ELSEVIER

Computer Networks 32 (2000) 35–60

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

A flexible multicast routing protocol for group communication

Sudhir Aggarwal [a,*], Sanjoy Paul [b], Daniel Massey [c], Daniela Caldararu [d]

[a] *Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, USA*
[b] *Bell Laboratories, Holmdel, NJ 07733, USA*
[c] *Department of Computer Science, UCLA, Los Angeles, CA 90024, USA*
[d] *Department of Computer Science, SUNY at Binghamton, NY 13902, USA*

Abstract

Multicast routing, once dominated by a single routing protocol, is becoming increasingly diverse. It is generally agreed that at least three routing protocols, PIM, DVMRP, and CBT will be widely deployed and must interoperate. This signals a shift from the Mbone as one large domain to a collection of administrative domains where each domain selects its own multicast routing protocol.

This paper proposes another multicast routing protocol, Conference Steiner Multicast (CSM), that is suited for domains that implement OSPF as the unicast routing protocol. CSM is targeted towards (sparse) multicast conferencing and online discussion groups. Characteristics of such discussion groups include any member being a speaker or listener and dynamic changes in the group membership. CSM is futhermore well suited for domains with mobile hosts because its basic architecture can support a mobile environment.

CSM is based on the use of a shared, heuristic Steiner minimal tree for interconnecting group members. A key component of the design is that it dynamically and reliably shifts to a different tree as changes warrant. CSM supports rudimentary entry control for security and permits application assistance over routing decisions (termed Application Assisted Routing).

This paper describes the architecture of CSM as well as a prototype implementation. Several CSM routers have been interconnected to form a Multicast Steiner Backbone (Msbone). Standard applications such as vat, vic, and wb [V. Jacobson, Multimedia Conferencing on the Internet, Tutorial 4, ACM SIG-COMM 94, August 1994] have been modified to run on Msbone. CSM is designed to connect to the Mbone via interoperation with DVMRP, and as interoperation standards develop it should be capable of implementing these standards.

Keywords: Multicast routing; Conference; Steiner tree; Shared tree; Shortest path tree; Mobile multicast; Inter-domain multicast; Switching tree

1. Introduction

Most communication on the Internet is based on a single sender, single receiver model. Examples include applications such as telnet, ftp, and web browsing. Many newer applications, however, do not fit into this model. Any application where the same data is sent to multiple receivers, such as an audio conference, is better supported by a model termed multicast where data is sent to a specific group and is only duplicated for delivery purposes when necessary. In the past decade, there has been an ever increasing interest in the design and

* Corresponding author.
E-mail addresses: sudhir@research.bell-labs.com (S. Aggarwal), sanjoy@research.bell-labs.com (S. Paul).

1389-1286/00/$ - see front matter
PII: S1389-1286(99)00123-1

implementation of multicast capabilities [6,11,13,14,32,42,43]. The idea was first implemented in the multicast backbone (Mbone) [15,42]. The Mbone began as a small virtual network on top of the existing Internet, workstations served as routers, and multicast packets were exchanged by encapsulating them in unicast packets, termed tunneling. The Distance Vector Multicast Routing Protocol (DVMRP) [42] specified the protocol for routing and delivery of multicast packets. The Mbone today covers thousands of networks with growing portions of the Mbone consisting of entire administrative domains with production routers handling both unicast and multicast routing. It is now almost universally agreed that the Mbone will not consist of a single routing protocol. Instead different domains will choose to implement different multicast routing protocols, just as the case for unicast routing protocols.

DVMRP remains the clearly dominant routing protocol. DVMRP, and Multicast Open Shortest Path First (MOSPF) [32], are designed for densely distributed group members. Furthermore, these schemes were designed in support of separate source based trees for each sender. Important protocols that have focused also on sparsely distributed group members, such as Protocol Independent Multicast (PIM) [12] and Core-Based Tree (CBT) [5], have received considerable attention. More recently, protocols like EXPRESS [18] and Simple Multicast [7] have simplified the multicast address allocation problem faced by PIM and CBT, respectively.

