Multicast Routing in Internetworks and Extended LANs

Stephen E. Deering
Computer Systems Laboratory
Stanford University

Abstract

Multicasting is used within local-area networks to make distributed applications more robust and more efficient. The growing need to distribute applications across multiple, interconnected networks, and the increasing availability of high-performance, high-capacity switching nodes and networks, lead us to consider providing LAN-style multicasting across an internetwork. In this paper, we propose extensions to two common internetwork routing algorithms—distance-vector routing and link-state routing—to support low-delay datagram multicasting. We also suggest modifications to the single-spanning-tree routing algorithm, commonly used by link-layer bridges, to reduce the costs of multicasting in large extended LANs. Finally, we show how different link-layer and network-layer multicast routing algorithms can be combined hierarchically to support multicasting across large, heterogeneous internetworks.

1 Introduction

The multicast capability of local-area networks such as Ethernet [8] provides two important benefits to distributed applications:

1. When an application must send the same information to more than one destination, multicasting is more efficient than unicasting: it reduces the transmission overhead on the sender and the network, and it reduces the time it takes for all destinations to receive the information.

2. When an application must locate, query, or send information to one or more hosts whose addresses are unknown or changeable, multicasting serves as a simple, robust alternative to configuration files, name servers, or other binding mechanisms.

Multicasting applications have proliferated in those environments in which the multicast capability has been made available to application programmers, whether in the form of process groups in the V System [5], UDP broadcast sockets in Berkeley UNIX [20], or NetBIOS multicast datagrams in MS-DOS [16]. In some cases, multicasting has played an important role in organizing the underlying operating systems and protocols themselves, as well as being offered as a service for applications.[1]

For networks in which all hosts share a common transmission channel, such as bus, ring, or satellite networks, the multicast capability is provided trivially and at the same cost to the network as unicasting. When such networks are interconnected by store-and-forward packet switches, multicasting across the resulting internetwork often requires the commitment of additional switching and transmission resources, beyond those required for unicasting. However, as those resources become more abundant, in the form of fast packet switches, cheap memories, and high-bandwidth local and long-haul communication links, an economic argument for denying users the benefits of an internetwork multicast capability becomes harder to sustain.

Link-layer bridges, such as the DEC LANBridge 100 [12] and the Vitalink TransLAN [11], have taken advantage of the improving economics of communication to extend LAN performance and LAN functionality—including multicast—across multiple networks. That is not yet the case with network-layer routers, such as DoD IP Gateways [14] or ISO Intermediate Systems [18]. Therefore, when moving multicast-based applications to an environment that includes network-layer routers, it is currently necessary to give up the efficiency of multicasting and to replace the flexible binding capability of multicasting with more complicated or fragile mechanisms. This paper addresses that problem by proposing extensions to two common routing algorithms used by network-layer routers—distance-vector routing and link-state routing—to provide LAN-style multicasting across datagram-based internetworks. We also suggest modifications to link-layer bridge routing to improve the efficiency of multicasting in large extended LANs.

In the next section of this paper we define what we mean by "LAN-style multicasting." In Section 3 we describe the environment in which multicast routing is to take place. Then follow three sections, describing specific multicast extensions to the single-spanning-tree, distance-vector, and link-state routing algorithms. In Section 7, we describe how a variety of link-layer and network-layer multicast routing schemes may be combined to support multicasting in a large, heterogeneous internetwork. In Section 8 we call attention to other work in the same area, and in the concluding section we summarize our results and point the way to further work.

[1] Some of these systems have implemented multicasting by using the local-area network's *broadcast* facility, relying on software filtering in the receiving hosts. This approach incurs undesirable overhead on those hosts that must receive and discard unwanted packets, overhead that gets worse as more and more applications use multicasting. Fortunately, this problem can be avoided in modern LANs, such as Ethernet and other networks conforming to the IEEE 802 [15] standards, which provide multicast addresses that can be recognized and filtered by host interface hardware.

2 Desired Properties for Internetwork Multicasting

Existing multicast-based distributed applications have been developed in the LAN environment. To support the migration of such applications to an internetwork environment, it is desirable to retain, to the degree possible, the following important properties of LAN multicasting:

- *Group addressing.* In a LAN, a multicast packet is sent to a group address which identifies a set of destination hosts. The sender need not know the membership of the group and need not itself be a member of the group. There is no restriction on the number or location of hosts in a group. Hosts can join and leave groups at will, with no need to synchronize or negotiate with other members of the group or with potential senders to the group.

  With such group addressing, multicasting can be used for such purposes as locating a resource or a server when its specific address is unknown, searching for information among a dynamically-changing set of information providers, or distributing information to an arbitrarily-large, self-selected set of information consumers.

- *High probability of delivery.* In a LAN, the probability that a member of a group successfully receives a multicast packet sent to the group is usually the same as the probability that the member successfully receives a unicast packet sent to its individual address. Furthermore, that probability of successful reception by every member is very high, in the absence of partitioning. This property allows the designers of end-to-end reliable multicast protocols to assume that a small number of retransmissions of a multicast packet will result in successful delivery to all destination group members that are up and reachable. The probability of damage, duplication, or misordering of multicast packets in a LAN is very low, but not necessarily zero; recovery from such events is also the responsibility of end-to-end protocols, to the extent required by particular applications.

  The probability of successful multicast delivery in an internetwork may well decrease as the distance between sender and group members increases, but it must stay within bounds that allow successful recovery by end-to-end protocols.

- *Low delay.* LANs impose very little delay on the delivery of multicast packets. This is an important property for a number of multicast applications, such as distributed conferencing, parallel computing, and resource location. Also, the delay between when a host decides to join a group and when it can start receiving packets addressed to that group, called the *join latency*, is very low in a LAN, usually just the time required to update a local address filter. Low join latency is important for certain applications, such as those that use multicasting to communicate with migrating processes or mobile hosts.

  The delay properties of large internetworks are, inevitably, worse than LANs because of their greater geographic extent and their greater number of links and switches. However, the use of high-speed packet switches and low-delay long distance communication links such as optical fibers has the potential to significantly reduce the gap between local-area network and internetwork delay characteristics. In order to exploit that potential, it is important that internetwork multicast routing algorithms produce low-delay routes, in preference to routes that maximize bandwidth or minimize network resource consumption. The availability of bandwidth and other network resources keeps improving; delay is the limiting factor for wide-area communication.

