Centralized Multicast

S. Keshav and S. Paul†*
*: Department of Computer Science, Cornell University, skeshav@cs.cornell.edu
†: Bell Laboratories, Holmdel, NJ 07733, sanjoy@research.bell-labs.com

Abstract

Most current schemes for multicast routing assume that multicast routers participate both in forwarding multicast packets and in control algorithms for routing, resource reservation, and group management. By separating data and control flow, and by centralizing control in distinct control elements, we have designed a simple and scalable approach to IP multicast that we call *Centralized Multicast*. We present the details of our approach, a proof of its correctness, analysis of its performance, and a discussion of its advantages over current schemes.

1 Introduction

The key concept in IP multicast is that of a multicast group. Such a group has two defining properties. First, packets sent by *any* member of a group are received by *all* other members. Second, members may join or leave a group without reference to other members. The multicast group concept is powerful because it provides a primitive to arbitrarily group widely dispersed processes. It can provide an efficient infrastructure for advanced services such as group communication, fault-tolerant computing, information dissemination, and hierarchical web caching.

In the literature, 'IP multicast' refers loosely to a set of four related problems (a)*routing*: finding a path from a sender to the other members in the multicast group, (b)*reservation*: reserving resources along such a path (c) *reliability*: assuring that all packets from a sender reach all receivers, despite packet loss and corruption, and (d) *flow control*: regulating the rate at which a sender sends packets so that intermediate links and receivers are not overloaded. These problems are more or less orthogonal, in that we can pick and choose solutions to each problem independently. In this paper, we focus only on the first two problems. Approaches to the remaining problems can be found in references [FJM+95][YGS95][HSC95][WMK95][H96][LLG96][PSLB97][M97][VR98][BKTN98]. In the rest of the paper, unless otherwise specified, we use the term 'IP multicast' to refer to the routing problem.

To be of practical use, IP multicast must be efficient, scale well, and be incremetally deployable. By *efficiency*, we mean that setting up and maintaining the group should require only a few control messages. Moreover, multicast packets should follow an optimal route from a source to the receivers. By *scalability* we mean that the number of control messages and the amount of state in network elements should grow at most linearly with the number of receivers in the group and the size of the network. *Incremental deployability* means that we should be able to add the multicast algorithm to the Internet without requiring a simultaneous change at all routers and endpoints.

Current algorithms for IP multicast fail one or more of these three criteria. For example, the flood and prune approach in the current MBONE [D88] is inefficient for sparse groups because packets are flooded to all routers, even those that are not interested in the group. We believe that this lack of efficiency has resulted in the relative paucity of multicast applications and multicast-capable routers in the Internet. Other proposals in the literature, such as CBT [B93, BFC 93] and PIM [DEF+ 94, DEF+ 96] also fail one of the three criteria (we discuss these failings in more detail in Section 7). In contrast, we propose a *centralized* approach to multicast that we show is efficient, scalable, and incrementally deployable.

Our approach to multicast routing derives from the following observation. Traditional multicast routing algorithms require routers to participate in both data movement and control algorithms. For instance, with the DVMRP approach [D88, D89], every multicast capable router must agree not only to forward multicast packets (i.e. those packets with Class D addresses that correspond to previously established multicast groups), but also participate in the DVMRP protocol. The CBT [B93, BFC93] and PIM [DEF+94, DEF+96] approaches have similar requirements. We believe that requiring routers to participate in both data forwarding and control algorithms results in adding substantial complexity to routers. Multicast forwarding is a hard enough problem; requiring routers to additionally support complex routing, flow, and reservation protocols has made the task all the more difficult. Moreover, decentralized control necessarily leads to well-known problems with distributed state such as the lack of global knowledge, the need for synchronization of state, and, frequently, the need to live with non-optimal solutions [L96].

Our approach, therefore, (a) decouples data movement and control and (b) centralizes control in special control elements. The control elements are arranged in a two-level hierarchy, corresponding to the control of multicast routing within and between Autonomous Systems (or do-

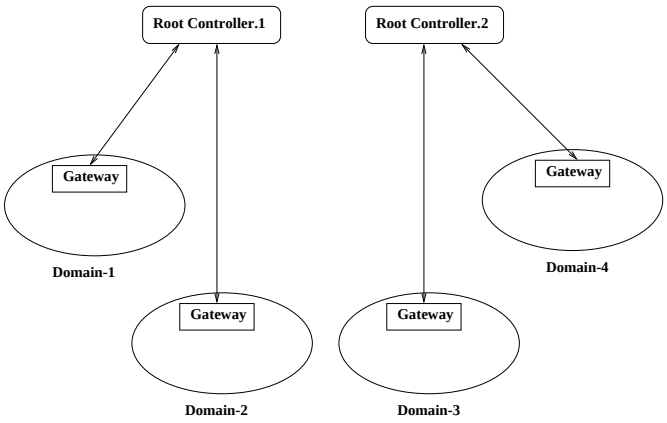


Figure 1: Gateways and root Controllers

mains). We call the control element in each domain a *gateway* and the control elements linking gateways *root controllers* (see Figure 1). Because of the key role of centralization of control in our approach, we call it *Centralized Multicast* or *CM*.

We note here that we are keenly aware of the two major problems with centralization, namely, overhead arising from central participation in all decision making, and the possibility of catastrophic failure from a single fault. We address these issues by using the standard techniques of creating a hierarchy (of root controllers and gateways), replication (of gateways), and loose synchronization (among root controllers), as described in more detail in Section 3. Section 7 has a discussion of the pros and cons of centralization in greater depth.

2 Assumptions

We precede a detailed description of CM by laying out the assumptions implicit in our work. We assume that the Internet consists of a number of Autonomous Systems or *domains*, where each domain is under the administrative control of a single entity. In practice, the Internet is somewhat more hierarchical, with one or two additional levels in the hierarchy. However, this additional complexity does not materially affect our work, since we can extend our algorithms to multiple hierarchical levels (though we do not do so in this paper).

Our second assumption is that each domain runs a link-state routing protocol, such as OSPF [Mo94a] or IS-IS [P92]. While this is currently almost universally true, in Section 5.6 we extend our scheme to deal with the domains that continue to run distance-vector protocols such as RIP [P92].

Our third and final assumption is that all domains are willing to trust a set of *root controllers* to establish and manage (IP-in-IP) tunnels between them. We choose the name 'root controller' to echo the 'root domain' in the Domain Name System. We hope that domains will be willing to trust the activities of a handful of multicast root controllers in the same way that current domains trust the names resolved by DNS roots.

3 Basic scheme

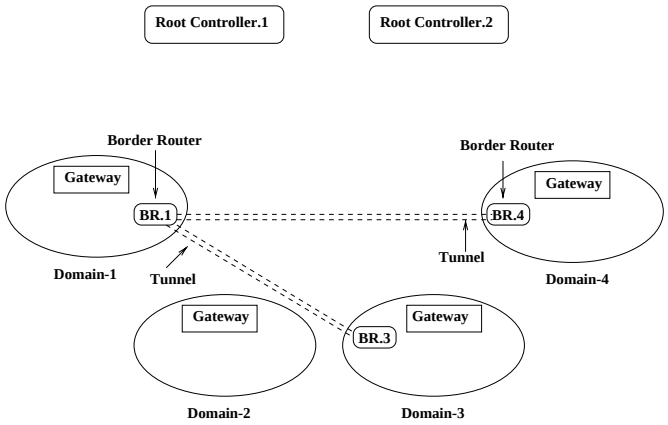


Figure 2: Domains, Root Controllers, Gateways, and Tunnels

In this section, we outline the basic operation of CM, describing only its role in determining multicast routes. We present its use in resource reservation in Section 4. To aid clarity, we will describe the scheme in three stages, discussing first the actions at a host (Section 3.1), then the actions at a gateway (Section 3.2), and finally the actions at a root-controller (Section 3.3). At each stage, we will progressively flesh out the scheme, reserving Section 5 for a detailed and complete explication.

3.1 Host actions

In CM, each domain is associated with a well-known *gateway*. Hosts that want to participate in a multicast group can determine the IP address of the gateway either through a well-known name (such as multicast-gateway.my_univ.edu or multicast-gateway.my_company.com) or by manually configuring individual hosts, as is already done for DNS servers. In either case, hosts know where to send a multicast join or leave request.

The basic action of a host is to join or leave a multicast group. Hosts may discover the Class D IP address of the group in a variety of ways. The address may be given to the host using a session directory (as in `sd` [J94]), or found in a centralized server, such as in a web server that lists currently known multicast groups. In any case, the host sends a JOIN request to the gateway with this address, and an additional flag if the host believes that it is the first member of the group (i.e., its creator). The gateway participates in the intra-domain routing protocol of the domain and knows about all other join and leave requests made in the domain (since it is the sole recipient of such requests). It therefore knows both the topology of the domain and the set of multicast groups in the domain. It uses this information to handle the join or leave request, as described next.

