Multicast Routing in Datagram Internetworks and Extended LANs

STEPHEN E. DEERING and DAVID R. CHERITON
Stanford University

Multicasting, the transmission of a packet to a *group* of hosts, is an important service for improving the efficiency and robustness of distributed systems and applications. Although multicast capability is available and widely used in local area networks, when those LANs are interconnected by store-and-forward routers, the multicast service is usually not offered across the resulting *internetwork.* To address this limitation, we specify extensions to two common internetwork routing algorithms—distance-vector routing and link-state routing—to support low-delay datagram multicasting beyond a single LAN. We also describe 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 discuss how the use of multicast scope control and hierarchical multicast routing allows the multicast service to scale up to large internetworks.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*network topology, packet networks, store and forward networks*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*protocol architecture*; C.2.5 [**Computer-Communication Networks**]: Local Networks

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Broadcast, datagram, distance-vector routing, hierarchical routing, internetwork, link-state routing, local area network, multicast, reverse path forwarding, routing

1. INTRODUCTION

Multicasting in a datagram or connectionless network is the transmission of a packet to a subset of the hosts in the network. An efficient multicast facility provides packet delivery to groups of hosts at a lower network and host overhead than broadcasting to all hosts or unicasting to each host in a group.

Multicast capability is being recognized as an important facility for networks and internetworks because of its growing use in distributed systems, such as the V System [6] and the Andrew distributed computing environment [26], in popular protocol suites such as Sun's broadcast RPC service [28] and IBM's NetBIOS

This work was sponsored in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, and by Digital Equipment Corporation. An earlier version of this paper appeared in *Proceedings of the ACM SIGCOMM '88 Symposium* (Stanford, Calif., Aug. 1988).
Authors' address: Computer Science Department, Stanford University, Stanford, CA 94305.

[17], and in distributed applications such as conferencing [25]. A multicast service offers two important benefits to network applications:

Efficient multidestination delivery. When an application must send the same information to more than one destination, multicasting is more efficient than *unicasting* separate copies to each destination—it reduces the transmission overhead on the sender and, depending on how it is implemented, it can reduce the overhead on the network and the time taken for all destinations to receive the information. Examples of applications that can take advantage of multidestination delivery are:

—updating all copies of a replicated file or database;
—sending voice, video, or data packets to all members of a computer-mediated conference; and
—disseminating intermediate results to all participants in a distributed computation.

Robust unknown destination delivery. If a set of destinations can be identified by a single *group address* (rather than a list of individual addresses), such a group address can be used to reach destinations whose individual addresses are unknown to the sender, or whose addresses may change over time. Sometimes called *logical addressing* or *location-independent addressing*, this use of multicast serves as a simple, robust alternative to configuration files, directory servers, or other binding mechanisms. Examples of applications that can take advantage of logical addressing are:

—querying a distributed database or file store, where the particular location of the desired data is unknown;
—locating an instance of a particular network service, such as name service or time service; and
—reporting sensor readings to a self-selected, changeable set of monitoring stations.

Multicast is well supported by local area networks such as Ethernet [9] that provide efficient broadcast delivery and a large space of multicast addresses.[1] However, multicast as a general facility across large extended LANs and internetworks raises a number of concerns. First, networks and internetworks that are not based on a broadcast facility require extensions to their routing to provide efficient multidestination delivery. For example, DoD IP Gateways [15] and ISO Intermediate Systems [18] currently support unicast routing only. Second, multicasting in extended LANs and internetworks imposes additional routing and traffic costs that may limit the scalability of the multicast service or of the (inter)networks that support multicast. For example, link-layer bridges, such as

[1] Some applications have (unfortunately) implemented multicasting by using the network's *broadcast* addressing 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 [16] standards, which provide multicast addresses that can be recognized and filtered by host interface hardware.

the DEC LANBridge 100 [13] and the Vitalink TransLAN [12], can connect numerous Ethernets to form one logical Ethernet, confining unicast traffic to local segments when the source and destination are nearby. However, these systems flood multicasts to every attached segment. Finally, certain properties of LAN multicast, such as low delay, (almost) simultaneous delivery, and delivery with high probability to all hosts, may be infeasible to support in an internetwork, rendering multicast of significantly less value. These concerns have prompted some to conclude that multicast is a local network facility only, whose use is to be avoided in order to allow for scalability of systems and applications. We disagree.

In this paper, we present extensions to two common routing algorithms used by network-layer routers—distance-vector routing and link-state routing—to provide efficient routing for multicast across datagram-based internetworks. We also describe modifications to link-layer bridge routing to improve the efficiency of multicasting in large extended LANs. We analyze the costs and benefits of multicast support using these techniques for common (multicast) applications and conclude that internetwork multicast is feasible, highly desirable, and actually improves the scalability of applications.