The large scale and multi-hop nature of internetworks motivates a simple extension to LAN multicasting semantics to allow senders to limit the distance a multicast packet may travel. Internetwork datagram protocols, such as DoD IP [24] and ISO CLNP [17], include a *time-to-live* (TTL) field in the packet header for the purpose of bounding the amount of time a packet may be in transit. By using a very small TTL value, a sender may limit the "scope" of a multicast packet to reach nearby group members only.[2] This can be of benefit to the internetwork, by reducing the amount of multicast traffic that has to be carried long distances, and it can be of benefit to the sender, by reducing the number of responders when querying a large group. Even when it is desired to reach an entire group, if the sender knows that all the members are nearby, use of a small TTL can help to reduce the delivery costs incurred under some multicast routing schemes.

3 Assumed Environment for Internetwork Multicasting

We assume an environment of multi-access networks (LANs and, possibly, satellite networks) interconnected in an arbitrary topology by packet switching nodes (bridges and/or routers). Point-to-point links (both physical links such as fiber-optic circuits and virtual links such as X.25 virtual circuits) may provide additional connections between the switching nodes, or from switching nodes to isolated hosts, but almost all hosts are directly connected to LANs.

The LANs are assumed to support *intra*network multicasting. The hosts have address filters in their LAN interfaces which can recognize and discard packets destined to groups in which the hosts have no interest, without interrupting host processing. Bridges and routers attached to LANs are capable of receiving *all* multicast packets carried by the LAN, regardless of destination address.

Link-layer bridges perform their routing function based on LAN addresses that are unique across the collection of interconnected LANs. Network-layer routers perform routing based on globally-unique internetwork addresses which are mapped to locally-unique LAN addresses for transmission across particular LANs. We assume that globally-unique internetwork *multicast* addresses can be mapped to corresponding LAN multicast addresses according to LAN-specific mapping algorithms. Ideally, each internetwork multicast address maps to a different LAN address; in cases where address-space constraints on a particular LAN force a many-to-one mapping of internetwork to LAN multicast addresses, the hosts' address filters may be less effective, and additional filtering must be provided in host software.

[2] An interesting and useful application of TTL scope control is "expanding ring searching", a concept described by Boggs in his dissertation on internetwork broadcasting [3]. An example of its use is searching for the nearest name server: a host multicasts a name server query, starting with a TTL that reaches only its immediate neighborhood, and incrementing the TTL on each retransmission to reach further and further afield, until it receives a reply.

4 Single-Spanning-Tree Multicast Routing

Link-layer bridges [11, 12] transparently extend LAN functionality across multiple interconnected LANs, possibly separated by long distances. To maintain transparency, bridges normally propagate every multicast and broadcast packet across every segment of the extended LAN. This is considered by some to be a *dis*advantage of bridges, because it exposes the hosts on each segment to the total broadcast and multicast traffic of all the segments. However, it is the misguided use of broadcast packets, rather than multicast packets, that is the threat to host resources; multicast packets can be filtered out by host interface hardware. Therefore, the solution to the host exposure problem is to convert broadcasting applications into multicasting applications, each using a different multicast address.

Once applications have been converted to use multicast, it is possible to consider conserving bridge and link resources by conveying multicast packets across only those links necessary to reach their target membership. In small bridged LANs, bridge and link resources are usually abundant; however, in large extended LANs that include lower-bandwidth long-haul links or that have a lot of multicast traffic for groups that reside in small sub-regions of the extended LAN, it may be of great benefit not to send multicast packets everywhere.

Bridges typically restrict all packet traffic to a single spanning tree, either by forbidding loops in the physical topology or by running a distributed algorithm among the bridges to compute a spanning tree [23]. When a bridge receives a multicast or broadcast packet, it simply forwards it onto every incident branch of the tree except the one on which it arrived. Because the tree spans all segments and has no loops, the packet is delivered exactly once (in the absence of errors) to every segment.

If bridges knew which of their incident branches led to members of a given multicast group, they could forward packets destined to that group out those branches only. Bridges are able to learn which branches lead to *individual* hosts by observing the source addresses of incoming packets. If group members were to periodically issue packets with their group address as the source, the bridges could apply the same learning algorithm to group addresses.

For example, assume that there is an *all-bridges* group B to which all bridges belong. Each host that is a member of a group G may then inform the bridges of its membership by periodically transmitting a packet with source address G, destination address B, packet type *membership-report*, and no user data.

Figure 1 shows how this works in a simple bridged LAN with a single group member. LANs a, b, and c are bridged to a backbone LAN d. Any membership report issued by the one group member on LAN a is forwarded to the backbone LAN by the bridge attached to a, to reach the rest of the *all-bridges* group. There is no need to forward the membership report to LANs b or c because they are leaves of the spanning tree which do not reach any additional bridges. (Bridges are able to identify leaf LANs either as a result of their tree-building algorithm or by periodically issuing reports of their own membership in the the *all-bridges* group.)

After the membership report has reached all bridges, they each know which direction leads to the member of G, as illustrated by the arrows in Figure 1. Subsequent transmission of multicast packets destined to G are forwarded only in the direction of that membership. For example, a multicast packet to G originating

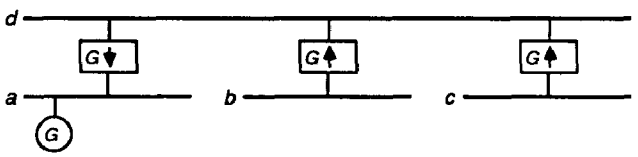

Figure 1: Bridged LAN with One Group Member

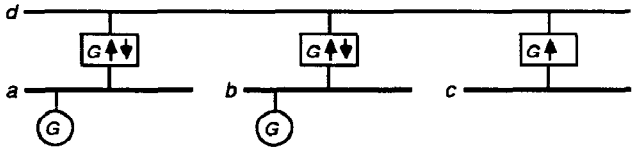

Figure 2: Bridged LAN with Two Group Members

on LAN b will traverse d and a, but not c. A multicast to G originating on a will not be forwarded at all.

Figure 2 shows the state of bridge knowledge after a second member joins the group on LAN b. Now multicast packets to G will be conveyed towards LANs a and b, but not towards c.