3.2 Gateway actions

3.2.1 Joining a locally known group

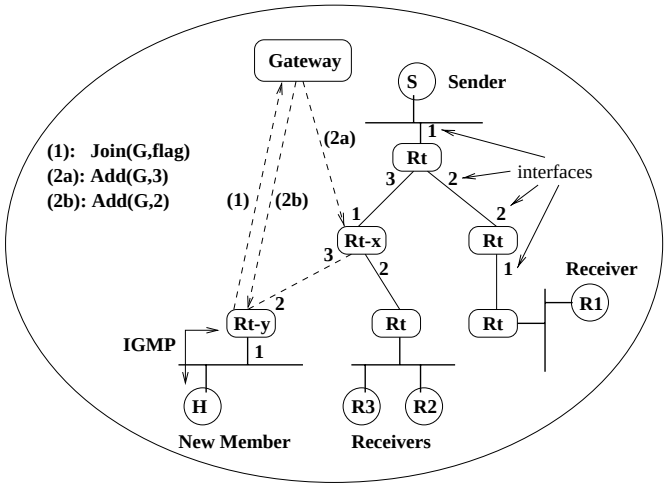
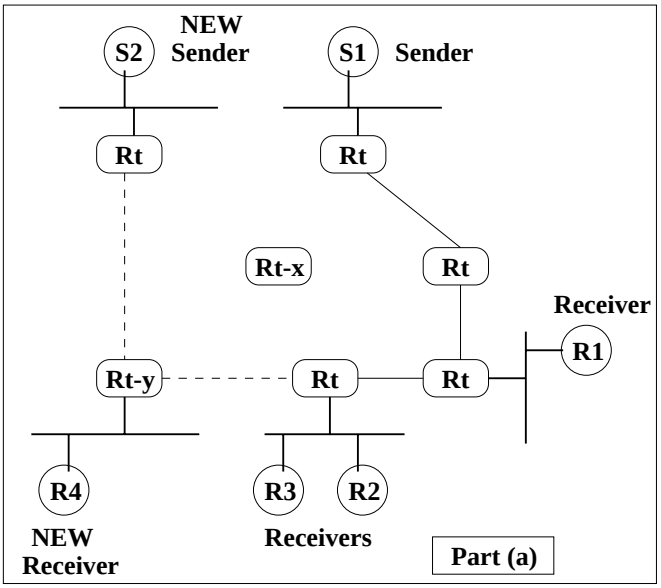


Figure 3: Joining a local group

The simplest case is when a host wants to join a multicast group that already has senders or receivers in its local domain (we call this a *local* join). In this case, the gateway first determines the router in the multicast tree that is nearest the host. It *grafts* the host on to the existing tree by sending ADD messages to the routers that lie on a path connecting the host to this router. In practice, ADD messages are not sent all way to the host, but only until its nearest IGMP querier. Hosts can then leave the group either by sending an explicit leave message to the gateway, or by not responding to an IGMP query.

An instance of this portion of the algorithm is shown in Figure 3. Here, the new member H informs the IGMP querier router (Rt-y) that it wants to join group G using IGMP [D89]. Rt-y sends a Join(G,Flag) message to the gateway where Flag=1 if the new member thinks it is the first member of the group. Otherwise, Flag=0. In this example, we assume the existence of group G (shown by the multicast tree connecting sender S and receivers R1 through R3). The gateway sends messages (2a) and (2b) to routers Rt-x and Rt-y respectively so that Rt-y is grafted onto the existing multicast tree.

Although we could use a variety of mechanisms to send an ADD message from a gateway to a router, there are three reasons why using SNMP to update the routing table in the router's MIB is by far the best choice. First, SNMP is nearly universally deployed. Therefore, a gateway can execute an ADD without requiring changes to routers already deployed in the field. This is critically important for widespread adoption of the scheme. Second, the SNMP router MIB allows a gateway to access a router's routing table with minimal processing at the router. This reduces the control overhead at the router. Finally, SNMP is a lightweight protocol that incurs little overhead both in terms of round trip times and in the number of packets. Consequently, the use of SNMP leverages its simplicity and universal deployment to relieve routers of the burden of participating in control algorithms for multicast routing. Thus, we believe that it is a crucial ingredient in making CM efficient, and, more important, incrementally deployable in the Internet.



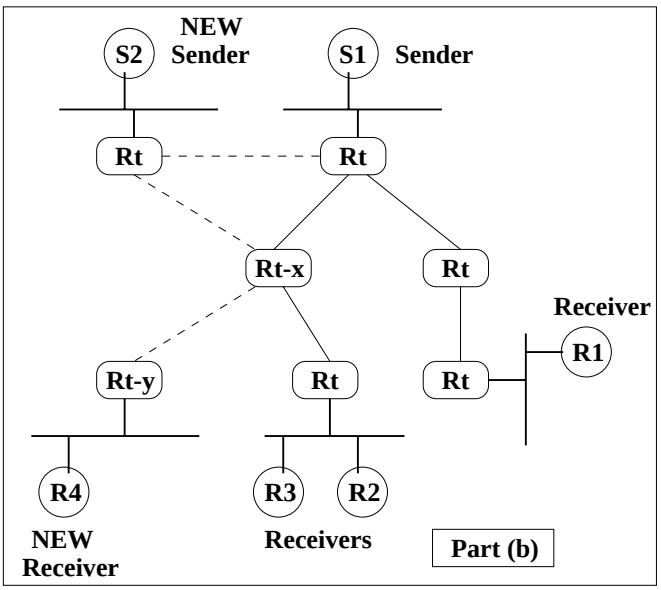


Figure 4: Shared tree vs. Shortest Path Tree

Note that the ADD message can add one of two possible entries to that router's forwarding table. First, the router may add an entry of the form {*S,G, incoming interface, outgoing interface list*} which tells the router that a packet from a particular source (S) belonging to a particular group (G) that arrives at the *incoming interface* must be forwarded to the *outgoing interface list* indicated in the entry. Clearly, the number of forwarding entries at each router increases linearly both with the number of

multicast groups and the number of potential senders in each group. This limits the scalability of the multicast protocol. An alternative approach, discussed in [AP96], uses the notion of an *io-set*. An io-set entry is of the form {*G, interface list*}, where the *interface list* consists of the interfaces that take part in multicast forwarding for group G. A packet with a group address G arriving on any interface x, such that x belongs to the interface list, is forwarded on all the *other* interfaces in the interface list. This construct saves space in the routing tables, at the expense of forcing all packets in the multicast group to traverse a shared multicast tree (as in the CBT approach [BFC93]). Thus, scalability is gained at the expense of a decrease in optimality (see Part (a) of Figure 4 for an example where the io-set approach forces the formation of a *shared* tree which saves forwarding table space, but creates a non-optimal multicast tree). Part (b) of Figure 4 shows the *shortest path* trees rooted at senders S1 and S2 for optimal performance. We reiterate the the CM scheme allows network operators to explicitly make the tradeoff between path optimality and size of the forwarding state.

A second optimization of the basic scheme is to make the forwarding state soft-state; in other words a router discards forwarding state after a certain time, unless this state is refreshed. Soft-state trades message overhead for robustness. Since we explicitly create state in the network, we advocate the use of soft state to allow the system to automatically recover from failures of routers and hosts. Achieving soft state requires decisions about which entity is responsible for deleting state and which entity is responsible for refreshing it. In general, the creator of state ought to refresh it, and the holder of state ought to destroy it. In our case this would mean that routers should periodically time out their multicast forwarding entries, and hosts should periodically refresh their join requests. We make a couple of slight modifications to this approach as discussed next. First, instead of requiring routers to time out state, we move this burden to the gateways. This is in keeping with out goal of requiring routers to do the least possible work, making them "big and dumb". Of course, this means that gateways must be extra-reliable, since a crash in a gateway will leave state permanently in routers. Because we have only one gateway in each domain, we require them to be fail-proof anyway. So, we do not think of this as an additional burden (we discuss techniques for making gateways fail-proof in Section 5.1). Second, we do not necessarily require a host to refresh gateway state. The IGMP protocol periodically probes hosts, and so we can defer the refresh operation to the IGMP-enabled router, which, on receiving an IGMP reply, forwards this reply to the gateway. These two modifications allow us to provide soft state with little additional burden to hosts and routers.

3.2.2 Creating a new group

We now describe the case where a host sends a join request to a group, along with an indication that it is the first member of the group. In the trivial case, the host also indicates that all members of the group are expected to be local, i.e., in the same domain. In this case, the gateway simply saves this information, waiting for additional join requests. A more complex situation arises when the host specifies the set of hosts that are allowed to join the group (so that the membership of the group is restricted). If so, the gateway performs admission control on behalf of the host. On receiving a join request, the ID of the joiner is compared with the list of allowed participants. If the joiner is not a member of the access-control list, the join request is denied. This is similar to the notion of "closed groups" in [AP96].

If the creator of a group specifies that the group may have non-local members, the gateway sends a message to the root controller with its ID, its domain ID, the Class D address of the multicast group, and an indication whether admission to the group is restricted. The root controller replies with either an acknowledgment, or an indication that the Class D address is already in use[1]. If the group is successfully created, then knowledge of this group is globally distributed, as described in Section 3.3.1.

We note that a useful option in this step is for a host to request creation of a group with an unspecified Class D address. This allows the gateway and the root controllers to return it a Class D address not already in use, which may be aggregatable in the backbone. With this flexibility, multicast addresses in the backbone can be aggregated hierarchically in the same way as unicast CIDR addresses. An example will clarify.

Example 1:

Suppose domain D already participates in multicast groups 225.1.3.2 and 225.1.3.6. By allocating a new group present in this domain a multicast address in the range 225.1.3.*, routing tables in the backbone need a single forwarding entry for the aggregated address. This is similar in concept to the MASC proposal [EHT97].

In this step, a gateway normally contacts the closest root-controller and waits indefinitely for an acknowledgement. We now introduce two optimizations to improve performance and robustness. First, a gateway is allowed to contact *any* root-controller with a join request. So, gateways that originate a burst of join requests can balance their load among a set of root-controllers. This improves overall performance. Second, when a gateway sends a root-controller a join message, it sets a timer, with a timeout value chosen from past measurements of acknowledgment times. If no reply is received before a

[1] This may happen if the multicast address allocation across all domains is not fully co-ordinated. However, if address allocation for all domains is done in a controlled manner, this situation will not arise.

timeout, the gateway automatically redirects its join request to one of the other root-controllers. In this way, gateways can transparently recover from the failure of a root-controller.