In the next section of this paper, we describe our multicast model in terms of the service provided, the expected use, and the assumed underlying communication facilities. Then follow three sections describing specific multicast extensions to the single-spanning-tree, distance-vector, and link-state routing algorithms. In Section 6, we discuss how the multicast routing schemes may be combined hierarchically to support multicasting in very large internetworks. In Section 7 we briefly discuss other work in the same area, and in the concluding section we summarize our results.

2. MULTICASTING MODEL

Our approach to multicast is based on a choice of service interface, assumed use, and assumed underlying facilities. This section describes these aspects.

2.1 Host Groups: The Service Interface

Each multicast address identifies a *host group* [5], the group of hosts that should receive a packet sent to that address. A multicast packet is delivered with "best efforts" datagram reliability to all members of the host group identified by the destination multicast address. The sender need not know the membership of the group and need not itself be a member of the group. We refer to such a group as *open*, in contrast to a *closed* group in which only members are allowed to send to the group.

The service interface imposes no restriction on the number or location of hosts in a group (although such restrictions may be imposed by higher-layer protocols or by administrative controls in individual routers or bridges). 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. A host may belong to more than one group at a time.

The sender can specify the *scope* of a multicast packet to limit its delivery to nearby group members only. For example, a query to the directory server group may be directed to a local subset of the group.

This model has several advantages. First, the addressing matches the single (and generally fixed) addressing format of unicast packets and the style of multicast addressing used on broadcast-based networks, such as the Ethernet. Besides convenience, this compatibility allows for efficient implementation on those networks. Second, since the sender need not know the constituents of the group, group addresses may be used for logical addressing. Third, the dynamic nature of the groups recognizes the dynamic nature of hosts and applications, due to crashes as well as changing participation. Finally, this model is efficiently implemented in extended LANs and internetworks, as we describe in subsequent sections.

2.2 Types of Multicast Groups

Multicast groups can be roughly divided into three categories, according to the distribution of their members across an internetwork:

Pervasive groups have members on all or almost all links or subnetworks in the internetwork. Examples of pervasive groups are widespread directory services, or *netnews* distribution groups. Such groups tend to be very long-lived, having "well-known" multicast addresses.

Sparse groups have members on only a small number of (possibly widely-separated) links. Examples of sparse groups are real-time, computer-supported conferences, or replicated databases. Such groups may be long-lived or transient.

Local groups have members on only a single link or on a set of links within a single administrative subdomain of the internetwork. Examples of local groups are distributed parallel applications or games executing at a single site. Such groups tend to be transient, existing only as long as required for the execution of a single program.

For a pervasive group, an efficient *broadcast* facility that conveys a multicast packet to all links via a shortest-path delivery tree can offer a significantly lower delivery cost and lower delivery delay than sending individual unicast packets to each member of the group. The multicast routing algorithms presented in this paper all support efficient broadcast delivery to all links, via a shortest-path tree.

For some pervasive groups, such as directory server groups, it is essential that a multicast packet *not* be delivered to all members of the group, but rather to nearby members only. This is supported by the scope control facility that allows a sender to limit the propagation distance of a multicast packet. Not only is this crucial for the scalability of pervasive services such as directory service, but it also allows hosts to avoid being bombarded by replies from a populous group.[2]

[2] An interesting and useful application of 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 directory server: a host multicasts a directory query, starting with a scope that reaches only its immediate neighborhood, and incrementing the scope on each retransmission to reach further and further afield, until it receives a reply.

Some sparse groups can also benefit from scope control, for example, a sparse group of identical (replicated) servers, whereas others require delivery to all members, as in the case of a conference group. The multicast routing algorithms described in this paper provide selective delivery to sparse groups via sender and group-specific multicast trees, in order to avoid the overhead of transmitting a multicast packet across unnecessary links, and to avoid the redundancy and reduce the delay of multiple unicasts.

The distinction between pervasive and sparse groups is based on the relative costs of broadcast delivery, as compared to selective multicast delivery—broadcast delivery incurs the cost of delivering to all links, whether or not group members are present on those links (expensive for sparse groups), whereas selective delivery incurs the cost of control traffic by which the routers learn where the group members are located (expensive for pervasive groups). The precise costs depend on the particular topology of the internetwork, the multicast traffic distributions, and the specific routing algorithms in use. Thus, the boundary between pervasive and sparse varies from internetwork to internetwork. We expect most groups to be sparse and we have concentrated on developing efficient selective delivery algorithms. Specific groups in a given internetwork may be identified as pervasive, however, based on observed or predicted membership distribution, and assigned distinguished group addresses so that the routers may recognize and apply broadcast delivery algorithms to multicast packets for those groups.

Local groups are efficiently handled by a selective multicast delivery service. If all the senders to a local group are in the same locality, scope control combined with broadcast delivery can also provide low-overhead multicasting.