This multicast routing algorithm requires little extra work or extra space in the bridges. Typical learning bridges maintain a table of *unicast* addresses. Each table entry is a triple:

(*address, outgoing-branch, age*)

where the *age* field is used to detect stale data. The source address and source branch of each incoming packet is installed in the table, and the destination address of each arriving unicast packet is looked-up in the table to determine an outgoing branch. To support multicasting, the table must also hold multicast addresses. As seen in Figure 2, a single multicast address may have multiple outgoing branches (and *age* fields, as discussed below), so the table entries become variable-length records of the form:[3]

(*address*, (*outgoing-branch, age*),
(*outgoing-branch, age*), ...)

An arriving group membership report causes a table entry for its source address to be installed or updated. The destination address of an arriving multicast packet is looked-up in the table to determine the set of outgoing branches. The branch over which the multicast packet arrived is always deleted from the set of outgoing branches before forwarding.

The *age* field in table entries for multicast addresses is handled somewhat differently than for unicast addresses. When a bridge receives a unicast packet, if its destination address is absent from the table, or if its table entry has expired (i.e., its *age* exceeds some threshold), the packet is forwarded out *all* branches except the incoming one. It is expected that responding traffic from the destination will later allow the bridge to learn its location. When a bridge receives a multicast packet, on the other hand, it forwards the packet over only those branches that are identified by non-expired table entries. Expired entries are treated as evidence that there are no longer any members reachable over that branch. Therefore, group members must regularly report their memberships at intervals less than the membership expiry threshold.

[3]Many bridges are designed to connect only two, or some other small number, of links; for them, it may be acceptable to use fixed, maximum-sized records, in order to simplify memory management.

The overhead of membership reporting traffic is determined by the choice of reporting interval T_{report}—the larger T_{report}, the less the reporting overhead. On the other hand, choosing a large T_{report} has the following drawbacks:

- The expiry threshold T_{expire} for bridge table entries should be a multiple of T_{report} in order to tolerate occasional loss of membership reports. The larger T_{expire}, the longer a bridge will continue to forward multicast packets onto a particular branch after there are no longer any members reachable along that branch. This is not particularly serious, given that hosts are protected from unwanted traffic by their address filters.
- If a host is the first member of a group on a particular LAN and its first one or two membership reports are lost due to transmission errors, the bridges will be unaware of its membership until one or two times T_{report} has passed. This fails to meet the goal of low join latency, stated in Section 2. It can be avoided by having hosts issue several membership reports in close succession when they first join a group.
- If the spanning tree changes due to a bridge or LAN coming up or going down, the multicast entries in the bridge tables may become invalid for as long as T_{expire}. This problem can be avoided by having the bridges revert to broadcast-style forwarding for a period of T_{expire} after any topology change.

Therefore, none of these drawbacks is serious enough to prevent the use of a relatively large T_{report}, say on the order of minutes rather than seconds.

There is another technique that can be used to reduce the reporting traffic, apart from increasing T_{report}. When issuing a membership report for group G, a host initializes the destination address field to G, rather than the *all-bridges* address. The bridge(s) directly attached to the reporting member's LAN then replace the G with the *all-bridges* address before forwarding to the other bridges. (A bridge can recognize such reports by the fact that the source and destination are the same group address.) This allows other members of the same group on the same LAN to overhear the membership report and suppress their own, superfluous reports. In order to avoid unwanted synchronization of membership reports, whenever such a report is transmitted on a LAN all members of the reported group on that LAN set their next report timer to a random value in a range around T_{report}. The next report for that group is issued by whichever member times out first, at which time new random timeouts are again chosen. Thus, the reporting traffic originating on each LAN is reduced to one report per group present, rather than one report from *every member* of every group present, in every T_{report} period. This is a significant reduction in the common case where a single group has more than one member on a single LAN.

To get a feeling for the costs of this algorithm, assume that a typical extended LAN consists of 10 segments, on which each host belongs to 5 groups, each segment has members of 20 different groups, there are 50 groups in total, and the membership reporting interval T_{report} is 200 seconds. Then:

- The overhead on hosts is the transmission or reception of one membership report packet every 40 seconds.
- The overhead on leaf segments and on bridge interfaces to leaf segments is one membership report packet every 10 seconds.
- The overhead on non-leaf segments and on bridge interfaces to non-leaf segments is the sum of the reporting traffic from each segment, that is one membership report packet every second.
- The storage overhead in each bridge is 50 group address entries.

Such costs are insignificant compared to the available bandwidth and bridge capacity in current extended LAN installations. Furthermore, the overheads on hosts and leaf segments are independent of the total number of segments; extended LANs with hundreds of segments would see greater overheads only on the "backbone" segments, not on the (presumably) more numerous leaf segments to which most hosts would be connected.

The bridge multicast routing algorithm as described requires that hosts be modified to issue membership reports for those groups they belong to. This compromises the transparency property that is one of the important features of link-layer bridges. However, if hosts are to be modified anyway to use multicast rather than broadcast, the membership reporting protocol might reasonably be implemented at the same time. The reporting is best handled at the lowest level in the host operating system, such as the LAN device driver, in order to minimize host overhead. Future LAN interfaces might well provide the membership reporting service automatically, without host involvement, as a side-effect of setting the multicast address filter. Conversely, nonconforming hosts might be accommodated by allowing manual insertion of membership information into individual bridge tables.

5 Distance-Vector Multicast Routing

The distance-vector routing algorithm, also known as the Ford-Fulkerson [9] or Bellman-Ford [2] algorithm, has been used for many years in many networks and internetworks. For example, the original Arpanet routing protocol [22] was based on distance-vector routing, as was the Xerox PUP Internet [4] routing protocol. It is currently in use by Xerox Network Systems internetwork routers [27], some DARPA Internet core gateways [14], and numerous UNIX systems running Berkeley's *routed* internetwork routing process [13], to name only a few.

Routers that use the distance-vector algorithm maintain a routing table which contains an entry for every reachable destination in the internetwork. A "destination" may be a single host, a single subnetwork, or a cluster of subnetworks. A routing table entry typically looks like:

(*destination, distance, next-hop-address, next-hop-link, age*)

Distance is the distance to the destination, typically measured in hops or some other unit of delay. *Next-hop-address* is the address of the next router on the path towards the destination, or the address of the destination itself if it shares a link with this router. *Next-hop-link* is a local identifier of the link used to reach *next-hop-address*. *Age* is the age of the entry, used to time out destinations that become unreachable.