Into this growing mix of routing protocols, this paper explores the design and implementation of Conference Steiner Multicast (CSM), first proposed in [1]. This protocol is not proposed as the sole routing protocol for multicasting, as no single protocol will likely suffice. Instead CSM is designed as a protocol that could be used for a domain such as a university or company. The protocol is well suited for domains that wish to eventually support mobile hosts and that use Open Shortest Path First (OSPF) for unicast routing. We also believe that the protocol warrants consideration as a backbone protocol, similar in nature to Border Gateway Multicast Protocol (BGMP) [29]. CSM could connect a site to the Mbone through interoperation with DVMRP. While the standards for multicast routing protocol interoperations are still undergoing active development, we anticipate that CSM will be able to implement the standards as they take shape.

CSM is based on the use of a (shared) heuristic Steiner minimal tree for connecting members of a group. However, by using our notion of Application Assisted Routing (AAR), the shared tree for any particular conference can be determined through input from conference members. CSM has the following properties:

1. It supports group communication that takes place on a shared tree over which each user can send or receive. It can be used to support both sparse and dense groups although we feel it is more appropriate for the support of sparse groups.
2. It provides for basic entry control into a discussion group.
3. It supports application assisted routing where the application (specific discussion group) can suggest the type of tree to be used for multicast.
4. It provides for dynamic joining and leaving of members of the discussion group and the same primitives are used to implement the mobility aspects of the protocol.
5. It provides for a mechanism to asynchronously (with respect to membership changes) switch to a 'better' tree when necessary. The ability to seamlessly move to another tree is also a key component for supporting mobility.
6. It can scale to large numbers of multicast groups.
7. It separates issues of fault tolerance into standard mechanisms for handling failures of routers and links in the Internet and specific mechanisms for handling failures of CSM components.
8. It defines how inter-domain multicasting would be handled.

The idea of using shared trees for group communication is not novel. For example, the CBT [5] protocol also advocates the use of a shared tree. We argue, however, that it is preferable to use heuristic Steiner minimal trees rather than shortest path or ad hoc centered trees, as it will likely be essential to conserve network resources as the

number of multicast conferences in the network increases dramatically. Our approach is in contrast with [14] which suggests that the cost of shortest path source based trees is really not that bad as compared with heuristic Steiner minimal trees. Furthermore although the cost of using core based trees [10] might also be cost effective, we see no advantage to doing so as they may suffer from concentration problems unless the cores are chosen from a sufficiently wide set.

The idea of switching to another tree has also been previously suggested (see for instance [14] again) but to our knowledge no specific mechanisms or architectures have been proposed. [10] discusses migration of cores but also does not propose a precise architecture to effect the migration. In this paper, we discuss these mechanisms in great detail. We also explore to a greater extent than usual the issues of reliability, authentication and interdomain operations.

Our work describes how mobile multicast can be handled and this has not been proposed by other multicast protocols. The same ideas that are used to implement discussion groups with participants leaving and joining fairly regularly can also be used to support multicasting where the applications are *mobile*. Although we have not determined the overhead of supporting mobility using this approach, we believe it is worth exploring.

Our implementation illustrates many of the features of our protocol, shows CSM's ability to support existing conferencing applications, as well as its potential for interoperation with other multicast routing protocols.

In Section 2 of this paper, we discuss why a heuristic Steiner minimal tree is appropriate for discussion groups. In Sections 3 and 4, we present the CSM protocol architecture and show how it supports conferencing and AAR. The extensions necessary for supporting mobile applications are discussed in Section 5, resulting in mobile CSM. Implementation of CSM is described in Section 6, where we describe the development of our Msbone using CSM capable routers. We show how existing Mbone applications such as vat, vic, wb, etc. can interoperate transparently, over both our Msbone and the current Mbone. Section 7 compares our protocol with other multicast protocols developed over the past several years and the final section presents some conclusions and thoughts on future work. Appendix A contains a list of abbreviations used in this paper.

2. Multicast using Steiner trees

The problem of determining the minimum weight tree that spans a given subset of nodes D of a graph $G = (V, E)$ with edge weights is know as the Steiner Problem for graphs [17]. A Steiner minimal tree is defined as the optimal shared tree if the sum of the edge weights is to be minimized. The problem has been well studied and is known to be an NP-complete problem [24]. Reasonable heuristics exist [26,35,38], however, for finding an approximate solution in polynomial time. See also [31,33] for more efficient heuristics in the class that uses shortest path and minimal spanning tree algorithms. The time complexity of these heuristics is typically $O(|V|^3)$. For example, we use the Takahashi Matsuyama heuristic [38], which is $O(|D||V|^2)$ to determine our heuristic Steiner trees [3]. These heuristics are guaranteed to produce a tree whose cost is within twice that of the Steiner minimal tree. Thus, although finding a Steiner minimal tree is difficult, it is fairly straightforward to find heuristic Steiner trees. See [21,40] for interesting surveys on such problems. Ad hoc centered trees that use reasonable heuristics for finding 'good' shared trees may also be useful and are supported by our notion of Application Assisted Routing. In the rest of this paper, we will use the term CO Steiner tree to mean an approximate (close-to-optimal) Steiner minimal tree determined using a heuristic such as Takahashi Matsuyama. We will use the term Steiner tree to refer to a Steiner minimal tree or a CO Steiner tree.

