pos-Route(parent-label),
    and finish as
    RMA-Route(parent-label,source-label).
  Else forward packet out on all outgoing interfaces
    that don't lead to source, and finish as
    RMA-Route(parent-label,source-label).

```

Figure 12. Algorithm at router for one child to multicast to its siblings and parent in RMA.

A new register message is required only if no current parent of the covey is responsible for a new source.

5.4. Sophisticated Signaling

The motivation for contacting exactly one other covey for retransmissions is that it keeps RMA simple. Most reliable protocols were designed to handle a very large receiver set, and thus have complicated mechanisms for retaining scalability (e.g., NAK-avoidance, or building rings).

However, just as RMA allows any known protocol to operate within the coveys, RMA allows the same between any parent and its children. In other words, we have designed RMA as simple as possible, but the framework is present for more complicated interaction between coveys. Figure 12 shows the *RMA-Route()* algorithm for routing packets from one child to all its siblings and its parent. The algorithm calls for the label of the source of the multicast and the label of the parent. Therefore, NAK-avoidance [8] or a Ring-based [21] approach between a parent and its children would be feasible, scalable and transparent to the hosts; the construction of a separate multicast group is unnecessary.

A sophisticated framework is available for the retransmission strategy as well. Instead of sending retransmissions only to requesting coveys one-by-one, retransmissions may be sent to all children via the *Rev-Pos-Reachcast-Route()* algorithm from the parent. Another option is to send retransmissions to the *entire subtree* [15] each covey heads as a positional multicast message (see Section 2.2), an approach which greatly reduces the latency of receiving retransmissions. Multicasting retransmissions must be done carefully, as it may cause a lot of unnecessary traffic. RMA is different than previous protocols calling for multicast retransmissions to a large set of receivers (e.g., [15, 8]) because the ack tree built by RMA is always acceptable (this trivially follows from the definition of positional reachcasting). In an acceptable tree all hosts downstream on the implicit ack tree are necessarily downstream on the multicast routing tree, which means that packet loss correlation is maintained between a parent and all its descendants. Therefore, retransmissions received by descendants are very likely not to be extraneous traffic. No prior tree-based protocol builds acceptable ack trees (e.g., RMTP [15], TMTP [22], or Lorax [14]).

5.5. Comparison to Other Work

A previous approach to providing reliability is proposed by Kasera *et al* [12], which allocates separate multicast groups for each retransmission. All receivers must proactively request a missing packet by joining and leaving each group. As an alternative to avoiding the network traffic involved in joining and leaving, a system of *local filtering* is proposed. Host join or leave the retransmission groups locally, i.e., *all* retransmissions reach every host's local network, but are not necessarily forwarded on to the host. While, this reduces the work of the receiver, it floods the network with all retransmissions, which reduces throughput. Furthermore, the number of separate multicast groups that must be maintained increases with the number of sources and with the number of receivers.

In contrast, our approach does not set up, maintain, or tear down separate multicast groups, and our scheme only requires that the host closest to the point of loss on the network realize a packet is missing, all other receivers will be sent the retransmissions automatically via subtree multicast. The special routing performed is not only simple to implement, it retains independence of protocol layers. RMA works over a shared multicast routing tree as well as source-based multicast routing trees. Furthermore, the interface between covers is robust in that it can be simple, or may be designed to handle any previously proposed protocol for handling reliable communication (e.g., NAK-avoidance, or rings).

The control overhead involved for RMA is equal to the cost of maintaining the distance and positional labels, which piggyback over topology messages, and the cost is insignificant. If deletion is desired, registration messages occur between exactly two routers in the tree at all times. RMA overhead is independent of the number of sources, receivers, or packet loss rate. Registration messages are sent only when (and where) the routing tree changes.

Recent work by Papadopoulos *et al* [17] is similar to RMA in that it also considers special routing of reliable multicast protocol messages. However, unlike RMA, that proposal has difficulty working over a shared multicast routing tree protocol like CBT or OCBT. Furthermore, it is unable to avoid sending extraneous retransmissions to some receivers, and is unable to guarantee that all relationships between hosts are acceptable in the implicit ack tree they build. RMA avoids these problems with positional routing and reachcast routing from AIM (Section 2).