3.2.3 Joining a group not locally known

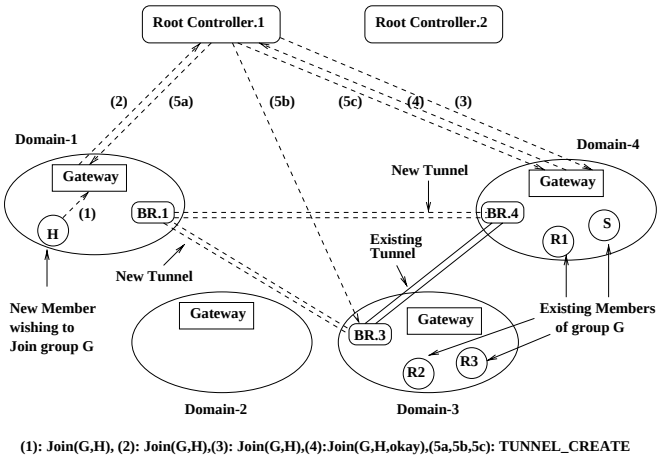


Figure 5: Non-local Join

We will use Figure 5 to illustrate the steps. Suppose a gateway receives a join request for a group that it is not already aware of (step (1) in the figure). By analogy with DNS, the gateway forwards this request to one of the root controllers, as discussed in more detail in Section 3.2.1 (step (2)). The selected root controller determines whether there are any admission restrictions for the group[2]. If so, the request is forwarded to the domain of the creator of the group (step (3)), where the gateway performs admission control. If the group is unrestricted, or if the admission request succeeds (step (4)), then the root controller sets up tunnels between all existing domains and the new domain (steps 5a through 5c). This is done by sending TUNNEL_CREATE requests to the appropriate gateways. The gateways create virtual interfaces and tunnels from their domain entry and exit points (see Figure 5) This does the job, but obviously is non-optimal. We discuss details of tunnel creation and optimal trees for of non-local groups in Section 3.3.2.

3.3 Root controller actions

By analogy with DNS, CM requires the operation of a small number (say, ten to twenty) of root controllers. The root controllers stay in loose synchronization by periodic exchange of state messages. The multiplicity of root controllers offers three benefits. First, it allows fault tolerance, so that if one root controller is down, another takes it place. Second, it distributes load, so that the overall stream of join requests is processed in parallel. Finally, it promotes efficiency, because, in the common case, a gateway needs to contact only the nearest root controller. The price to pay for these benefits is the overhead in synchronization and the complexity of fault recovery mechanisms. We describe in Section 3.3.3 and 5.7 how these overheads can be reduced. We now discuss the actions of the root controllers in response to actions of the gateways.

3.3.1 Group creation

When a gateway creates a new non-local group, it sends a message to the nearest root controller. (If this root controller is down, the gateway tries one of the alternates, as described earlier). The creation message includes the IP address of the gateway, the ID of the domain, the class D address of the multicast group, an indication whether access to the group is restricted, and an indication if the group creation is urgent[3]. The root controller stores this information in its local state. If the group is marked "urgent", then state information for the group is immediately disseminated to all other group controllers. As with router state, we expect root controller state to be soft state, to be refreshed periodically by the gateways. This allows robust recovery from root controller crashes at the expense of messaging overhead. Root controllers periodically exchange state with each other on a pre-established multicast group (similar to ALL-PIM-ROUTERS group in PIM [EFH+97] for example), thus allowing them to stay in loose synchrony. This state exchange is described in more detail in Section 3.3.3.

3.3.2 Joining a non-local group

In this subsection, we present more details of the approach outlined earlier in Section 3.2.3 When a root controller gets a join message from a gateway on behalf of a host in that domain, it checks to see if it has information about that group in its state table. There are three possibilities. First, the group exists and the root controller has state information for the group. In this case, the controller checks if the group is restricted. If so, the join request is passed on to the appropriate gateway for admission control. Second, the group may exist, but this information may not have been propagated to the root controller. This happens if the creator did not specify the group to be an "urgent" group. In this case, the join request is held until the next state synchronization point, at which time the join proceeds. Finally, no such group may exist. The root controller knows this if it receives no information about the group even after the synchronization phase. At this point, it informs the gateway, which forwards this information to the host. Thus, on receiving a join request, a root controller either processes the request immediately (if it has state for the group) or waits for state synchronization. After synchronization, the join is either processed or returned in error. Clearly, the delay in processing a join is being traded off here for the gain in

[2] All root-controller are made aware of all non-local groups using the algorithm described in Section 3.3.3.

[3] To preserve clarity, we did not mention the 'urgent' flag earlier.

efficiency by batching state updates. The trade off point can be tuned by choosing the state update interval appropriately. Note, however, that the use of the "urgent" group mechanism allows us to bypass the batching mechanism if necessary, at the cost of a loss in efficiency.

A join is declared successful if the group to be joined has unrestricted access or if the access control test is successful. A root controller that declares a join to be successful must add the joined domain to the existing multicast tree for the group. Before we explain how to carry this out, we first make a small detour to explain how tunnels are set up.

A tunnel allows packets leaving a domain to appear in another domain, uninterpreted by intermediate domains. Since CM separates data and control, tunnels need not originate or terminate at gateways. Instead, a gateway nominates some number of routers in that domain (typically the border routers) as tunnel entry and exit points (refer to Figure 5). To establish a cross-domain tunnel, a root controller sends a TUNNEL_CREATE message to corresponding gateways. This message gives the IDs of both domains, and the IP address of peer gateways. On receiving this message the two peer gateways exchange the IP address of the selected tunnel endpoints. On receiving this information, each gateway sends an appropriate message to its local tunnel endpoint requesting it to create a virtual interface for the tunnel, and add a multicast forwarding entry in the routing table for the virtual interface. In this way, tunnels are established in a hierarchical manner, without requiring the intervention of the root controller, except at a high level.

The tunnel establishment mechanism in hand, we now turn our attention to optimally joining a domain to an existing multicast group. We propose two complementary techniques to do so.

In the first solution, the joining domain is joined to the existing multicast group using TUNNEL_CREATE messages as described above. The key question is to pick the domains that ought to be linked with tunnels. The trivial solution is to establish a tunnel from every domain participating in the group to the joining domain. However, this solution requires $O(N^2)$ tunnels, where N is the number of domains. A better solution is to use a single backbone multicast tree, and use a single tunnel to graft a joining domain to the existing tree. To do so, root controllers periodically exchange the current multicast tree (at the inter-domain level) for each multicast group (details of this exchange are presented in Section 3.3.3). On receiving a join request, a root controller creates a tunnel from the domain to the nearest point of presence of the tree. If the tree is specified as a shared tree, the border routers participating in the backbone tunnel need only keep io-set state for that group (where the io-set refers to virtual interfaces instead of real ones). If the tree is a shortest path tree, then *all* border routers need to install additional (S,G) state corresponding to that domain. For scalability, we strongly advocate all backbone trees be shared. An example of the creation of the shared multicast backbone tree follows.

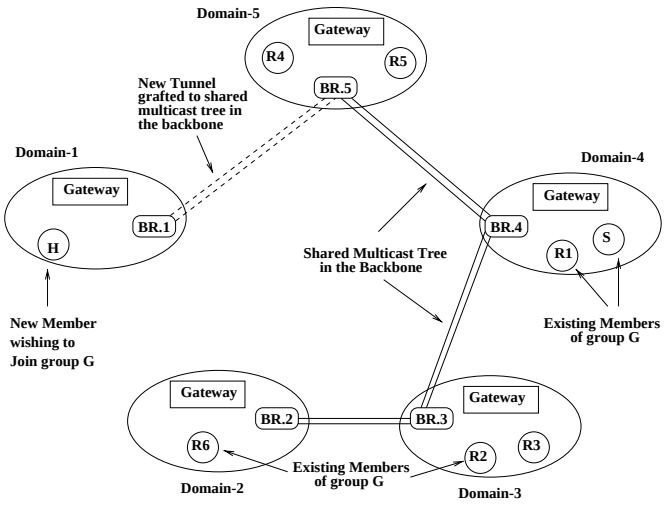


Figure 6: Grafting a new domain to the existing multicast tree in the backbone

Example 2

This example shows how a new domain can be grafted to an existing multicast tree in the backbone network. Refer to Figure 6 in which sender S and receivers R1 through R6 are subscribed to group G. A shared multicast tree is set up in the backbone network connecting domains 2 through 5. When host H from Domain-1 wants to join group G, a *single* tunnel is set up between Domain-1 and Domain-5 grafting Domain-1 onto the existing multicast tree. The other alternative would have required setting up tunnels between Domain-1 and *every* other domain.

The second technique to graft a domain leverages the current structure of the Internet (at least in the US), and the existence of the *routing arbiter*, an entity that resolves BGP polices at Network Access Points (NAPs) [GAV+98]. The arbiter receives BGP updates from all the ISPs incident at a NAP, and it comes up with a set of mutually consistent routing decisions, which it returns to each ISP's router. This allows routing exchanges at NAPs to proceed efficiently, while still allowing ISPs to specify routing policy directives. Given that the arbiter knows about backbone topology, it is natural extend the routing arbiter to also act as a root-controller. The root-controller/routing arbiter, therefore, can process join requests along with BGP updates. In this approach, to process a successful join, the root controller first contacts the gateway of the joining domain to find out which border router of that domain will be participating in the group. The root controller, by induction, already knows the border routers of all other domains. It simply installs multicast forwarding entries in all the border routers so that the group is established. Note this approach does

not require any tunnels. Moreover, since the routing arbiter knows about the routing policy of each domain, it could conceivably use this policy information when setting up multicast groups. However, the approach suffers from three major drawbacks. First, some domains may have border routers that are not represented at a NAP. Second, not all ISPs trust the routing arbiter with their internal topology, so the arbiter may not have full backbone information. Finally, ISPs may not want to have multicast forwarding entries set in their routers by third parties. In case any of these situations apply, we will need to fall back on the first solution, that requires the use of tunnels.

3.3.3 Periodic state exchange

Root controllers need to periodically exchange state to stay in loose synchrony. The synchronization period determines the delay in processing a join request: the more often state is exchanged, the faster the join succeeds. Thus, administrators can tune this exchange period to reflect the appropriate cost/benefit tradeoff. The state exchange protocol itself uses a hard-coded multicast group (similar to ALL-PIM-ROUTERS group in PIM) that links all root controllers. A root controller periodically multicasts its state on this group. Specifically, this state consists of the current set of tunnels, the current set of root controllers, and the current set of domains participating in each group (their IDs and the IP address of the corresponding gateways). This state is sufficient for a root controller to determine the best tunnel to join a new domain to an existing multicast group. Details of the state-exchange messages can be found in Section 5.4.