Periodically, every router sends a routing packet out each of its incident links. For LAN links, the routing packet is usually sent as a local broadcast or multicast in order to reach all neighboring routers. The packet contains a list of (*destination, distance*) pairs

(a "distance vector") taken from the sender's routing table. On receiving a routing packet from a neighboring router, the receiving router may update its own table if the neighbor offers a new, shorter route to a given destination, or if the neighbor no longer offers a route that the receiving router had been using. By this interaction, routers are able to compute shortest-path routes to all internetwork destinations. (This brief description leaves out several details of the distance-vector routing algorithm which are important, but not relevant to this presentation. Further information can be found in the references cited above.)

One straightforward way to support multicast routing in a distance-vector routing environment would be to compute a single spanning-tree across all of the links and then use the multicast routing algorithm described in the previous section. The spanning tree could be computed using the same algorithm as link layer bridges or, perhaps, using one of Wall's algorithms [26] for building a single tree with low average delay. However, in a general topology that provides alternate paths, no single spanning tree will provide minimum-delay routes from all senders to all sets of receivers. In order to meet our goal of low-delay multicasting, and to provide reasonable semantics for TTL scope control, we require that a multicast packet be delivered along a shortest-path (or an almost-shortest-path) tree from the sender to the members of the multicast group.

There is potentially a different shortest-path tree from every sender to every multicast group. However, every shortest-path multicast tree rooted at a given sender is a subtree of a single shortest-path *broadcast* tree rooted at that sender. In this section, we use that observation as the basis for a number of refinements to Dalal and Metcalfe's *reverse path forwarding* broadcast algorithm [6] which take advantage of the distance-vector routing environment to provide low-delay, low-overhead multicast routing.

5.1 Reverse Path Flooding (RPF)

In the basic reverse path forwarding algorithm, a router forwards a broadcast packet originating at source S if and only if it arrives via the shortest path from the router back to S (i.e., the "reverse path"). The router forwards the packet out all incident links except the one on which the packet arrived. In networks where the "length" of each path is the same in both directions, for example when using hop counts to measure path length, this algorithm results in a shortest-path broadcast to all links.

To implement the basic reverse path forwarding algorithm, a router must be able to identify the shortest path from the router back to any host. In internetworks that use distance-vector routing for unicast traffic, that information is precisely what is stored in the routing tables in every router. Furthermore, most implementations of distance-vector routing use hop counts as their distance measure. Thus, reverse path forwarding is easily implemented and effective at providing shortest-path broadcasting in most distance-vector routing environments. (Distance metrics other than hop counts may also support shortest-path or almost-shortest-path broadcasting, as long as the resulting path lengths are the same or almost the same in both directions.)

As described, reverse path forwarding accomplishes a *broadcast*. To use the algorithm for multicasting, it is enough simply to specify a set of internetwork multicast addresses that can be used as packet destinations, and perform reverse path forwarding on all packets destined to such addresses. Hosts choose which groups they wish to belong to, and simply discard all arriving packets addressed to any other group.

The reverse path forwarding algorithm as originally specified in [6] assumes an environment of point-to-point links between routers, with each host attached to its own router. In the internetwork environment of interest here, routers may be joined by multi-access links as well point-to-point links, and the majority of hosts reside on multi-access links (LANs). It is possible and desirable to exploit the multicast capability of those multi-access links to reduce delay and network overhead, and to allow host interface hardware to filter out unwanted packets. To accomplish this, whenever a router (or an originating host) forwards a multicast packet onto a multi-access link, it sends it as a local multicast, using an address derived from the internetwork multicast destination address. In this way, a single packet transmission can reach all member hosts that may be present on the link. Routers are assumed to be able to hear all multicasts on their incident links, so the single transmission also reaches any other routers on that link. Following the reverse path algorithm, a receiving router forwards the packet further only if it considers the sending router to be on the shortest path, i.e., if the sending router is the *next-hop-address* to the originator of the multicast.

The major drawback of the basic reverse path forwarding algorithm (as a broadcast mechanism) is that any single broadcast packet may be transmitted more than once across any link, up to the number of routers that share the link. This is due to the forwarding strategy of flooding a packet out all links other than its arriving link, whether or not all the links are part of the shortest-path tree rooted at the sender. This problem is addressed in [6] and also in the following subsection. To distinguish the basic flooding form of reverse path forwarding from later refinements, we refer to it as *reverse path flooding* or *RPF*.

5.2 Reverse Path Broadcasting (RPB)

To eliminate the duplicate broadcast packets generated by the RPF algorithm, it is necessary for each router to identify which of its links are "child" links in the shortest reverse-path tree rooted at any given source S. Then, when a broadcast packet originating at S arrives via the shortest path back to S, the router can forward it out only the child links for S.

In [6], Dalal and Metcalfe propose a method for discovering child links which involves each router periodically sending a packet to each of its neighbors, saying, "You are my next hop to these destinations." We propose a different technique for identifying child links which uses only the information contained in the distance-vector routing packets normally exchanged between routers.

The technique involves identifying a single "parent" router for each link, relative to each possible source S. The parent is the one with the minimum distance to S. In case of a tie, the router with the lowest address (arbitrarily) wins. Over each of its links, a particular router learns each neighbor's distance to every S—that is the information conveyed in the periodic routing packets. Therefore, each router can independently decide whether or not it is the parent of a particular link, relative to each S. (This is the same technique as used to select "designated bridges" in Perlman's spanning tree algorithm for LAN bridges [23], except that we build multiple trees, one for each possible source.)

How this works can be seen in the internetwork fragment illustrated in Figure 3. In this example, three routers x, y and z

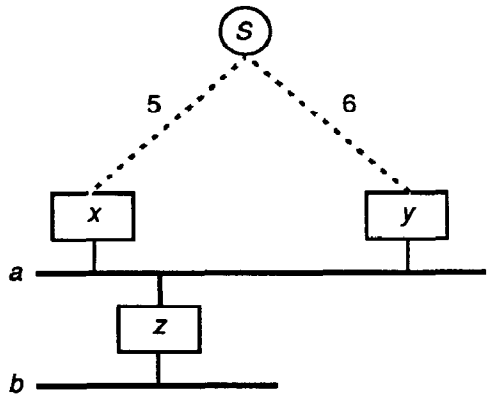

Figure 3: Reverse Path Forwarding Example

are attached to a LAN a. Router z is also connected to a leaf LAN b. The dashed lines represent the shortest paths from x and from y to a particular source of broadcast packets S, somewhere in the internetwork. The distance from x to S is 5 hops and the distance from y to S is 6 hops. Router z is also 6 hops from S, via x.