Regardless of the type of group, it is important that the delivery characteristics of multicast packets be the same as unicast packets. In particular, a multicast packet should be delivered to each member of its destination group (within the specified scope) with probability and delay very close to that of a unicast packet sent to that same member. This property gives higher-layer protocols a basis to handle packet loss by retransmission. Delivery probability comparable to unicast precludes "systematic errors" in delivery so a small number of repeated transmissions results in delivery to all group members within the specified scope, unless a member is disconnected or has failed. Low delay is an important property for a number of multicast applications, such as distributed conferencing, parallel computing, and resource location. Besides the basic delivery delay, the multicast facility should minimize the delay between the time a host joins a group and the time it can start receiving packets addressed to that group, called the *join latency*. (On a LAN, this time is usually just the time required to update a local address filter.) Low join latency is important to minimize packet loss after joining and to facilitate synchronizing with unicast delivery.

2.3 Base Communication Environment

Communication is provided by 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 that 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.

Internetwork packets include a header field that limits the number of hops that a packet may travel, usually called the *time-to-live* field, which can provide multicast scope control.[3] Fine-grain scope control, as used for expanding-ring searching, is accomplished by using very small time-to-live values; in particular, we assume that a sufficiently small value can be chosen to limit a multicast to a single hop. Scope control at the granularity of administrative boundaries is supported by having the internetwork routers that interconnect different administrative domains enforce a certain minimum time-to-live on multicast packets leaving a domain, so that packets sent with a time-to-live less than the minimum can be guaranteed not to leave the domain. For example, assume that the boundary routers for a domain whose diameter is administratively constrained to be less than 32 hops refuse to forward a multicast packet with a remaining time-to-live of less than 32 hops. Then any multicast packet sent from within the domain with an initial time-to-live of 31 hops can reach any group member within the domain, but is prevented from leaving the domain.

3. SINGLE-SPANNING-TREE MULTICAST ROUTING

Link-layer bridges [12, 13] 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

[3] The *time-to-live* field in an internetwork packet is usually expressed in units of time, such as seconds, and serves to limit the maximum time that the packet may live in the internetwork, in order to satisfy the assumptions of typical transport protocols. However, the field is always decremented by at least one unit by every router, so that it also behaves as a hop limit, which protects against forwarding loops caused by errors or transient inconsistencies in routing tables. Thus, it serves as an upper bound on both time and hops—the packet may be discarded before its time has expired (too many hops) or before its hop limit is exceeded (too much time).

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 segments necessary to reach their target hosts. In small bridged LANs, bridge and link resources are usually abundant; however, in large extended LANs that have a lot of multicast traffic directed to sparse or local groups, it can be of great benefit not to send multicast packets everywhere.

3.1 Description of the Algorithm

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 *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 transmissions of multicast packets destined to G are forwarded only in the direction of that membership. For example, a multicast packet to G originating on LAN b will traverse d and a, but not c. A multicast to G originating on a will not be forwarded at all.

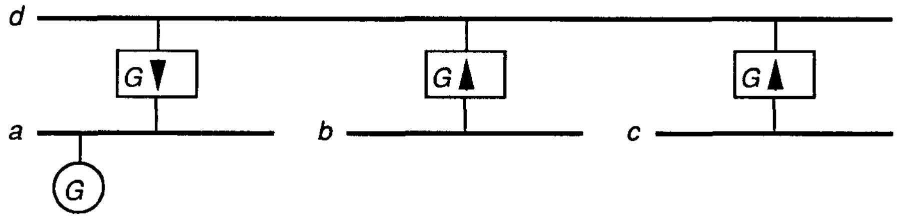


Fig. 1. Bridged LAN with one group member.

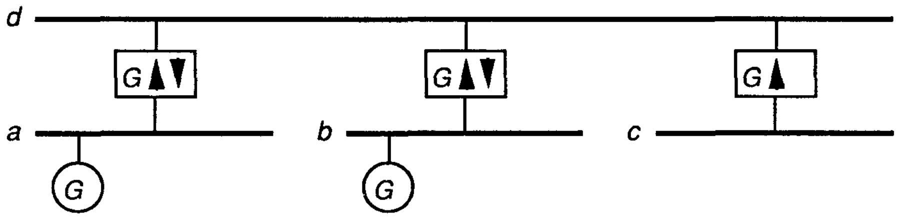


Fig. 2. Bridged LAN with two group members.

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 strategy 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:[4]

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

The basic multicast routing algorithm then becomes:

If a packet arrives with a multicast source address, record the arrival branch and an associated *age* of zero in a table entry for that multicast address.

Periodically increment the *age* fields of all multicast table entries. If an *age* field reaches an *expiry threshold*, T_{expire}, delete the associated *outgoing-branch* from the table entry, and if no *outgoing-branches* remain, delete the entire entry.

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

If a packet arrives with a multicast destination address, forward a copy of the packet out every *outgoing-branch* recorded in the table entry for that multicast address, excluding the arrival branch. If there is no table entry for that multicast address, discard the packet.