Since we expect this state to change only slowly over time, we propose that the actual state exchange should be of deltas in the state, with a periodic dump of the entire state to allow resynchronization (a full dump could also be requested by a root controller to force synchronization, for instance to allow it to recover from a crash). Also recall that the creation of an urgent multicast group results in the immediate dissemination of the associated state.

3.3.4 Reconfiguration

While each join request may be optimally satisfied, it is possible that a series of join requests may result in a non-optimal tree.

EXAMPLE 3

Refer to Figure 7 to see what might happen if a sequence of join requests are processed such that new domains are always grafted to the *nearest* entry point of the existing multicast tree. When Domain-6 is grafted to the existing tree, the maximum inter-domain delay in the tree increases to five units (assuming each hop adds a delay of one unit). However, a much better tree could have been

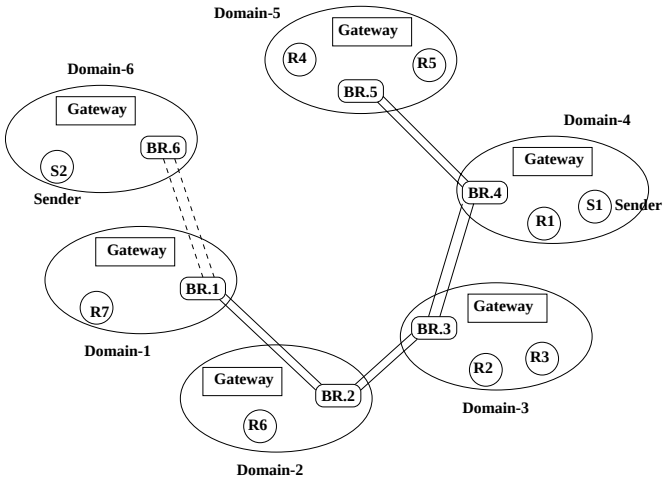


Figure 7: Suboptimal tree formation

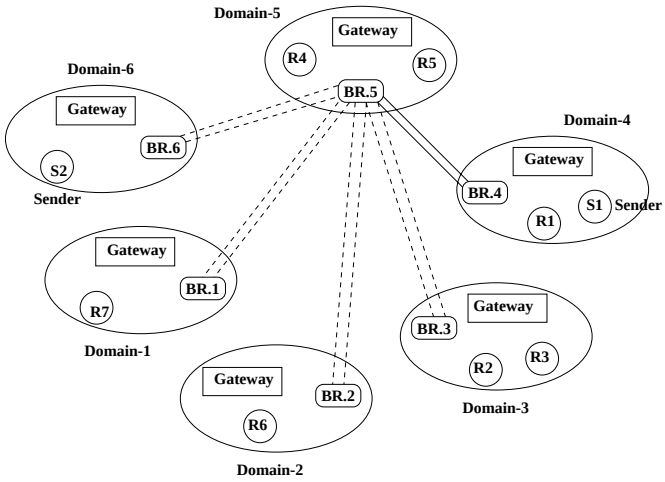


Figure 8: Optimal tree with the same set of members

computed as shown in Figure 8 in which the maximum inter-domain delay is two units.

To regain optimality, root controllers could periodically reconfigure the multicast tree. They can do this by using a make-then-break approach (similar to the tree switching algorithm described in [AP96]). That is, the new tree is created by adding new multicast routing entries (and tunnels) to the tree. Old entries and tunnels are deleted only after the new tree is fully established. In the transition period, routing loops may form, and some destinations may receive duplicate packets. Packets caught in loops will be discarded when their TTL drops to zero, and duplicates can be detected by upper layer protocols. Thus, this approach, when judiciously applied, allows the backbone to be asynchronously reconfigured with little extra effort. Similar benefits of periodic recomputation of the multicast tree are shown in [AP96].

4 Reservation

Although we view CM as primarily a scheme for multicast routing, it can be extended to provide per-stream

or per-aggregate quality of service by carrying out reservations and admission control. To do so, we will need the following assumptions. First, we assume that routers in the network provide some form of reservation, and consequently, some form of differential service (there is little point of making a reservation if unreserved traffic, which will continue to consist the bulk of the traffic, gets identical priority, bandwidth, and delay guarantees). Second, we assume that reservation state can be remotely accessed, for instance by means of an SNMP reservation MIB. Third, we assume that every reservation request, whether it be for a unicast or a multicast connection, is sent to a gateway. We will relax the last assumption later in this discussion.

With these assumptions in hand, let us consider the case where a host wants to join a *local* multicast group that already has made some reservations along the edges of a multicast tree. (If the existing group is a singleton, and consequently the tree is null, then this reduces to setting up a unicast reserved connection.) As with RSVP [ZDE+93, BZB+97], the creator of the group and each subsequent sender advertise a traffic spec for their traffic. Unlike RSVP, however, where this advertisement is sent to a multicast group, in CM, the traffic spec is sent to the gateway in the local domain. On receiving a join request, which may be accompanied by an RSVP-style filter, the gateway computes the nearest point of attchment of the host to the tree, and simultaneously determines the additional reservations that need to be made along the tree. This computation is identical in sprit to the distributed computation carried out in RSVP. If resources are insufficient, the receiver is notified along with an indication of the resources actually available, so that it may try again. If there are sufficient resources, the gateway installs reservation state (including appropriate filters) in the routers, for instance using the reservation MIB. When this operation successfully completes, the gateway notifies the receiver. If this is the first receiver to join, the sender is also notified, so that data transmission may commence. Routers periodically destroy state, and this state is refreshed by the gateway, as discussed in Section 3.2.1.

It is instructive to compare CM with RSVP. Our scheme reuses many design elements in RSVP: the traffic spec, the reservation spec, the style of filters, the use of soft-state, and receiver-initiated join. The major change is that the reservation and admission control decisions are centralized, not distributed. This has both pros and cons. On the con side, the scheme is not as robust as RSVP, in that if a gateway fails, no reservations can be made. We believe that it is possible to construct fail-safe gateways, and we discuss this in more detail in Section 5.1. Note that, just as in RSVP, CM can detect the failure of a router and reroute calls, because it participates in routing protocols (we discuss this in more detail in Section 5.7). Also, because both RSVP and CM use soft-state, they are equally good at recovering from host failures.

It may appear that CM is more expensive than RSVP, because it exchanges more messages than RSVP to accomplish the same reservation. Let us make the reasonable assumption that the main cost in the control path is in the number of messages processed. We show next that the difference in the number of messages exchanged by RSVP and CM is small. Suppose a host joins a tree along path R_1, R_2, ..., R_n. With RSVP, this requires $n-1$ messages to be exchanged, one from R_i to R_{i+1}. Using CM, the host sends a single message to the gateway, which then sendS n messages, one to each router. Thus, the cost, in terms of number of messages, increases only by two messages. In terms of delay, the CM approach may actually be faster, since all the n messages can be sent in parallel.

The main advantage of CM over RSVP is that it removes the burden of admission control and reservations from routers. They only need to carry out the reservation decisions made by the gateway. This makes them simpler. The second advantage is more subtle. In RSVP, RESV reservation messages follow the route pre-established by the sender's PATH messages. Thus, a router must forward PATH messages on links that will allow future reservations to succeed. Unfortunately, a router only has a local view of the reservations in the network. Thus, it is likely to make suboptimal forwarding choices. In contrast, with CM, because the gateway has a global resource view, it can make optimal load balancing decisions.

We now relax the third assumption made earlier, that is, that all resource reservations are made through the gateway. Suppose that some reservations in the network are made using a mechanism other than CM (such as RSVP, for instance). In this case, when a gateway receives a join request, it cannot immediately process it, because it does not know the resource availability at the routers. Therefore, on getting a join request, the gateway first computes the best possible path that would join the host to the existing multicast tree. It then sends a message to each intermediate router on the path, asking either for the current reservation state, or giving it the reservation information, and asking it to perform reservation and admission control. With the former, admission control is done at the gateway, and if the admission tests are successful, the routers are requested to modify their reservations. With the latter, admission control decisions are made at each individual router. The result, however, is the same. Either all the routers along the path have sufficient resources and the join succeeds, or one of the routers has insufficient resources. In this case, the joiner is contacted with an indication of the amount of available resources and invited to try again.

We now discuss the case of a non-local join, that is, joining a tree that has senders or receivers outside the local domain. CM handles the case of non-local joins less well because tunnels do not allow resource reservation. Since our approach to wide area groups depends primar-

ily on tunnels, it cannot support reservations for groups that span multiple domains. However, this is true for *all* current multicast reservation protocols. Even RSVP does not guarantee quality of service across tunnels. Since the current MBONE is mostly composed of tunnels, at least in the backbone, RSVP is as much a non-solution as CM! We believe, however, that routing arbiters can be used to create non-local groups with reservations, but we do not discuss this further because this is unlikely to be realized in the near term, if ever, until pricing and differential service issues are sorted out.

5 Some details

In this section, we present some details of our approach, filling in the gaps left in Section 3.

5.1 Gateway as a fail-safe process group

Unlike root-controllers, which are replicated and synchronized, each domain has a single gateway. Thus, if a gateway crashes, though existing multicast groups remain unaffected, no new join and leave requests can be processed. Therefore, CM requires gateways to be fail-proof. Fortunately, we can use a variety of standard techniques to prevent the crash of a gateway [BJ87, B93, BR94]. For instance, we can implement the gateway as a *reliable process group*. In this approach, gateway functionality is provided by a set of processes that use a hard-coded multicast group to keep in tight synchrony. Messages sent to 'the' gateway are automatically replicated and sent to all members of the group. (This can be done, for example, by a router on a LAN that translates the IP address of the group to a multicast Ethernet address in that LAN. Thus, all messages sent to this IP address are automatically received by all hosts on that LAN.) The simplest form of a process group is a primary-backup mechanism. Here, a backup gets a copy of each message sent to and from the primary, and thus has an exact copy of its state. The backup also monitors the primary to see if it is up. If the primary dies, the backup takes over as the primary and establishes another backup. A gateway implemented in this fashion therefore survives the failure of the primary (though not the failure of the router). A similar approach to fault-tolerance has been used in [AP96]. More elaborate techniques for fault-tolerance based on atomic broadcast and virtual synchrony can deal both with process and router failure, and are well-known in the literature [BR94]. These can be readily adapted to our purpose.