To understand the problem being solved, first consider what happens under the basic RPF algorithm. Both x and y receive a broadcast from S over their shortest-path links to S, and both of them forward a copy onto LAN a. Therefore, any hosts attached to a receive duplicate copies of all packets broadcast from S. Router z, however, will forward only one of the copies, the one from x, onto LAN b, because x is z's *next-hop-address* for S.

Now consider how the parent-selection technique solves the problem. All three routers, x, y, and z, periodically send distance-vector routing packets across LAN a, reporting their distance to every destination. From these packets, each of them learns that x has the shortest distance to S. Therefore, only x adopts LAN a as a child link, relative to S; y no longer forwards superfluous broadcasts from S onto LAN a.

If both x and y had a distance of 5 hops to S, the one with the lowest address (say x) would be the parent of LAN a. Note that, in this case, z might choose either x or y as its *next-hop-address* to S. In some implementations of distance-vector routing, z might even alternate between using x and using y to reach S, in order to spread packet traffic over multiple, equally-short paths. However, for the purpose of reverse-path forwarding, every router has to choose a *single* shortest reverse path for each source S. The tie-breaking scheme for parent selection implies that a router with multiple shortest-path routes to S should use the one whose *next-hop-address* is the lowest when deciding whether or not to forward a broadcast from S. Thus, in the example, z forwards broadcasts onto LAN b only if they come from x.

The parent-selection technique for eliminating duplicates requires that one additional field, *children*, be added to each routing table entry. *Children* is a bit-map with one bit for each incident link. The bit for link l in the entry for *destination* is set if l is a child link of this router for broadcasts originating at *destination*.

We call this variant of the algorithm *reverse path broadcasting* or *RPB* because it provides a clean (i.e., no duplicates) broadcast to every link in the internetwork, assuming no transmission errors or topology disruptions.

5.3 Truncated Reverse Path Broadcasting (TRPB)

The RPF and RPB algorithms implement shortest-path broadcasting. They can be used to carry a multicast packet to all links in an internetwork, relying on host address filters to protect the hosts from receiving unwanted multicasts. In a small internetwork with infrequent multicasting, this may be an acceptable approach, just as link-layer bridges that send multicast packets everywhere are acceptable to some. However, as in the case of large extended LANs, it is desirable in large internetworks to conserve network and router resources by sending multicast packets only where they are wanted. This requires that hosts inform the routers of their group memberships.

To provide shortest-path multicast delivery from source S to members of group G, the shortest-path broadcast tree rooted at S must be pruned back to reach only as far as those links that have members of G. This could be accomplished by requiring members of G to send membership reports back up the broadcast tree towards S, periodically; branches over which no membership reports were received would be deleted from the tree. Unfortunately, this would have to be done separately for every group, over every broadcast tree, resulting in reporting bandwidth and router memory requirements on the order of the total number of groups times the total number of possible sources.

In this subsection, we describe an alternative in which only non-member *leaf* networks are deleted from each broadcast tree. It has modest bandwidth and memory requirements and is suitable for internetworks in which leaf network bandwidth is a critical resource. The next subsection addresses the problem of more radical pruning.

For a router to forgo forwarding a multicast packet over a leaf link that has no group members, the router must be able to (1) identify leaves and (2) detect group membership. Using the algorithm of the previous subsection, a router can identify which of its links are *child* links, relative to a given source S. Leaf links are simply those child links that no other router uses to reach S. (Referring back to Figure 3, LAN b is an example of a leaf link for the broadcast tree rooted at S.) If we have every router periodically send a packet on each of its links, saying, "This link is my next hop to these destinations," then the parent routers of those links can tell whether or not the links are leaves, for each possible destination. In the example, router z would periodically send such a packet on LAN a, saying, "This link is my next hop to S". Hence, router x, the parent of LAN a, would learn that LAN a is not a leaf, relative to S.

Some implementations of distance-vector routing already implicitly convey this next hop information in their normal routing packets, by claiming a distance of *infinity* for all destinations reached over the link carrying the routing packet. This is done as part of a technique known as *split horizon* which helps to reduce route convergence time when the topology changes [13]. In those cases where the next hop information is not already present, it is necessary only to add one extra bit to each of the (*destination, distance*) pairs in the routing packets. The bits identify which destinations are reached via the link on which the routing packet is being sent.

In the routing tables, another bit-map field, *leaves*, is added to each entry, identifying which of the *children* links are leaf links.

Now that we can identify leaves, it remains for us to detect whether or not members of a given group exist on those leaves.

To do this, we have the hosts periodically report their memberships. We can use the membership reporting algorithm described in Section 4, in which each report is locally multicast to the group that is being reported. Other members of the same group on the link overhear the report and suppress their own. Consequently, only one report per group present on the link is issued every reporting interval. There is no need for a very small reporting interval, because it is generally not important to quickly detect when all the members of a group on a link have departed from the group; it just means that packets addressed to that group may be delivered to the link for some time after all the members have left.

The routers then keep a list, for each incident link, of which groups are present on that link. If the lists are stored as hash tables, indexed by group address, the presence or absence of a group may be determined quickly, regardless of the number of groups present. The reverse path forwarding algorithm now becomes: if a multicast packet from S to G arrives from the *next-hop-address* for S, forward a copy out all child links for S except leaf links which have no members of G.

To summarize the costs of this algorithm, which we call *truncated reverse path broadcasting* or *TRPB*:

- It has a storage cost in each router of a few bits added to every routing table entry plus a group list for each of the router's links. The group lists should be sized to accommodate the maximum number of groups expected to be present on a single link (although temporary overflows of a group list may safely be handled by temporarily treating the corresponding link as a non-leaf, forwarding *all* multicast packets onto the link).
- It has a bandwidth cost on each link of one membership report per group present per reporting interval. The membership reports are very small, fixed-length packets, and the reporting interval may reasonably be on the order of minutes.
- The bandwidth cost of conveying next hop information in the routing packets is typically zero, either because the split horizon technique is used, or because an unused bit can be stolen from the existing (*destination, distance*) pairs to carry that information.

5.4 Reverse Path Multicasting (RPM)

As mentioned in the previous subsection, pruning the shortest-path broadcast trees by sending membership reports towards each multicast source results in an explosion of reporting traffic and router memory requirements. In a large internetwork, we would not expect every possible source to send multicast packets to every existing group, so the great expense of pruning every possible multicast tree would be wasted. We would prefer, then, to prune only those multicast trees that are actually in use.