3.2 Costs of the Algorithm

The main cost of this multicasting algorithm, as compared to the broadcast algorithm used by current bridges, is the overhead of the periodic membership reports. This overhead can be made very small by choosing a large *reporting interval*, T_{report}. The following observations justify the use of a large T_{report}, say on the order of several minutes:

—The expiry threshold, T_{expire}, for bridge table entries must be several times larger than 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 T_{report} has passed one or two times. This fails to meet the goal of low join latency, stated in Section 2.2. It can be overcome 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.

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. 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. It requires the following simple addition to the basic algorithm described above, to be performed before looking up the destination table entry:

If a packet arrives with the same multicast address as both source and destination, replace its destination address with the *all-bridges* multicast address.

With this addition, the costs of the multicast routing algorithm are as follows, assuming each host belongs to G_{host} groups, each segment contains members of $G_{segment}$ groups, and there are S segments and G_{total} groups in total:

—Each host sends *or* receives G_{host}/T_{report} membership reports per second.

—Each *leaf* segment, and each bridge interface to a leaf segment, carries $G_{segment}/T_{report}$ membership reports per second.

—Each *non-leaf* segment, and each bridge interface to a non-leaf segment, carries the reporting traffic from all segments, that is $S \times G_{segment}/T_{report}$ membership reports per second.

—Each bridge requires storage for G_{total} multicast table entries.

For example, in an extended LAN with 10 segments, 5 groups per host, 20 groups per segment, 50 groups total, and a reporting interval of 200 seconds, the host cost would be one packet every 40 seconds, the leaf cost would be one packet every 10 seconds, the non-leaf cost would be one packet every second, and the bridge memory cost would be 50 table entries of length 20–30 bytes each. Such costs are insignificant compared to the available bandwidth and bridge capacity in current extended LAN installations.

In return for this low overhead, this algorithm saves the bandwidth consumed by transmitting multicast packets on unnecessary segments, thus greatly improving the scalability of the extended LAN. In particular, for an extended LAN that becomes saturated with multicast traffic for sparse groups, this routing algorithm keeps that traffic off most leaf segments. This allows the congestion to be relieved by installing higher-capacity backbone segments only, rather than having to upgrade the (presumably) more numerous leaf segments to which most hosts would be connected.

3.3 Other Issues

The link-layer headers on which bridge routing depends do not lend themselves to multicast scope control as described in Section 2, having no hop-count or time-to-live field. A simple alternative is to set aside a block of multicast addresses for local (single segment) use only and have bridges refuse to forward any packets sent to such addresses. This provides two levels of multicast scope: local and global. Hosts that belong to groups for which scope-controlled access is useful, such as a replicated directory server group, join both a local group and a global group; a client host sends a query to the local multicast address first, and if that fails, retransmits to the global multicast address. If local addresses are distinguished from global addresses by a single bit in the address, then hosts can easily determine the local address corresponding to a given global address, and bridges can easily recognize those multicast packets that must not be forwarded.

The algorithms described for forwarding multicast packets and learning multicast addresses are simple extensions of the unicast forwarding and learning algorithms, which ought to make them easy to add to existing bridge implementations. However, some current bridges employ custom hardware to provide very fast handling of unicast traffic, and such hardware may not directly accommodate the extensions specified here. Assuming those bridges also contain adequate memory and general processing capability, multicast packets could be handled by trapping to software; with current LAN multicast applications, multicast traffic is a small percentage of total traffic, so such an approach should have acceptable performance. We see no reason why future bridges cannot be built to handle high-speed multicast forwarding for new applications such as real-time video transmission.

The bridge multicast routing algorithm as described requires that hosts be modified to issue membership reports for those groups to which they belong. This compromises the transparency property that is one of the attractive 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.

4. DISTANCE-VECTOR MULTICAST ROUTING

The distance-vector routing algorithm, also known as the Ford-Fulkerson [10] or Bellman-Ford [2] algorithm, has been used for many years in many networks and internetworks. For example, the original Arpanet routing protocol [21] 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 [31], some DARPA Internet core gateways [15], numerous UNIX® systems running Berkeley's *routed* internetwork routing process [14], and many proprietary routers.

Routers that use the distance-vector algorithm maintain a routing table that 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

® UNIX is a trademark of AT&T Bell Laboratories.

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 the 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 that 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 [30] 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 objective of low-delay multicasting, and to provide reasonable semantics for 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 the following sections, we use that observation as the basis for a number of refinements to Dalal and Metcalfe's *reverse path forwarding* broadcast algorithm [7], which is well-suited to the distance-vector routing environment. The refinements provide increasing "precision" of multicast delivery by increasingly pruning the shortest (reverse) path broadcast trees.

4.1 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. Thus, reverse path forwarding is easily implemented and, as long as path lengths are symmetric (or nearly symmetric), effective at providing shortest-path (or nearly shortest-path) broadcasting in distance-vector routing environments.

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 [7] 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 as 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 end, 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 by Dalal and Metcalfe in [7] and also in the following section. To distinguish the basic flooding form of reverse path forwarding from later refinements, we refer to it as *reverse path flooding* or RPF.