5.2 Messages between a host and a gateway

A host can send one of two messages to a gateway: a JOIN message and a LEAVE message. The JOIN message has the following options:

1. *Address:* this the Class D address of the group the host wants to join. If the host is also the creator, then the address can be left unspecified, and the gateway will reply with an unused Class D message.
2. *Creator flag:* this is set if the host is the creator of the group.
3. *Urgent flag:* this is set if the group should be immediately advertised by the nearest root controller.
4. *Local flag:* this is set if the group will be visible only to hosts within the domain.
5. *Access control list:* this is a list of hosts who are allowed to join the group. Alternatively, it may consist of a password that must accompany a join request.
6. *UID:* this is a unique ID associated with a particular join request and gateway. It allows remote gateways to deal with duplicates, as described in Section 6.3.

The LEAVE message has only one field, the address of the multicast group.

The gateway replies to a JOIN message with a JOIN_ACK message. This contains two fields: status and address.

1. *Status:* this is either JOIN_SUCCESSFUL or JOIN_UNSUCCESSFUL.
2. *Address:* if the join is successful and the address was unspecified, this field contains the address assigned to the group.

Join requests are periodically refreshed by IGMP-capable routers on receiving an IGMP reply from a host.

5.3 Messages between a gateway and a root controller

The gateway exchanges four messages with the nearest root controller: JOIN, LEAVE, TUNNEL_CREATE and TUNNEL_DELETE. The JOIN message has the following fields:

1. *Address:* this the address of the group the gateway wants to join. If the domain contains the creator of the group, then the address can be left unspecified, and the root controller will reply with an unused Class D message.
2. *Creator flag:* this is set if the domain contains the creator of the group.
3. *Urgent flag:* this is set if the group should be immediately advertised by the root controller.
4. *Domain ID:* this is the ID of the gateway's domain. We assume that each domain has an ID that is globally disseminated using DNS.
5. *Gateway IP address:* this is the gateway's IP address.

The LEAVE message is identical to the LEAVE message described above.

The TUNNEL_CREATE message asks the gateway to create a tunnel with a peer domain. It contains two fields:

1. *Gateway address:* this is the IP address of the peer gateway.

2. *Peer domain:* this is the ID of the peer domain.

The gateway replies to the TUNNEL_CREATE message by choosing a border router, and informing the peer gateway of the IP address of the border router. It also expects to hear a message from the peer with that peer's border router. This information allows it so create one end of the tunnel. TUNNEL_CREATE messages are periodically refreshed by the root controller.

The TUNNEL_DELETE message is self explanatory, and contains the IP address of the peer gateway.

5.4 Messages between root controllers

Root controllers periodically multicast their state (actually, the deltas in their state) to each other. A root controller may also ask another root controller to dump its entire state over a unicast path. The multicast information has the following fields:

1. *NEW_GROUPS:* A list of new groups that have been created since the last update, and the domains associated with each group.

2. *DESTROYED_GROUPS:* A list of groups that have been destroyed since the last update.

3. *NEW_DOMAINS:* A list of existing groups and the new domains that have joined these groups.

4. *NEW_TUNNELS:* Tunnels that have been created since the last update.

5. *DESTROYED_TUNNELS:* Tunnels that have been destroyed since the last update.

A root controller requests a full update by sending a FULL_DUMP message to another root controller. This contains the requestor's IP address, and the requestee replies with its entire state. A full update is sent using a reliable transport protocol such as TCP.

5.5 Switching trees

Recall that root-controllers typically graft a domain by adding a single tunnel from that domain to a border router in the existing multicast tree. As discussed earlier, locally optimal decisions in creating such links can result in a globally non-optimal solution. Thus, we propose to periodically rearrange tunnels to regain optimality. The root-controller that first received the message to create the group is responsible for switching trees. To switch trees, the root controller sends a TUNNEL_CREATE message to the gateways in the multicast group with the following information:

1. *GROUP:* this is the group for which a forwarding entry needs to be created.

2. *DOMAIN_LIST:* this is the list of domains with which the gateway needs to establish tunnels.

3. *GATEWAY_LIST:* this is the list of IP addresses of peer gateways with which the gateway needs to establish tunnels.

Each gateway contacts the gateways listed in the GATEWAY_LIST to discover the border routers involved in the tunnels, as described in Section 3.3.4. After creating tunnel endpoints, each gateway sends a CREATE_DONE message to the root controller. When the root controller receives CREATE_DONE message from all the gateways, it sends a TUNNEL_DELETE message to the gateways. This message contains only the GROUP address whose corresponding tunnel entry needs to be deleted. The root controller periodically sends the DELETE_ENTRY message until it receives DELETE_DONE messages from all the relevant gateways. For robustness, TUNNEL_CREATE messages are reliably transmitted either by using a reliable transport protocol, such as TCP, or by the standard techniques of timeouts and retransmissions implemented over UDP.

5.6 Domains using distance-vector routing protocols

In a domain that use a distance-vector routing protocol, such as RIP, the gateway does not know the network topology. Therefore, when the gateway receives a join request, it does not know the set of routers to which it should send an ADD message. To deal with this problem, in such domains, members of multicast groups are interconnected using tunnels. These tunnels define a *virtual topology that is known to the gateway.* When a new member wants to join an existing group, the gateway simply creates a tunnel between the new member and the point nearest the new member in that group's multicast tree. This is identical in spirit to the use of tunnels for dealing with the lack of knowledge of network topology between domains.

This is certainly not the optimal way to form multicast trees! RIP domains, however, are typically quite small and hence sub-optimality will not have any pronounced affect on performance. If performance becomes a problem for a given multicast group, the gateway can create a more optimal tree and then initiate a tree switch as described in the interdomain case (Section 3.34).

5.7 Recovery from failure of tunnel endpoints

Tunnels play a crucial role in interconnecting domains in CM. Therefore, it is important to keep tunnels "up". A tunnel fails if one of its end-points fails (recall that the tunnel end-points are border routers of the domains).

We consider recovery from endpoint failure in two cases: (1) the domain runs link-state routing protocol, such as OSPF and (2) the domain runs distance-vector routing protocol, such as RIP.

Case 1: Here, the gateway participates in the link-state routing protocol just like any other router in the domain. When a router fails, the neighboring routers

generate a triggered update which is flooded throughout the domain. Therefore, the news of a border router failure rapidly reaches the gateway. The gateway maps from the border router that failed to the set of tunnels that passed through that gateway. It selects a new border router as the tunnel end-point for every failed tunnel. Since the gateway knows its peer gateway for every failed tunnel, it can communicate the IP address of the new border router to its peers. The peers then reconfigure their border routers, which leads eventually to the resumption of multicast delivery.

Case 2: In a domain that uses a distance-vector routing protocol, when a border router fails, the distance vector to the border router becomes infinite. The gateway, which participates in the distance-vector protocol, just like any other router, therefore detects the failure and starts the recovery process. Recovery is done by selecting a different border router and by informing the peer gateway of the IP address of the new border router, as discussed in Case 1.

6 Evaluation

6.1 Scaling

The limits to scalability of a multicast routing scheme arise from a super-linear growth either in the state stored in routers and control elements or in the number of control messages. In this section, we describe the design features in our scheme that allow it to scale.

6.1.1 Scaling in state

The CM scheme stores state in gateways, in root controllers, and in routers. Here, we show that each component of the system state grows only linearly with the number of multicast groups, and with the size of a given multicast group.

A gateway stores the topology of the domain and the multicast tree corresponding to each multicast group. The topology of a domain is represented by nodes and edges. Clearly, per-node topology state increases linearly with the number of nodes. However, it is possible that per-edge state grows quadratically with the number of nodes, since a network with N nodes can have as many as $O(N^2)$ edges. Fortunately, in most real-life networks, the number of edges is O(N), in which case the topology state overall grows linearly with N. We now turn our attention to the storage requirements to represent a multicast tree. The multicast tree for a group can be either a shared tree, or a per-source shortest-path tree. Assuming that a domain has O(N) edges, a shared tree requires the gateway to keep track of at most O(N) edges per multicast group. Thus, keeping the number of multicast groups fixed, the state grows linearly with N. Similarly, keeping N fixed, the state grows linearly with the number of multicast groups. A per-source shortest path tree, however, does not scale as well. For such a tree, the gateway must keep track of a shortest-path tree (potentially of size O(N)) for each of potentially O(N) sources in the tree, leading to a state of size $O(N^2)$ per multicast group. Thus, to allow scaling, we recommend that all multicast trees be shared trees. To sum up, assuming that domains have O(N) edges, and that all trees are shared trees, the state in the gateway scales linearly with the size of a domain.

CM requires only one gateway per domain. Thus, the number of gateways grows linearly with the number of domains. As a domain grows, however, a single gateway may prove to be a bottleneck. We therefore envisage heavily loaded gateways to be implemented in the form of a computing cluster connected by a fault-tolerant and load-balancing middleware infrastructure. Such clusters can be scaled nearly arbitrarily, as is exemplified in real-life by Web server clusters.

The state in a root controller is the current set of multicast tunnels, and the domains that participate in each non-local multicast group. This state is isomorphic to the state in a gateway. Thus, under the same assumptions, that is, that the backbone topology has O(N) edges (where N is the number of domains), and that the multicast trees in the backbone are shared trees, we obtain a linear scaling in root-controller state.

The state in a router is the forwarding state, which consists either of an io-set (for a shared tree), or a set of (S,G) entries (for a shortest-path tree). With a shared tree, the number of routing entries grows linearly with the number of multicast groups, and sub-linearly with the size of a given group. By aggregating multicast addresses, as described in Section 3.2.2, we can reduce even this overhead substantially. With a shortest path tree, this state grows linearly both with the number of multicast group, and with the size of a given group. For good scaling, again, the use of a shared tree is certainly the best choice.

6.1.2 Scaling in message overhead

We use three techniques to allow our scheme to scale in terms of message overhead: hierarchy, batching, and the use of deltas.