Our final variation on the reverse path forwarding strategy provides *on-demand pruning* of shortest-path multicast trees, as follows. When a source *first* sends a multicast packet to a group, it is delivered along the shortest path broadcast tree to all links except non-member leaves, according to the TRPB algorithm. When the packet reaches a router for whom all of the child links are leaves and none of them have members of the destination group, a *non*-membership report (NMR) for that (*source, group*) pair is generated and sent back to the router that is one hop towards the source. If the one-hop-back router receives NMRs from all of its child routers (that is, all routers on its child links that use those links to reach the source of the multicast), and if its child links also have no members, it in turn sends an NMR back to its predecessor. In this way, information about the absence of members propagates back up the tree along all branches that do not lead to members. Subsequent multicast packets from the same source to the same group are blocked from travelling down the unnecessary branches by the NMRs sitting in intermediate routers.

A non-membership report includes an *age* field, initialized by the router that generates the report, and counted up by the router that receives the report. When the age of an NMR reaches a threshold, T_{maxage}, it is discarded. The NMRs generated at the leaves start with age zero; NMRs generated by intermediate routers, as a consequence of receiving NMRs from routers nearer the leaves, start with the maximum age of all of the subordinate NMRs. Thus, any path that is pruned by an NMR will rejoin the multicast tree after a period of T_{maxage}. If, at that time, there is still traffic from the same source to the same group, the next multicast packet will trigger the generation of a new NMR, assuming there is still no member on that path.

When a member of a new group on a particular link appears, it is desirable that that link immediately be included in the trees of any sources that are actively sending to that group. This is done by having routers remember which NMRs they have sent and, if necessary, send out cancellation messages to undo the effect of the NMRs.

If an NMR is lost in transit, a subtree will remain in the multicast tree unnecessarily, but that will last only until the next multicast packet stimulates generation of another NMR. Loss of a cancellation message is more serious, because a new path will fail to join the tree when it should, and group members on that path will fail to receive multicast packets from that tree for a period of up to up to T_{maxage}. If we require that cancellation messages be positively acknowledged by their receivers, we can afford to have a very long T_{maxage}, which reduces the amount of multicast traffic down unnecessary branches.

This algorithm, which we call *reverse path multicasting* or *RPM*, has the same costs as the TRPB algorithm, plus the costs of transmitting, storing, and processing NMRs and cancellation messages. Those extra costs depend greatly on such factors as the number and locations of multicast sources and of group members, the multicast traffic distributions, the frequency of membership changes, and the internetwork topology. In the worst case, the number of NMRs that a router must store is on the order of the number of multicast sources active within a T_{maxage} period, times the average number of groups they each send to in that period, times the number of adjacent routers. There are a couple of factors that can alleviate these storage requirements:

- All hosts attached to the same link may be treated as a single source of multicasts, as long as a router is able to identify the source link from the source addresses of datagrams, as is the case, for example, with DoD IP addresses [24].
- Multicast datagrams sent with a small time-to-live may expire before reaching many routers, thus avoiding the generation of NMRs in those routers.

We believe that many applications of internetwork multicasting will be able to use TTL scope control effectively, either because they require communication with only a nearby subset of a large group (e.g., when looking for a nearby name server), or

because all group members are known to be close to the senders (e.g., when a parallel computation is distributed across computers at a single site). If that is so, and the cost of memory keeps falling, storage space for NMRs should not be a limiting factor in typical distance-vector routing environments (fewer than a hundred links). Bandwidth can also be expended to recover memory, by reducing T_{maxage}. However, experience with real multicast traffic in real internetworks will be needed before recommendations can be made as to router memory sizes, timeout values, or even whether the greater "precision" of the RPM algorithm is worth the extra complexity and overhead, as compared to the simpler TRPB algorithm.

One issue that has not yet been mentioned in this discussion of reverse path forwarding schemes is the effect of topology changes. As explained in [6], reverse path forwarding can cause packets to be duplicated or lost if routing tables change while the packets are in transit. Since we require only datagram reliability, occasional packet loss or duplication is acceptable; hosts are assumed to provide their own end-to-end recovery mechanisms to the degree they require them. Implementations of the RPM algorithm, however, must be careful to take into account any topology changes that might modify the pruned multicast trees. For example, when a router gains a new child link or a new child router, relative to a given multicast source, it must send out cancellation messages for any outstanding NMRs it has for that source, to ensure that the new link or router is included in future multicast transmissions from that source.

6 Link-State Multicast Routing

The third major routing style to be considered is that of link-state routing, also known as "New Arpanet" or "Shortest-Path-First" routing [21]. As well as being used in the Arpanet, the link-state algorithm has been proposed by ANSI as an ISO standard for intra-domain routing [18].

Under the link-state routing algorithm, every router monitors the state of each of its incident links (e.g., up/down status, possibly traffic load). Whenever the state of a link changes, the routers attached to that link broadcast the new state to every other router in the internetwork. The broadcast is accomplished by a special-purpose, high-priority flooding protocol that ensures that every router quickly learns of the new state. Consequently, every router receives information about all links and all routers, from which they can each determine the complete topology of the internetwork. Given the complete topology, each router independently computes the shortest-path spanning tree rooted at itself, using Dijkstra's algorithm [1]. From this tree, it determines the shortest path from itself to any destination, to be used when forwarding packets.

It is straightforward to extend the link-state routing algorithm to support shortest-path multicast routing. Simply have routers include as part of the "state" of a link, a list of groups that have members on that link. Whenever a new group appears, or an old group disappears, from a link, the routers attached to that link flood the new state to all other routers. Given full knowledge of which groups have members on which links, any router can compute the shortest-path multicast tree from any source to any group, using Dijkstra's algorithm. If the router doing the computation falls within the computed tree, it can determine which links it must use to forward copies of multicast packets from the given source to the given group.

To enable routers to monitor group membership on a link, we again use the technique, introduced in Section 4, of having hosts periodically issue membership reports. Each membership report is transmitted as a local multicast to the group being reported, so that any other members of the same group on the same link can overhear the report and suppress their own. Routers monitoring a link detect the departure of a group by noting when the membership reports for that group stop arriving. This technique generates, on each link, one packet per group present per reporting interval.