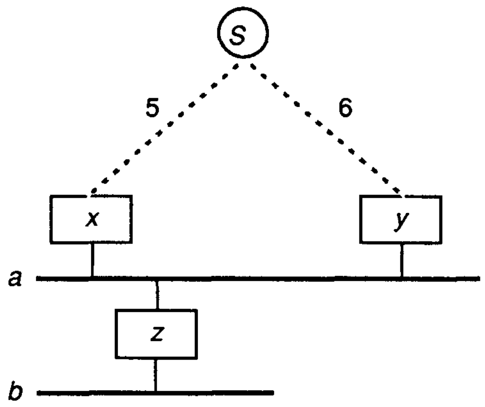
4.2 Reverse Path Broadcasting (RPB)

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.

Dalal and Metcalfe [7] propose a method for discovering child links that 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 that 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. For the link to which S itself is attached, S is considered the parent. On all other links, the parent is the router 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

Fig. 3. Reverse path forwarding example.



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 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 any 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 become the parent of LAN a.

This 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*.

With the duplicates eliminated, the decision whether or not to forward a broadcast packet can be based on the *next-hop-link* for the packet's source, rather than its *next-hop-address*. This is advantageous because it eliminates the need to examine the link-layer source address when making the forwarding decision in the internetwork layer. The forwarding algorithm becomes:

> If a broadcast packet with source address S arrives from the *next-hop-link* for S, forward a copy of the packet on all *child* links for S.

The only significant cost of this algorithm is the extra storage consumed by the *children* bit-map in each routing table entry. Each bit-map requires one bit for each incident link, typically a very small number.[5] The benefit of this algorithm is the saving of link bandwidth and of host and router processing time achieved by eliminating duplicate broadcast packets.

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.

4.3 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 in small extended LANs. 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 needed, especially if a significant amount of multicast traffic is directed at sparse or local groups.

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 section, 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

[5] The parent selection technique assumes that a router maintains a copy of the most recent routing packet received from each of its neighboring routers, in order to compare its own distances and address with those of its neighbors. However, some implementations of the distance-vector routing algorithm do not maintain copies of recent routing packets—each incoming routing packet is used to update the routing table and then immediately discarded. An "incremental" parent selection technique can be used to avoid having to save recent routing packets, at the cost of duplicate broadcast packets during periods immediately following a router failure or recovery, and the cost of storing in each routing table entry an array of *parent-addresses*, rather than an array of *child* bits, one for each incident link. See [29] for a specification of such a technique. However, the memory savings achieved by not retaining recent routing packets does not seem to warrant the extra complexity of the incremental approach and the cost of the extra duplicate broadcasts, given today's low cost of memory.

bandwidth is a critical resource. The next section 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 section, 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 [14]. 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 3, 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 forwarding algorithm now becomes:

> If a multicast packet with source address S and destination group G arrives from the *next-hop-link* for S, forward a copy of the packet on all *child* links for S, except leaf links that have no members of G.

The costs of this algorithm, which we call *truncated reverse path broadcasting*, or TRPB, can be summarized as follows:

—It has a storage cost in each router of the *children* and *leaves* bit-maps added to every routing table entry plus a group list for each of the router's links.

Each bit-map requires one bit for each incident link. 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.

4.4 Reverse Path Multicasting (RPM)

As mentioned in the previous section, 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 nonmember 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 (i.e., 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 traveling down the unnecessary branches by the NMRs sitting in intermediate routers.

A nonmembership 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. Cancellation messages must be positively acknowledged by the receiving router to guarantee that the loss of a cancellation message does not cause a new group member to be excluded from its associated multicast trees (for a period of up to T_{maxage}).

Each NMR is also positively acknowledged, so that the sender of the NMR can be sure that the previous hop router has received it. Then, if packets with a source address and group address covered by an outstanding NMR are received, they can simply be dropped without triggering a new NMR. A router may continue to receive multicast packets for which an NMR is outstanding because members of the destination group may be present on the incoming link or as descendants of other routers attached to the incoming link.

Any change of topology that might modify the pruned multicast trees must also be taken into account.[6] In particular, 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.

This final refinement of the reverse path forwarding scheme, 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. If the router is able to identify the originating link from the source address of a packet, as is the case, for example, with DoD IP [24], this storage requirement can be reduced by treating all source hosts attached to the same link as a single source.

The storage costs can also be greatly reduced by the use of multicast scope control, as discussed in Section 2.2. Multicast packets whose scope prevents them from reaching many routers avoid the generation of NMRs in the unreached routers. Scope control can and should be used for any application in which multicasts need to reach only nearby members of a host group, such as a replicated directory server group, or in which all members of the destination group are known to be near the sender, such as a computation distributed across multiple computers at a single site.

[6] Topology changes were not explicitly addressed by the preceding algorithms because they depend only on the existing routing tables, which are updated in response to topology changes by the normal operation of the distance-vector routing algorithm. As explained in [7], 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.