The *hierarchical partitioning* of control among root controllers and gateways allows local state to be hidden from global view. This contributes to scaling in two ways. First, we expect that a considerable number of multicast groups will have only local members. With CM, state regarding these groups is explicitly hidden from global view (in contrast, with schemes such as DVMRP/flood-and-prune, the hiding of local groups is implicitly accomplished by using the TTL field in multicast IP packets). Second, the hierarchy allows distributed control in establishing non-local multicast groups. A root controller only needs to decide which domains must be connected by a tunnel: the actual selection of border routers is deferred to the peer gateways. This decentralized decision-making allows the scheme to make locally optimal choices with-

out having to share domain-specific information with root controllers.

We use *batching* to minimize communication overheads in all message exchanges. Gateways are responsible for refreshing soft state both at routers and at root controllers. This exchange is batched, so that a single message can refresh multiple routing table entries and multiple root-control state entries. Batching also reduces the number of messages exchanged between root controllers to synchronize their state.

Finally, we use *deltas* wherever possible to reduce the size of a state exchange. For instance, root controllers usually send only a delta to peer root controllers. Deltas may lead to drift, however, so we periodically perform a full update to bound the degree of state drift.

6.2 Efficiency

In this section, we analyze the efficiency of the CM scheme and compare it with that of PIM, CBT, and flood-and-prune (also known as DVMRP). Note that we compare only the efficiency of *control*, instead of the efficiency of data movement. This is because the CM scheme allows the construction of shortest-path trees, which are the most efficient for data forwarding (albeit at the cost of an increase in the per-router multicast forwarding state). So, CM can be as efficient in data-movement as any other scheme.

In our analysis, we make the following important assumption: the cost of an algorithm is a function of the number of messages processed, independent of the path taken by a message and of its size. This assumption follows from our expectation that the bottleneck resource in a network is the CPU and interrupt handling cost in processing a message, rather than the uninterpreted forwarding of a message (which is typically performed by 'dumb' hardware at little cost). The assumption reflects current architectures such as Cisco's Silicon Switching, where IP packets with options, which require CPU intervention, take an order of magnitude more time to process than IP packets without options, and future architectures such as MPLS, which can switch in hardware, but must process control messages in software.

We use the following notation. We assume that the network has M nodes, that a multicast group has on the order of N nodes ($N \leq M$), and that there are G groups in the network. We also assume that a multicast group has, on average, S senders (sources).

6.2.1 DVMRP

We first analyze the efficiency of the flood-and-prune (DVMRP) algorithm. This algorithm has three main steps in its operation. In the first stage, the M routers in the network participate in a distance-vector routing algorithm to compute the shortest path interface for an incoming packet from a given source. This requires O(M) messages (assuming that the maximum connectivity of a node is a small integer independent of M) because each of M routers sends a distance vector to a each of its neighbors. In the second stage, a router attached to a source floods the entire network with the source's ID. The flooding operation takes O(M) messages, and each source does a flood, for a total of O(SM) messages. In the third state, each of the M-N routers not in the group can ask for a prune (and prunes may cascade, an effect we will ignore), so that we have an additional O(M-N) prune messages. Thus, the total cost of flood and prune is O(M) + O(SM) + O(M-N) = O(SM) per group. The largest term here is the O(SM) term, since this term grows with the size of the network, independent of the size of a multicast group. That is why flood-and-prune is rather inefficient for sparse multicast groups, as pointed out in References [B93, BFC93]. Note that flood-and-prune simultaneously joins all receivers. Thus, it does not have a per-receiver join cost, reducing its overall cost when a multicast group is dense.

6.2.2 CBT

The Core Based Tree approach is more efficient in its operation. Here, each member of a multicast group sends a join message towards a core node. Multicast-capable routers need to run a multicast routing protocol in order to determine the next hop towards the core, which may use a virtual interface. If a router that is a part of the shared multicast tree sees this message, it adds a path to this node to the tree, and does not propagate the message further. Unlike flood-and-prune, there are no additional costs for flooding and pruning. Thus, CBT has two sets of costs, one for performing multicast routing, and one for joining each sender or receiver to the shared tree. The routing algorithm takes O(M) work, as with flood-and-prune. The cost of each join message depends on how far away a joining node is from the tree. The closer it is to a tree, the cheaper it is to join a tree. Intuitively, therefore, joins to a sparse tree are more expensive than joins to a dense tree.

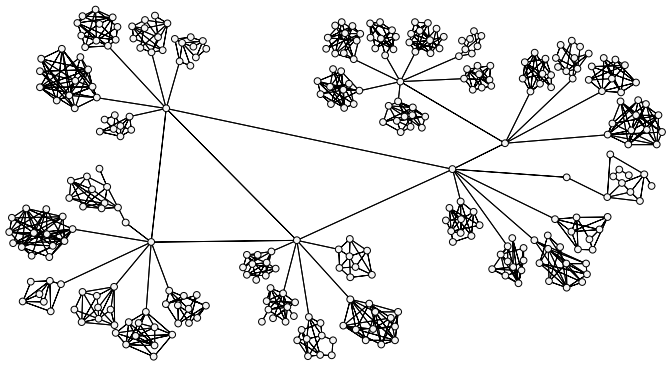


Figure 9: An example of a topology generated by the Zegura-Calvert algorithm

We now analyze the join cost of CBT in more detail.

This is necessary because it turns out that the cost of CM is a constant additive factor more than the cost for CBT (a fact we will prove in Section 6.2.3). Our analysis is empirical and based on simulation. The goal is to compute the mean cost for a node to join a multicast tree as a function of the network size and the number of participants in the tree. The simulation is set up as follows. We use the Zegura-Calvert algorithm [ZCD 97] to generate random multilevel hierarchical networks that are similar in topology to the Internet. An example of a network generated with this algorithm is shown in Figure 9.

We then choose K nodes at random in the network, and join them by a shortest-path tree. This gives us (approximately) the multicast tree that would be created had we used CBT to link these nodes together[4]. We then compute the mean cost from a node chosen at random in the network to join this tree. This is simply the minimum number of hops that separate this node from the closest node already on the tree. We use a variant of breadth-first-search to speed up this computation. We mark all nodes on a multicast tree as having weight 0. We then mark all neighbors of these nodes as having weight 1, taking care never to overwrite a 0 with a 1. In the next step, neighbors of nodes with weight 1 are marked with weight 2, again never overwriting a lower weight. In this fashion, we can quickly compute the number of nodes in the network that have a particular cost to join a particular tree. The mean cost to join this tree, therefore, is the weighted average of these numbers. We average this cost twice more: first over randomly selected multicast trees within the same network, and second over many random networks. This procedure, described in pseudocode below, thus empirically computes the mean join cost for a node as a function of the network size and the size of the multicast tree. The result of our simulations, where we chose eight random trees for a given a network, three random networks for each network size, and four network sizes, is shown in Figure 10. Figure 11 is a close-up of the knee of the curves.

```
for M_MIN ≤ M ≤ M_MAX {
    for each randomly generated tree (using the Zegura-
        Calvert algorithm) of size M {
        for (K=0; i < M; K++) {
            select a random subset of K nodes and join
                them to form a multicast tree
            for each j not in the subset {
                compute the cost from j to a node in
                the tree using the optimized algorithm
                described above
            }
            compute average cost to join this tree with
                K members
        }
        compute average cost to join a tree as a func-
            tion of K
    }
    compute average cost to join a tree as a function of
        K and M
}
```

Pseudocode for computing mean cost to join a multicast tree

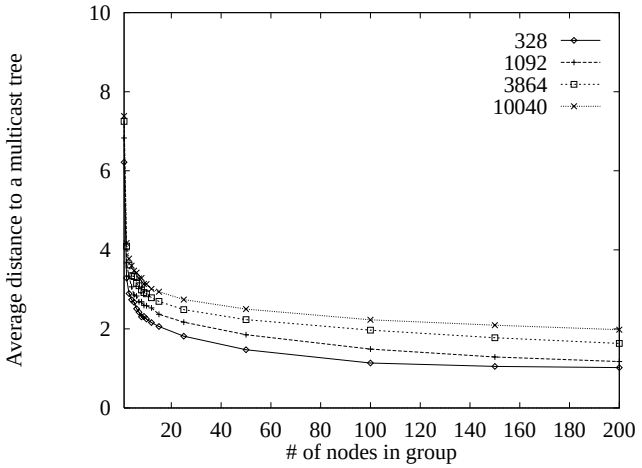


Figure 10: Mean cost to join a multicast tree varying network and group size

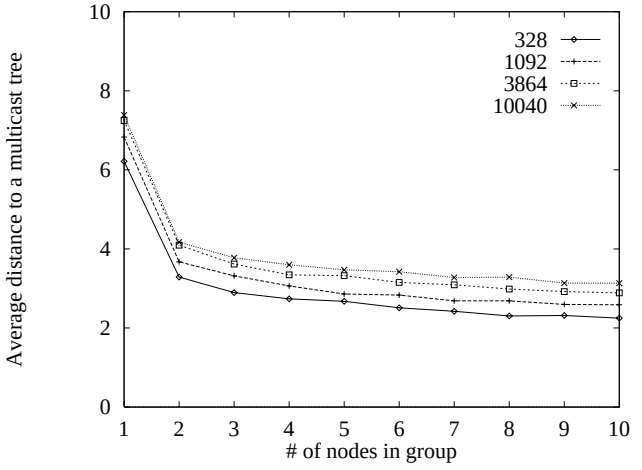


Figure 11: Mean cost to join a multicast tree varying network and group size (close up of knee region).