It is preferable for only one of the routers attached to a link to monitor the membership of that link, thereby reducing the number of routers that can flood membership information about the link. In the link-state routing architecture proposed in [18], this job would fall to the "LAN Designated Router", which already performs the task of monitoring the presence of individual hosts.

As pointed out in Section 5, there is potentially a separate shortest-path multicast tree from every sender to every group, so it would be very expensive in space and processing time for every router to compute and store all possible multicast trees. Instead, we borrow from Section 5.4 the idea of only building trees on demand. Each router keeps a cache of multicast routing records of the form:

(*source*, *subtree*, (*group*, *link-ttls*),
(*group*, *link-ttls*), ...)

Source is the address of a multicast source. *Subtree* is a list of all descendent links of this router, in the shortest-path spanning tree rooted at *source*. *Group* is a multicast group address. *Link-ttls* is a vector of time-to-live values, one for each incident link, specifying the minimum TTL required to reach the nearest descendent member of the group via that link; a special TTL value for *infinity* identifies links that do not lead to any descendent members.

When a router receives a multicast packet, it looks up the source of the packet in its multicast routing cache. If it finds a record, it looks for the destination group in the (*group*, *link-ttls*) fields. If the group is found, the router forwards the packet out all links for which the minimum required TTL in *link-ttls* is less than or equal to the TTL in the packet header.

If the source record is found, but the destination group is not in the record, the router must compute the outgoing links and corresponding TTLs. To do this, it scans through the links in *subtree*, looking for links that have members of the destination group, and computing the minimum TTLs required to reach any member links found. The new *group* and *link-ttls* are added to the record and used in the forwarding decision.

Finally, if a record is not found for the source of an incoming multicast packet, the complete shortest-path spanning tree for that source must be computed. From the tree, the subtree of descendents of the router can be identified. The *source* and *subtree* are then installed as a new record in the multicast routing cache. The *link-ttls* for the destination group are also computed as part of computing the full tree, added to the record, and used in the forwarding decision. (A router for whom memory is scarcer than processing power might choose not to store the *subtrees* in the multicast routing cache, and simply recompute the full tree whenever a new group for a particular source is encountered.)

Cache records need not be timed out. When the cache is full, old records may be discarded on a least-recently-used basis. Whenever the topology changes, all cache records are discarded. Whenever a new group appears, or an old group disappears, on a

link, all (*group, link-ttls*) fields identifying that group are removed from the cache.

Like the RPM algorithm described in the previous section, the costs of this algorithm are very dependent on the internetwork multicast traffic patterns. Assuming that there are generally fewer groups present on a single LAN than there are individual hosts, the bandwidth required for group link state packets should be no more than that required for "End System" link state packets, in the proposed ANSI routing scheme [18]. The same is true of the memory needed in the routers to hold the link membership information. The major costs of the algorithm are in the memory required to store the multicast routing cache records and the processing requirements of computing the multicast trees. Assuming that most multicast packets are required to traverse a small percentage of the routers in the internetwork, this algorithm requires less storage space than the RPM algorithm, because storage is consumed only in those routers that must be traversed, rather than in those that must *not* be traversed.

One possible drawback of this algorithm is the additional delay that may be imposed on the first multicast packet transmitted from a given source—at each hop, the routers must compute the full tree for that source before they can forward the packet. The complexity of the tree computation is of the order of the number of the links in the internetwork (for sparsely-connected interworks); decomposing a large internetwork into routing subdomains, as proposed in the ANSI scheme, is an effective way of controlling the number of links within any domain.

7 Hierarchical Multicast Routing

All of the algorithms discussed so far are appropriate for a single routing domain, in which all routers are running the same algorithm. Large internetworks often span *multiple* routing domains. For example, a LAN that is part of a distance-vector routing environment may actually be an extended LAN containing spanning-tree bridges, or one "link" in a link-state routing environment may actually be an entire internetwork using distance-vector routing. Such hierarchical composition—treating one routing domain as a single link in a higher-level routing domain—has many advantages. It reduces the amount of topology information any one router has to maintain, thereby improving scaleability [19]; it accommodates different technologies for which different routing strategies are appropriate; and it allows different organizations to choose the routing style that best fits their needs, while still interoperating with other organizations.

All of the multicast routing algorithms we have proposed may be used to route multicast packets between "links" that happen to be entire routing subdomains, provided that those subdomains meet our requirements for links. Section 3 identifies the two generic types of links assumed by the multicast algorithms: point-to-point links and multi-access links. A subdomain may be treated as a point-to-point link if it used only for pairwise communication between two routers or between a router and a single host. Alternatively, a subdomain may be treated as a multi-access link if it satisfies the following property:

- If any host or superdomain router attached to the subdomain sends a multicast packet addressed to group G into the subdomain, it is delivered (with high probability) to all hosts that are members of G *plus* all superdomain routers attached to the subdomain, subject to the packet's time-to-live (TTL).

In addition, if the superdomain multicast routing protocol does *not* use the approach of delivering every multicast packet to every link, it must be possible for the superdomain routers to monitor the group membership of hosts attached to the subdomain. This may be done using the membership reporting protocol described in the previous sections, or via some other, subdomain-specific, method.

The above property is required of a subdomain when using our algorithms as superdomain multicast routing protocols. Looking at it from the other side, when using our algorithms as *subdomain* multicast routing protocols beneath an arbitrary superdomain protocol, we find that we do not quite satisfy the above property for subdomains. We must extend our algorithms to include all superdomain routers as members of every group, so that they may receive all multicast packets sent within the subdomain. This is accomplished simply by defining within the subdomain a special "wild-card" group that all superdomain routers may join; the changes to each algorithm to support wild-card groups are straightforward.

8 Related Work

A variety of algorithms for multicast routing in store-and-forward networks are described by Wall [26], with emphasis on algorithms for constructing a single spanning tree that provides low average delay, thereby striking a balance between opposing goals of low delay and low network cost.

Frank, Wittie and Bernstein [10] provide a good survey of multicast routing techniques that can be used in internetworks, rating each according to such factors as delay, bandwidth, and scaleability.

Sincoskie and Cotton [25] propose a multicast routing algorithm for link-layer bridges which supports a type of group in which all senders must also be members of the group. Such groups are acceptable for some applications, such as computer conferencing, but are not well suited to the common client/server type of communication where the (client) senders are generally not members of the (server) group and should not receive packets sent to the group.