6. Conclusion

We have shown the versatility of new multicast-based routing services, which are made possible by extending the IP-multicast architecture to include group-relative addressing information. The lightweight mechanisms of positional routing and reachcasting that make up AIM are not simply methods to enhance delivery options; they are robust enough to act as building blocks for other IP Multicast enhancements, as we have shown with RMA. Group-relative addressing has relevance to several services that build on multicast services; we have considered subtree multicasting, anycasting, layered multicast, routing tree pruning, and reliable multicast in this paper. In all cases, cooperation and communication between network layers is accomplished efficiently and without violating layer independence.

References

- [1] E. Bakker, J. van Leeuwen, and R. B. Tan. Prefix routing schemes in dynamic networks. *Computer Networks and ISDN Systems*, 26(4):403–21, December 1993.
- [2] A. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT): An architecture for scalable inter-domain multicast routing. In *Proc. ACM SIGCOMM'93*, pages 85–95, October 1993.
- [3] B. Cain, S. Deering, and A. Thyagarajan. Internet group management protocol, version 3. Internet draft, January 1997.
- [4] B. DeCleene, J. Kurose, D. Towsley, S. Bhattacharaya, T. Friedman, M. Keaton, and D. Rubenstein. RMF: A transport protocol framework for reliable multicast applications. Internet Draft <draft-decleene-rmf-00.txt>, March 1997.
- [5] S. Deering. Host extensions for IP multicasting. Request For Comments 1112, August 1989.
- [6] S. Deering *et al*. An architecture for wide-area multicast routing. In *Proc. ACM SIGCOMM'94*, pages 126–135, 1994.
- [7] S. E. Deering. Multicast routing in internetworks and extended lans. In *Proc. ACM SIGCOMM'88*, volume 18, pages 55–64, 1988.
- [8] S. Floyd *et al*. A reliable multicast framework for lightweight sessions and application level framing. In *Proc. ACM SIGCOMM'95*, pages 342–356, August 1995.
- [9] R. Hinden and S. Deering. IP version 6 addressing architecture. Request for Comments 1884, December 1995.
- [10] D. Hoffman and M. Speer. Hierarchical video coding for packet-switching networks—an integrated approach. In *Proc. SPIE Conference on Visual Communications and Image Processing*, September 1996.
- [11] H. Kanakia, P. P. Mishra, and A. Reibman. An adaptive congestion control scheme for real-time packet video transport. In *Proc. ACM SIGCOMM'93*, pages 20–31, September 1993.
- [12] S. Kasera, J. Kurose, and D. Towsley. Scalable reliable multicast using multiple multicast groups. Technical Report TR-96-73, Univ. Mass., October 1996.
- [13] B. N. Levine and J.J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *ACM Multimedia Systems Journal*, 1997. To appear.
- [14] B. N. Levine, D. Lavo, and J.J. Garcia-Luna-Aceves. The case for reliable concurrent multicasting using shared ack trees. In *Proc. ACM Multimedia*, pages 365–376, November 1996.
- [15] J. Lin and S. Paul. RMTP: A reliable multicast transport protocol. In *Proc. IEEE Infocom*, pages 1414–1425, March 1996.
- [16] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proc. ACM SIGCOMM'96*, pages 117–130, August 1996.
- [17] C. Papadopoulos, G. Parulkar, and G. Varghese. An error control scheme for large-scale multicast. Unpublished technical report, Washington University, St Louis, MO, August 1997. <http://www.csrc.wustl.edu/~christos/PostScriptDocs/Infocom98.ps.Z>.
- [18] M. Parsa and J.J. Garcia-Luna-Aceves. A protocol for scalable loop-free multicast routing. *IEEE Journal on Selected Areas in Communications*, 15(3):316–331, April 1997.
- [19] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. Request For Comments 1546, November 1993.
- [20] C. Shields and J.J. Garcia-Luna-Aceves. The ordered core based tree protocol. In *IEEE Infocom'97*, April 1997.
- [21] B. Whetten, S. Kaplan, and T. Montgomery. A high performance, totally ordered multicast protocol. In *Theory and Practice in Distributed Systems, International Workshop, LNCS 938*, September 1994.
- [22] R. Yavatkar, J. Griffioen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proc. ACM Multimedia*, pages 333–44, November 1995.