In a small distance-vector-based internetwork, the extra complexity and extra costs of the RPM algorithm over the TRPB algorithm may not be justified, especially if the internetwork consists of a single "backbone" link connecting a large number of leaf links, or if there is little traffic for sparse groups. On the other hand, there seems to be a movement away from distance-vector routing algorithms and toward link-state algorithms for large or richly-connected internetworks, due to the poor convergence properties of distance-vector algorithms during topology changes [20]. If the TRPB algorithm proves to be inadequate in limiting congestion due to multicasting in a particular internetwork, it may well be preferable to switch to link-state routing rather than implementing RPM. The next section describes a simple and efficient link-state multicasting algorithm.

5. 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 [20]. 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], and is being considered as a standard "open" routing protocol for the DARPA/NSF Internet [22].

5.1 Description of the Algorithm

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, reliable, high-priority flooding protocol that ensures that every router quickly learns 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 consider as part of the "state" of a link the set of groups that have members on that link. Whenever a new group appears, or an old group disappears, on 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 3, 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 by ANSI [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 4, 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 4.4 the idea of computing trees only on demand. Each router keeps a cache of multicast routing records of the form

(*source*, *group*, *min-hops*)

Source is the address of a multicast source. *Group* is a multicast group address. *Min-hops* is a vector of distances, one for each incident link, specifying the minimum number of hops required to reach the nearest descendant member of the group via that link; a special hop value of *infinity* identifies links that do not lead to any descendant members.

When a router receives a multicast packet from source S to group G, it looks in its cache for a record with the corresponding *source* and *group*. If no match is found, the router uses Dijkstra's algorithm to compute the shortest-path spanning tree rooted at S, to discover which of the router's incident links lead to members of G in the subtree rooted at the router itself. For each incident link that leads to one or more group members, the distance in hops to the nearest member is computed and recorded in a new cache entry for S and G.

Once a cache entry is obtained, the router forwards the packet out all links for which the *min-hop* is less than or equal to remaining hops in the packet's *time-to-live* field.

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 the link membership of a group changes, all cache records corresponding to that group are discarded.

5.2 Costs of the Algorithm

The costs of this link-state multicast algorithm may be divided into two categories: costs related to forwarding multicast packets and costs related to disseminating and maintaining group membership information.

In support of multicast forwarding, the algorithm incurs the storage cost of the multicast routing cache and the spanning-tree computation cost of cache misses, each of which may be traded off in favor of the other. For a sparse network such as most existing internetworks, the cost of Dijkstra's algorithm for computing a shortest-path tree is on the order of the number of links in the network, assuming the distance metric for each link is defined over a small finite field [18]. For a typical internetwork containing up to 200 links, the worst-case tree computation time on a contemporary (approximately 10 MIPS) microprocessor is under 5 milliseconds. Unlike the tree computation that is performed on a topology change, the multicast tree computations need not hold up the forwarding of other traffic—they may be performed in parallel in a multiprocessor-based router or performed as a lower-priority task in a multi-threaded uniprocessor implementation.

The frequency of cache misses depends on the distribution of multicast traffic, the distribution of group membership changes, the frequency of topology changes, and, of course, the cache size. The traffic and membership distributions are impossible to predict in the absence of any current internetwork multicast applications. Once such applications are developed and deployed, it will be important to measure their characteristics and experiment with different cache sizes.

The dissemination and maintenance of group membership information incurs the cost in each router of storing the list of groups present on each link, and the cost of updating all routers whenever the group membership changes on any link. Assuming that there are generally fewer groups present on a single LAN than there are individual hosts, the storage required in the routers is less than that needed to record all host addresses, as is necessary in the proposed ANSI routing scheme [18].

Although there will generally be fewer groups represented than hosts present on a single LAN, group membership is expected to change more frequently than host presence. Whenever a group appears on a link (i.e., the first member host on that link joins the group), or a group disappears from a link (i.e., the last member host on that link leaves the group), an update must be sent from that link's designated router to all other routers. The reliable flooding protocol used to disseminate updates requires every router to send a packet (either an update or an acknowledgment) on each of its incident links, except those links that have no other routers attached. As mentioned above, the dynamics of group membership changes are impossible to predict at this time, although a number of observations may be made:

—Most memberships will be long-lived. For well-known service groups, each member host will likely join the group when it starts up and remain a member until it fails or is restarted. Some transient groups, such as those supporting human conferencing, will also be relatively long-lived. Such groups will not be the source of rapid join/leave updates.

—Volatile groups, that is, groups with short-lived memberships, will be sparsely distributed. An example is a group set up for a short, distributed computation. Such groups will generate rapid join/leave updates from only a small number of links at a time.

—The membership update generated when a group disappears from a link may safely be delayed and piggybacked on other updates. The consequence of delaying a report of group disappearance is the possible forwarding of packets onto links where there are no longer any members, until the update is eventually sent.