A glance at Figures 10 and 11 reveals two key features. First, the cost to join a multicast tree drops rapidly with the size of the group. With a group size of 3 or more, the join cost is nearly constant. Second, the join cost

[4] The tree is approximate because a badly placed core router could give us somewhat skewed trees. We expect that in the normal case, with a well-chosen core, the shared core-based tree and the shortest-path tree will approximately coincide

is nearly independent of the size of the network. As we increase the number of nodes in the network from around 300 to around 10,000, the mean join cost increases only by about one additional hop. The reason for these features is that the Zegura-Calvert algorithm generates hierarchical topologies. Choosing two nodes at random results in a multicast tree that almost surely touches the backbone. Thus, the cost to join any other node is just the cost to reach the backbone, which averages to between three and four hops. This cost is more or less unaffected by the size of the network, since the depth of a hierarchical network grows only logarithmically with the number of nodes in it. Zegura et al argue that the topologies produced by the algorithm match the real Internet's characteristics fairly well. If this is the case, then we can expect to see a nearly constant join cost when using CBT in the Internet. Thus, the total cost for CBT is O(M) + a constant cost per join = O(M) + O(N)

6.2.3 CM

Given the cost of the CBT algorithm, as analyzed above, we compute the cost of CM as follows. With CBT, consider a join message sent from a router R_x to the core along path $R_x \rightarrow R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_n$ where R_n is either a node on the shared tree or the core itself. This join requires n messages, since each node along the path must examine the join message. With the CM scheme, R_x sends a join message to the gateway, which sends an add message to R_1, R_2 etc. Since the gateway sends n messages to the routers along the path, and this path is the same as the path taken by a CBT shared tree, with CM the cost for this join is the cost for CBT+2, where the '+2' accounts for the two additional messages exchanged between R_x and the gateway. In other words, CM has a constant additive factor in cost compared to CBT per join, nearly independent of the size of the group and the size of the network. Accounting form O(M) work for the routing protocol, this leads to a cost of O(M) + O(N) work for N joins.

6.2.4 PIM

The cost for the PIM algorithm depends on whether it is dense-mode or sparse mode. With dense-mode PIM, the cost is the same as flood-and-prune, i.e. O(SM). With sparse-mode PIM, the cost is the same as of CBT. In this analysis, we ignore the control cost of changing from a shared tree to a shortest-path tree, which is nominal.

The table below compares the costs of the various algorithms. It is clear that CM is as efficient as CBT and sparse-mode PIM.

DVMRP	O(SM)
CBT	O(M)+O(N)
CM	O(M)+O(N)
PIM-sparse	O(M)+O(N)
PIM-dense	O(SM)

6.3 Correctness

In this section we outline proofs of correctness for the algorithms proposed in earlier sections. We make the following assumptions:

1. The period of state exchange between root controllers is bounded, and a state exchange message is delivered with non-zero probability.
2. Gateway to root-controller and root-controller to gateway communications take finite time.

The second assumption assures us that a gateway can detect and recover from the failure of a root-controller in finite time.

Lemma 1 *Data transmitted by a new sender to a multicast group G will reach all members of G within a bounded time.*

Proof: We need to consider two cases: (1) Group G is local, that is, G does not span multiple domains and (2) Group G spans multiple domains.

Case 1: A new sender contacts the gateway in its domain. The gateway either (a) grafts the sender to an existing multicast tree, if it exists or (b) creates a new tree with the sender as the only node. Grafting can be completed in bounded time because it is a matter of computing the shortest path connecting the sender to the existing tree, and sending ADD messages to the appropriate routers. Even if routers selected for grafting fail, their failure is known to the gateway in bounded time from the domain's routing algorithm (as shown in Section 5.7). Thus the gateway can recover from these failures and complete the addition in bounded time. Case (b) does not even involve the computation of shortest path, and is trivially satisfied.

Case 2: A new sender contacts the gateway in its domain and the gateway, in turn, contacts its nearest root controller. Communications between the gateway and the root controller are subject to a timeout, as explained in Section 3.2.2, so the failure of the nearest root controller can be recovered from in a finite time. There are now two possibilities: (a) the root controller knows about group G, (b) the root controller does not know about the existence of group G. In case (a), the root controller forwards the JOIN request to the gateway of the domain that created group G for admission control. Since the gateways are modelled as reliable process groups, they are always "up". Therefore, the gateway will reply to the root controller in bounded time. Once the root controller receives an "OK" message from the gateway, it grafts the new sender onto the existing tree in bounded time, in a manner similar to that described in Case 1. One subtle possibility is that the root-controller may fail after contacting a gateway, but before receiving a reply. In this case, the gateway originating the join request retries the request with a different

root controller. Thus, each gateway should be prepared for receiving duplicate requests; idempotency is assured by associating a join request with a unique serial number, as described earlier in Section 5.2.

Case (b) is slightly more involved. If the root controller does not know about the existence of the group, either the group does not exist or the group has been formed since the last time the root controllers synchronized. In either case, the root controller has to wait *at most* the period of state exchange to know about the group. If the group does not exist, the root controller creates the new group and forward the information to the other root controllers. If the group exists, the root controller follows the steps of case (a) to graft the new sender onto the existing tree. Since case (a) can be performed in bounded time and the root controller has to wait at most for the period of state exchange, case (b) can also be performed in bounded time.

Therefore, the new sender will be part of the multicast tree in bounded time. This implies that the receivers of the multicast group will start receiving the data from the new sender in bounded time.

Lemma 2 *Data multicast to a group G will reach a new receiver of group G within a bounded time.*

Proof: The proof is similar to that of Lemma 1.

Lemma 3 *A new multicast group G can be created in bounded time.*

Proof: there are two possibilities: (1) Group G is local and (2) Group G spans several domains.

Case 1: If the new group is local, the gateway need not forward information about the group to the root controller. Therefore, creation of a new group merely involves creating an entry in the gateway corresponding to group G.

Case 2: If the new group is not local, the gateway informs its nearest root controller that a new group needs to be created. If the "urgent flag" is set, the root controller informs all the other root controllers immediately. Otherwise, other root controllers are informed after the periodic exchange of state. Since the period of state exchange is finite, and state exchange messages are delivered with non-zero probability (the first assumption made earlier), all root controllers will eventually know about group G and create a corresponding entry. Even in the worst case, a root-controller can always request a full dump of a peer root-controller's state through a reliable TCP channel.

Therefore, a new group will be created in bounded time.

Theorem 1 *CM can construct a correct multicast tree for a group with dynamically changing membership within a bounded time.*

Proof: Constructing a multicast tree with dynamically changing membership has three parts:

1. Creation of a new group G
2. Adding a new sender to G
3. Adding a new receiver to G

Lemmas 1 through 3 ensure that each of the above three steps can be executed in bounded time using CM. That proves the theorem.

7 Discussion

Centralized multicast (CM) is a technique that separates control and data flow for Internet multicast routing. The control structure consists of root controllers at the highest level and a gateway per domain at the next level. Routers are managed by control elements and are simply used for data forwarding.

The main drawbacks of our scheme are (a) a higher join latency, (b) overhead for maintaining soft state, (c) need for fault-tolerant gateways and (d) the lack of reservation across tunnels. Join latency in CM is slightly higher than with other schemes because the join is not data driven. We require explicit signaling between members, gateways and the root controllers to create a multicast tree, and to add or delete group members. We believe that this is not a serious disadvantage. Most multicast groups are used for a long time after the initial setup. This is particularly true for groups used for data dissemination, distance education, videoconferencing etc. Thus, the added join latency is not likely to be a problem in practice, at least for a large class of applications. Second, soft state at the routers and the root controllers is refreshed periodically by the gateways. This entails additional signaling load on the gateways. While we can eliminate soft-state to gain efficiency, we believe that the additional work is more than made up by the gain in overall robustness. Third, we require gateways to be fail-proof. At first sight, this seems to be an imposing requirement. However, this is a common one in real life. Most commercial servers, such as transaction servers and high-profile web servers cannot fail without serious consequences. A variety of well-known software systems, such as clusters and process groups, allow us to easily satisfy this requirement. Finally, the domains in inter-domain multicast trees are connected by tunnels and when tunnels are used, no guarantees can be made on the quality of service. However, this is true for all multicast schemes. It is likely that we will see QoS-enabled wide-area connections only when the related pricing and differential service issues are properly addressed.

We believe that the advantages of centralized multicast, that we enumerate next, far outweigh its shortcomings. First, centralized multicast provides the flexibility

of building either a shared tree or a source-based shortest path tree on the same framework. Thus, a network operator can trade optimal data paths for an increase in the control state. Even with a shared tree, CM allows switching to a more optimal multicast tree after a sequence of join and leave operations, thus restoring a shared tree to optimality. Second, with CM, routers are much simpler because they do not need to take part in the multicast routing protocol. Routers are used as forwarding engines only: the RISC approach in microprocessors has proved that 'big and dumb' is better! Third, CM simplifies group control. Since the gateway controls the join operation, it can allow or disallow members to join a multicast group based on an access control list. Fourth, because CM allows the use of tunnels, building inter-domain multicast trees is simple. Tunnels can also be used in RIP-like domains which use distance vector routing protocols. Fifth, CM leverages SNMP to alter routing table entries. This is possible because of the clean separation of control and data flows. Sixth, CM provides a choice between a triggered join and a lazy join through the use of the urgent flag during a join operation. Triggered join has the benefit of low join latency at the expense of higher signaling load, while lazy join provides the benefit of batching updates at the expense of higher join latency. Finally, gateways, in conjunction with the root controllers, can perform a host of useful functions, such as, allocation of multicast addresses on demand, aggregation of multicast addresses for scalability, distribution of group keys for secure multicast sessions and even reservation and allocation of resources.

The fundamental difference between CM and other schemes in the literature is that with CM, routers need not participate in control protocols in order to set up multicast forwarding state. In most schemes, forwarding state is set up either in response to the arrival of join requests from IGMP-enabled routers, or, as in the case of DVMRP, the arrival of prune packets from a downstream router. This has two consequences. First, the construction of the multicast tree is a distributed process. Second, each router must participate in the construction of the multicast tree. Both of these raise problems that are cleanly addressed by CM. Any distributed computation is, by its very nature, acting on a partial understanding of global state. Thus, the outcome is usually non-optimal. In contrast, within a domain, CM has complete knowledge of the topology and the members of every group. Thus, it can create more efficient multicast trees than with any distributed approach. CM also frees routers from the burden of processing multicast join and leave messages. Current-generation routers are very good at forwarding uninterpreted data, but process messages less well. Most current multicast routing schemes require complex processing of join and leave requests at routers. In contrast (and this is the very rationale for CM), our scheme requires routers to only add and delete forwarding entries from routing tables, which is a much simpler task. Thus, we believe that CM has significant advantages over most existing schemes, by design. We now focus our attention on comparisions with specific schemes.