9 Conclusions

We have proposed a number of algorithms for routing multicast datagrams in internetworks and extended LANs. The goal of each algorithm is to provide a multicast service that is as similar as possible to LAN multicasting, so that applications that currently benefit from LAN multicasting may be moved to a multiple-network environment with little or no change. In particular, we have concentrated on *low delay* multicasting, in order to minimize the effect of going from the LAN environment to a store-and-forward environment.

Different multicast routing algorithms were developed as extensions to three different styles of *unicast* routing: the single-spanning-tree routing of extended LAN bridges, and the distance-vector and link-state routing commonly used in internetworks. These different routing styles lead to significantly different multicast routing strategies, each exploiting the particular protocols and data structures already present.

For most of the algorithms, the additional bandwidth, memory and processing requirements are not much greater than those of

the underlying unicast routing algorithm. In the case of distance-vector routing, we presented a range of multicast routing algorithms based on Dalal and Metcalfe's reverse path forwarding scheme, providing increasing "precision" of delivery (flooding, broadcasting, truncated broadcasting and multicasting) at a cost of increasing amounts of routing overhead.

In spite of the wide difference in multicast routing strategies, all except the flooding and broadcasting variants impose the same requirement on hosts: a simple membership reporting protocol which takes good advantage of multicasting to eliminate redundant reports. Thus, the same host protocol implementation may be used without change in a variety of different multicast routing environments.

Finally, we have shown how different routing domains using these or other multicast routing protocols may be combined to extend multicasting across a large, hierarchical internetwork.

We have implemented the host membership reporting protocol in the 4.3BSD UNIX kernel as the first step in an experiment with internetwork multicasting of DoD IP datagrams [7], and implementations of both the reverse path multicast (RPM) and the link-state multicast routing algorithms are under way. From these implementations, we plan to derive detailed specifications for each of the multicast routing algorithms, and to start gathering measurements of multicast traffic patterns and their effect on routing overhead, for a variety of distributed multicast applications, such as computer conferencing, name binding, and network management. Once we get a better idea of multicast "workloads", we hope to provide stronger criteria for choosing among the various multicast routing algorithms.

Acknowledgements

The idea of applying the membership reporting strategy to the extended LAN environment was suggested by David Cheriton. David Waitzman pointed out how the link-state multicasting algorithm could take into account TTL scope control. They, as well as Bruce Hitson, Cary Gray, and an anonymous reviewer, provided many other helpful comments on an earlier draft of this paper. The DARPA Internet task force on end-to-end protocols, chaired by Bob Braden, has encouraged and contributed to the development of these ideas.

This work was sponsored in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, and by Digital Equipment Corporation.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, Mass., 1983. Dijkstra's algorithm.

[2] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.

[3] D. R. Boggs. *Internet Broadcasting*. PhD thesis, Electrical Engineering Dept., Stanford University, January 1982. Also Tech. Rep. CSL-83-3, Xerox PARC, Palo Alto, Calif.

[4] D. R. Boggs, J. F. Shoch, E. A. Taft, and R. M. Metcalfe. PUP: an internetwork architecture. *IEEE Transactions on Communications*, COM-28(4):612–624, April 1980.

[5] D. R. Cheriton and W. Zwaenepoel. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.

[6] Y. K. Dalal and R. M. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, December 1978.

[7] S. E. Deering. *Host Extensions for IP Multicasting*. RFC 1054, SRI Network Information Center, May 1988.

[8] Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. The Ethernet: a local area network; data link layer and physical layer specifications, version 1.0. *Computer Communications Review*, 11(3):20–66, September 1980.

[9] L. R. Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, N.J., 1962.

[10] A. J. Frank, L. D. Wittie, and A. J. Bernstein. Multicast communication on network computers. *IEEE Software*, 2(3):49–61, May 1985.

[11] J. Hart. Extending the IEEE 802.1 MAC bridge standard to remote bridges. *IEEE Network*, 2(1):10–25, January 1988.

[12] W. R. Hawe, M. F. Kempf, and A. J. Kirby. The extended local area network architecture and LANBridge 100. *Digital Technical Journal*, (3):54–72, September 1986.

[13] C. Hedrick. *Routing Information Protocol*. RFC (in preparation), SRI Network Information Center, November 1987.

[14] R. Hinden and A. Sheltzer. *The DARPA Internet Gateway*. RFC 823, SRI Network Information Center, September 1982.

[15] IEEE Computer Society. Standards for local area networks: logical link control. ANSI/IEEE Standard 802.2-1985 (ISO/DIS 8802/2), 1985.

[16] International Business Machines Corporation. *Technical Reference PC Network*. document 6322916.

[17] International Organization for Standardization (ISO). *Draft International Standard 8473, Protocol for Providing the Connectionless-Mode Network Service*. March 1986.

[18] Secretariat USA (ANSI) ISO TC97 SC6. *Intermediate System to Intermediate System Intra-Domain Routing Exchange Protocol*. November 1987.

[19] L. Kleinrock and F. Kamoun. Hierarchical routing for large networks; performance evaluation and optimization. *Computer Networks*, 1:155–174, 1977.

[20] S. J. Leffler, R. S. Fabry, W. N. Joy, P. Lapsley, S. Miller, and C. Torek. An advanced 4.3BSD interprocess communication tutorial. In *Unix Programmer Supplementary Documents, Part 2*, University of California, Berkeley, Ca., April 1986.

[21] J. M. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, COM-28(5):711–719, May 1980.

[22] J. M. McQuillan and D. C. Walden. The ARPANET design decisions. *Computer Networks*, 1, August 1977.

[23] R. Perlman. An algorithm for distributed computation of a spanning tree in an extended LAN. In *Proc. 9th Data Communications Symposium*, pages 44–53, ACM/IEEE, September 1985.

[24] J. Postel. *Internet Protocol*. RFC 791, SRI Network Information Center, September 1981.

[25] W. D. Sincoskie and C. J. Cotton. Extended bridge algorithms for large networks. *IEEE Network*, 2(1):16–24, January 1988.

[26] D. W. Wall. *Mechanisms for Broadcast and Selective Broadcast*. PhD thesis, Electrical Engineering Dept., Stanford University, June 1980. Also Tech. Rep. 190, Computer Systems Lab., Stanford.

[27] Xerox Corporation. *Internet Transport Protocols*. XSIS 028112, Xerox, Stamford, Connecticut, December 1981.