—If necessary, updates generated when a group first appears on a link may be rate-limited, at the cost of greater join latency. For example, a router may be prohibited from generating group appearance updates more frequently than once every five seconds, with each update listing all groups that appeared in the past five seconds. This scheme is already used to limit the rate of normal link-state updates caused by a link that is oscillating between being the "up" and "down" states.

As discussed in Section 2.2, pervasive groups may be efficiently handled by broadcast delivery, instead of selective multicast delivery. If pervasive groups can be recognized by their multicast addresses, routers can refrain from sending join/leave updates for those groups, thus eliminating the associated membership dissemination and maintenance costs, and on reception of packets addressed to such groups, can simply treat them as if they were all addressed to a single group with members on every link.

One drawback of the link-state multicast algorithm is the additional delay that may be imposed on the first multicast packet transmitted from a given source to a given group—at each hop, the routers must compute the tree for that source before they can forward the packet. As discussed above, the tree computation time grows with the number of the links in the internetwork; decomposing a large internetwork into hierarchical routing subdomains, as proposed in the ANSI scheme, is an effective way of controlling the number of links within any domain.

6. HIERARCHICAL MULTICAST ROUTING

Most of the multicast routing algorithms discussed so far have costs that grow with the number of links present in the internetwork or extended LAN, as do the underlying unicast routing algorithms on which the multicast algorithms are built. These routing overheads, both unicast and multicast, limit the scalability of a single routing domain. The common solution for unicast routing is to decompose a very large routing domain into multiple subdomains, organized hierarchically, such that one subdomain is treated as a single link in a higher-level domain [19]. The same strategy can be applied to multicast routing domains to scale the multicast service up to large internetworks.

All of the multicast routing algorithms we have described 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 2.3 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 is 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 datagram reliability) to all hosts that are members of *G plus* all superdomain routers attached to the subdomain, subject to the packet's scope control.

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 attached 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.

In the important and common case of "leaf subdomains," that is, those subdomains that are not required to carry transit traffic between other subdomains, it is possible to relax the requirement for all attached superdomain routers to receive all internally-originated multicasts—a single superdomain router may be given responsibility for forwarding multicasts with sufficiently large scope to the rest of the internetwork. The superdomain routers are generally able to detect that a subdomain is a leaf, and in that case, can dynamically elect a single router to join the wild-card group and perform multicast forwarding duties.

7. RELATED WORK

A variety of algorithms for multicast routing in store-and-forward networks are described by Wall [30], 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 [11] provide a good survey of multicast routing techniques that can be used in internetworks, rating each according to such factors as delay, bandwidth, and scalability.

Sincoskie and Cotton [27] 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.

8. CONCLUSIONS

We have described a number of algorithms for routing multicast datagrams in internetworks and extended LANs. Different multicast routing algorithms were developed for each of the following popular 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. For the more sophisticated algorithms, such as RPM and the link-state multicast algorithm, the amount of routing overhead is very sensitive to the multicast traffic distribution and to the dynamics of group membership, with which we currently have no experience. Although we anticipate certain applications for internetwork multicasting, such as resource location or conferencing, there is no way to predict what the mix of applications will be. Further evaluation of these algorithms awaits the deployment and measurement of real multicast applications in real internetworks.

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 discussed how the use of multicast scope control and hierarchical multicast routing allows the multicast service to scale up to large internetworks.

At the time of writing,[7] a version of the membership reporting protocol for use with the DoD IP protocol has been adopted as an optional Internet Standard [8], and has been implemented for several variants of the Berkeley Unix operating system. The TRPB algorithm described in Section 4.3 has also been implemented for the same systems, and is currently supporting internetwork multicast experiments at a small number of research sites. A working group of the Internet Engineering Task Force has taken on the task of extending the OSPF routing protocol [22] to incorporate our link-state multicast routing algorithm and hierarchical multicast routing scheme, with the goal of making an internetwork multicast service much more widely available in the DARPA/NSF Internet.

ACKNOWLEDGMENTS

We would like to thank David Waitzman for pointing out how the link-state multicast algorithm could take into account scope control and for the lessons learned from his implementation of the TRPB algorithm. We are grateful for the helpful comments of Bruce Hitson, Cary Gray, and the anonymous reviewers of earlier versions of this paper. The DARPA Internet task force on end-to-end

[7] January 1990.

protocols, chaired by Bob Braden, has encouraged and contributed to the development of these ideas.