Compared with DVMRP, CM is more efficient because it does not have the overhead of periodic flooding and pruning. We analyzed the efficiency of DVMRP and CM in Section 6.2, and this clearly shows the gain in efficiency by using CM. PIM has been proposed as an efficient solution to multicast routing because it supports both dense and sparse networks, and can automatically switch from a shared tree to a shortest-path tree. However, its main failing is that it assumes that all routers in the network run PIM (otherwise, the shortest-path interface that is determined from the unicast routing table does not make sense). Unlike PIM, CM does not require all routers to run CM protocol. This is possible because non-CM routers can be connected using tunnels. Thus, we believe that, unlike PIM, CM satisfies our requirement that a multicast routing protocol should be incrementally deployable. Even if PIM were to be enhanced to give rendezvous points and routers a copy of the domain topology, it would still lack the global network view that CM gateways have. Moreover, PIM does not support a two-part hierarchy, which we believe is critical to allow scaling. A recent contender for multicast routing, Border Gateway Multicast Protocol (BGMP) [TEM97], is being developed in IETF as the inter-domain multicast routing protocol. BGMP builds a bi-directional shared tree in the backbone and allows the grafting of shortest-path branches. However, BGMP still needs help from MASC [EHT97] for allocation of blocks of multicast addresses for aggregation and from MBGP [BCKR97] for distribution of multicast address prefixes between domains. CM can achieve the same goals of BGMP, MASC and MBGP using its gateways and root controllers except that CM separates control and data while they do not. We believe that this separation is critical for simplifying routers and enabling widespread deployment of IP multicast.

At this point, we would like to review the design features in CM that allow it to satisfy our requirements for scalability, efficiency and incremental deployability. CM is scalable in message overhead because of its use of hierarchy, batching, and deltas. It is scalable in state because of its used of shared trees and the careful use of hierarchy. CM can create efficient shortest-path multicast tree (but at the expense of a decrease in scalability). Thus, it can be as efficient as the best-possible multicast routing protocol. In practice, we believe that the shared tree approach, in conjunction with periodic tree switching, is the best solution. Finally, CM is designed to be incremantally deployable. It uses SNMP to manage router state, and does not require routers to implement any additional protocols. In particular, routers do not need to participate in control algorithms, making them simpler. Thus, we believe that CM meets the requirements for practical adoption that we laid out in the Introduction.

8 Conclusions

Centralized multicast is a new way of looking at multicast routing in the Internet. Traditionally, routers in the Internet not only take part in forwarding the data traffic, but also in the routing protocol. This has led to tremendous complexity of the routers, particularly for multicast and as a result, IP multicast has not become as mainstream a technology as was expected. As a solution to the problem, we have proposed centralized multicast which uses a hierarchy of control elements to set up multicast trees, leverages SNMP to update the routing table entries and uses the routers simply as forwarding engines. We have shown through analysis that CM scales in state and message overhead and is very efficient. We also proved that CM can form a correct multicast tree for a dynamically changing set of members within a bounded time. As discussed in Section 7, we believe that CM is suited for widespread adoption in the Internet. An implementation of the scheme is currently in progress, and will be reported on in future work.

9 Acknowledgements

The idea of centralized multicast arose, in part, in discussions with Rosen Sharma. The simulations on join cost were done by Snorri Gylfason. Our sincere thanks to them both.

References

[AP96] S. Aggarwal and S. Paul, "A Flexible Protocol Architecture for Multi-Party Conferencing," *Proceedings of ICCCN'96,* Pages 81-91, October 1996.

[B93] K.P. Birman, "The process group approach to reliable distributed computing," *Communications of the ACM*, Vol.36, No.12, Pages 37-53, December 1993.

[B97] A. Ballardie, "Core Based Trees (CBT Version 2) Multicast Routing – Protocol Specification," *RFC-2189*, September 1997.

[BCKR97] T. Bates, R. Chandra, D. Katz and Y. Rekhter, "Multiprotocol Extensions for BGP-4," *Internet Draft*, draft-ietf-idr-bgp4-multiprotocol-01.txt, September 1997.

[BFC93] T. Ballardie, P. Francis and J. Crowcroft,"Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing," *Proceedings of ACM SIGCOMM '93*, September 1993.

[BJ87] K.P. Birman and T.A. Joseph, "Reliable Communication in the Presence of Failures," *ACM Transactions on Computer Systems*, Vol. 5, No. 1, February 1987.

[BKTN98] S. Bhattacharyya, J.F. Kurose, D. Towsley and R. Nagarajan, "Efficient Rate-Controlled Bulk-Data Transfer using Multiple Multicast Groups," To appear in *Proceedings of IEEE INFOCOM '98.*

[BR94] K.P. Birman and R. van Renesse, "Reliable Distributed Computing with the ISIS Toolkit," *IEEE Computer Society Press*, Los Alamitos, California, 1994.

[BZB+97] B. Braden, Ed., L. Zhang, S. Berson, S. Herzog and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version1 Functional Specification," *RFC-2205*, September 1997.

[D88] S.E. Deering, "Multicast Routing in Inter Networks and Extended LANs," *ACM Computer Communications Review*, Vol. 19, No. 4, 1988, Pages 55-64.

[D89] S.E. Deering, "RFC-1112: Host Extension for IP Multicasting," August, 1989.

[DO97] D. DeLucia and K. Obraczka, "Multicast Feedback Suppression Using Representatives," *Proceedings of IEEE INFOCOM'97.*

[DEF+94] S.E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C-G Liu and L. Wei, "An Architecture for Wide-Area Multicast Routing," *Proceedings of ACM SIGCOMM '94*, Pages 126-135, October 1994.

[DEF+96] S.E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C-G Liu and L. Wei, "The PIM Architecture for Wide-Area Multicast Routing," *IEEE/ACM Transactions on Networking,* Vol. 4, No. 2, Pages 153-162, April 1996.

[EFH+97] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification," *RFC-2117*, June 1997.

[EHT97] D. Estrin, M. Handley and D. Thaler, "Multicast-Address-Set advertisement and Claim mechanism," Work in Progress, June 1997.

[FJM+95] S. Floyd, V. Jacobson, S. McCanne, C-G. Liu and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *Proceedings of ACM SIGCOMM '95*, Pages 342-356, October 1995.

[GAV+ 98] . Govindan, C. Alaettinoglu, K. Varadhan, D. Estrin, "Route Servers for Inter-Domain Routing," *to appear in Computer Networks and ISDN Systems*, 1998.

[H96] M. Hofmann, "A Generic Concept for Large-Scale Multicast," *Proceedings of International Zurich Seminar on Digital Communications (IZS '96)*, Zurich, Switzerland, Springer Verlag, February 1996.

[HSC95] H.W. Holbrook, S.K. Singhal and D.R. Cheriton, "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation," *Proceedings*

of ACM SIGCOMM '95, Pages 328-341, October 1995.

[J94] V. Jacobson, "Multimedia Conferencing on the Internet," Tutorial 4, *ACM SIGCOMM 94*, August 1994.

[L96] N. Lynch, "Distributed Algorithms," *Morgan Kaufman'96.*

[LLG96] B.N. Levine, D.B. Lavo, and J.J. Garcia-Luna-Aceves, "The Case for Reliable Concurrent Multicasting Using Shared ack Trees," *Proceedings of ACM Multimedia '96.*

[LPA98] X. Li, S. Paul and M.H. Ammar, "Layered Video Multicast with Retransmission (LVMR): Evaluation of Hierarchical Rate Control," To appear in *Proceedings of IEEE INFOCOM '98.*

[M97] T. Montgomery, "A Loss Tolerant Rate Controller for Reliable Multicast," *IRTF Meeting*, Cannes, France, September 1997.

[Mo94] J. Moy. Multicast routing extensions for OSPF, *Comm. of the ACM* Vol. 37, No. 8, Pages 61-66, August 1994.

[Mo94a] J. Moy, "OSPF Version 2," *RFC-1583*, March 1994.

[Mo94b] J. Moy, "Multicast Extensions to OSPF," *RFC-1584*, March 1994.

[MJV96] S. McCanne,V. Jacobson and Martin Vetterli, "Receiver-Driven Layered Multicast," *Proceedings of ACM SIGCOMM '96*, October 1996.

[NBT97] J. Nonnenmacher, E. Biersack, and D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission," *Proceedings of ACM SIGCOMM '97*, September 1997.

[P92] R. Perlman "Interconnections: Bridges and Routers," *Addison-Wesley Publishing Company, INC.*, 1992.

[PSLB97] S. Paul, K.K. Sabnani, J.C. Lin, and S. Bhattacharyya, "Reliable Multicast Transport Protocol (RMTP)," *IEEE Journal on Selected Areas in Communications*, Vol. 15, No. 3, April 1997, Pages 407-421.

[TEM97] D. Thaler, D. Estrin and D. Meyer, "Border Multicast Routing Protocol (BGMP): Protocol Specification," *Internet Draft*, draft-ietf-idmr-gum-01.txt, expires April 1998.

[WMK95] B. Whetten, T. Montgomery, and S. Kaplan, "A High Performance Totally Ordered Multicast Protocol," *Theory and Practice in Distributed Systems K.P. Birman, F. Mattern, A. Schiper (Eds) Springer Verlag LNCS 938.*

[VR98] L. Vicisano and L. Rizzo, "TCP-like Congestion Control for Layered Multicast Data Transfer," To appear in *Proceedings of IEEE INFOCOM '98.*

[YGS95] R. Yavatkar, J. Griffioen, and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications,"*Proceedings of ACM Multimedia '95*, Pages 333-344.

[ZCD 97] .W. Zegura, K. Calvert and M.J. Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology," *IEEE/ACM Transactions on Networking* , December 1997.

[ZDE+93] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zapppala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network Magazine*, September 1993.