REFERENCES

1. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *Data Structures and Algorithms.* Addison-Wesley, Reading, Mass., 1983.
2. BELLMAN, R. E. *Dynamic Programming.* Princeton University Press, Princeton, N.J., 1957.
3. BOGGS, D. R. Internet broadcasting. Ph.D. thesis, Electrical Engineering Dept., Stanford Univ., Jan. 1982. Also Tech. Rep. CSL-83-3, Xerox PARC, Palo Alto, Calif.
4. BOGGS, D. R., SHOCH, J. F., TAFT, E. A., AND METCALFE, R. M. PUP: An internetwork architecture. *IEEE Trans. Commun. COM-28*, 4 (Apr. 1980), 612–624.
5. CHERITON, D. R., AND DEERING, S. E. Host groups: A multicast extension for datagram internetworks. In *Proceedings of the 9th Data Communications Symposium* (Sept. 1985), ACM/IEEE, New York, 1985, 172–179.
6. CHERITON, D. R., AND ZWAENEPOEL, W. Distributed process groups in the V kernel. *ACM Trans. Comput. Syst. 3*, 2 (May 1985), 77–107.
7. DALAL, Y. K., AND METCALFE, R. M. Reverse path forwarding of broadcast packets. *Commun. ACM 21*, 12 (Dec. 1978), 1040–1048.
8. DEERING, S. E. Host extensions for IP multicasting. RFC 1112, SRI Network Information Center, Aug. 1988.
9. DIGITAL EQUIPMENT CORP., INTEL CORP., AND XEROX CORP. The Ethernet: A local area network; data link layer and physical layer specifications,. version 1.0. *Comput. Commun. Rev. 11*, 3 (Sept. 1980), 20–66.
10. FORD JR., L. R., AND FULKERSON, D. R. *Flows in Networks.* Princeton University Press, Princeton, N.J., 1962.
11. FRANK, A. J., WITTIE, L. D., AND BERNSTEIN, A. J. Multicast communication on network computers. *IEEE Softw. 2*, 3 (May 1985), 49–61.
12. HART, J. Extending the IEEE 802.1 MAC bridge standard to remote bridges. *IEEE Network 2*, 1 (Jan. 1988), 10–25.
13. HAWE, W. R., KEMPF, M. F., AND KIRBY, A. J. The extended local area network architecture and LANBridge 100. *Digital Tech. J.* 3 (Sept. 1986), 54–72.
14. HEDRICK, C. Routing information protocol. RFC 1058, SRI Network Information Center, June 1988.
15. HINDEN, R., AND SHELTZER, A. The DARPA internet gateway. RFC 823, SRI Network Information Center, Sept. 1982.
16. IEEE COMPUTER SOCIETY. Standards for local area networks: Logical link control. ANSI/IEEE Standard 802.2-1985 (ISO/DIS 8802/2), 1985.
17. INTERNATIONAL BUSINESS MACHINES CORP. *Technical Reference PC Network.* Doc. 6322916.
18. ISO TC97 SC6, SECRETARIAT USA (ANSI). *Intermediate System to Intermediate System Intra-Domain Routing Exchange Protocol*, Oct. 1989.
19. KLEINROCK, L., AND KAMOUN, F. Hierarchical routing for large networks; performance evaluation and optimization. *Comput. Netw. 1* (1977), 155–174.
20. MCQUILLAN, J. M., RICHER, I., AND ROSEN, E. C. The new routing algorithm for the ARPANET. *IEEE Trans. Commun. COM-28*, 5 (May 1980), 711–719.
21. MCQUILLAN, J. M., AND WALDEN, D. C. The ARPANET design decisions. *Comput. Netw. 1* (Aug. 1977).
22. MOY, J. The OSPF specification. RFC 1131, SRI Network Information Center, Oct. 1989.
23. PERLMAN, R. An algorithm for distributed computation of a spanning tree in an extended LAN. In *Proceedings of the 9th Data Communications Symposium* (Sept. 1985), ACM/IEEE, New York, 1985, 44–53.
24. POSTEL, J. Internet protocol. RFC 791, SRI Network Information Center, Sept. 1981.
25. SARIN, S. K. Interactive on-line conferences. Tech. Rep. MIT/LCS/TR-330, MIT Laboratory for Computer Science, Dec. 1984.
26. SATYANARAYANAN, M., AND SIEGAL, E. H. MultiRPC: A parallel remote procedure call mechanism. Tech. Rep. CMU-CS-86-139, Carnegie-Mellon Univ., Aug. 1986.

27. SINCOSKIE, W. D., AND COTTON, C. J. Extended bridge algorithms for large networks. *IEEE Network 2*, 1 (Jan. 1988), 16–24.
28. SUN MICROSYSTEMS. *Remote Procedure Call Reference Manual.* Mountain View, California, Oct. 1984.
29. WAITZMAN, D., PARTRIDGE, C., AND DEERING, S. Distance vector multicast routing protocol. RFC 1075, SRI Network Information Center, Nov. 1988.
30. WALL, D. W. Mechanisms for broadcast and selective broadcast. Ph.D. thesis, Electrical Engineering Dept., Stanford Univ., June 1980. Also Tech. Rep. 190, Computer Systems Lab., Stanford.
31. XEROX CORP. Internet transport protocols. XSIS 028112, Xerox, Stamford, Conn., Dec. 1981.

Received October 1988; revised January 1989; accepted January 1990