2.1. Steiner trees and shortest path trees

Our architecture for CSM is only dependent on running an algorithm that determines a shared tree. In fact our routers can run a variety of shared tree algorithms based on traffic considerations. Thus we implement Application Assisted Routing where the application itself provides information

on the best-shared tree to use. As default, however, we create a CO Steiner tree. For example, if the traffic was such that a single source was the dominant transmitter and delay was important then a heuristic such as [27,28] could be used to obtain a shared tree. Or if it was felt that the traffic was likely to come equally from a fixed number of participants, then an average centered shared tree [39,41] could be used. Heuristics to select cores for core based trees [10] could also be used for the shared tree. However, as discussed in [10] the problem of concentration can become a worse problem for core based trees unless the cores are selected from a wide enough set.

Many computationally tractable heuristic algorithms have been developed in the literature to determine CO Steiner trees. For current multicasting, however, the dominant algorithms in use create a shortest path tree from the source (assuming a single source) to the receivers [37,44]. The shortest path tree is the tree that results when the shortest paths to each receiver are combined to form a tree by not duplicating common edges. Steiner trees have been deemed to be unsuitable for multicasting as compared with shortest path trees because of two problems:

1. the maximal delay is longer than that for shortest path trees; and
2. it is often argued that traffic is concentrated on the Steiner spanning tree links whereas in the shortest path trees the traffic is more distributed.

Shortest path trees are usually optimal if it is desired to minimize the delay from each conference participant, although even this may not be true if the additional load imposed by shortest path trees is significant. If necessary, maximal delay bounds can be incorporated into the shared tree heuristics. We believe that the low cost of shared trees is a more compelling argument and that the network delays are greatly dominated by delays in the software processing of the protocol stack in the end applications. It was also shown by [14] that the maximum path lengths of CO Steiner trees seemed to be at most twice that of shortest path trees when comparing single sources.

The second argument is actually misleading when one considers the situation of having a large number of discussion groups, each of which is a sparse subset of a larger graph. Actually, the notion of concentration is rarely defined precisely. Intutively, for a single discussion group it is true that the traffic is 'concentrated' on the Steiner links of that discussion group. This concentration, however, simply means that the Steiner tree links support all the traffic and that the other links in the network support zero traffic. For source based shortest path trees more links in the network are supporting traffic related to that discussion group. In fact more traffic is generated overall because the shortest path trees tend to bifurcate earlier and hence a packet multicast over a Steiner tree that would not be duplicated until later, is often duplicated very early in a shortest path tree. See Fig. 1. Many studies have shown that there are inefficiencies of shortest path trees over Steiner trees. These inefficiencies translate into overall load in the network rather than problems of 'concentration' when discussing large numbers of multicast groups.

In the next subsection we discuss some results of a study [9] that compares Steiner trees to shortest path trees when there are a large number of discussion groups in the network.

2.2. Simulation results of Steiner trees vs. shortest path trees

The essence of our argument is that as the number of discussion groups increases in the net-

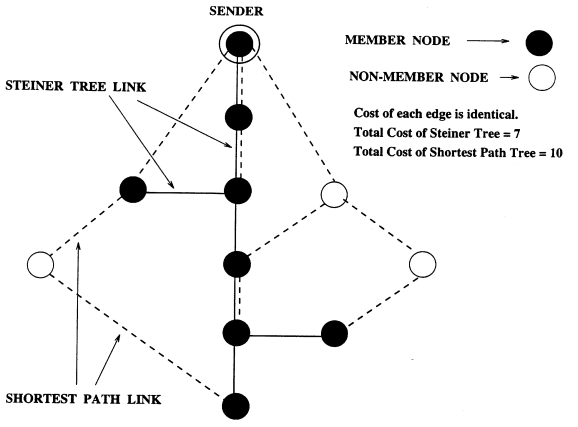


Fig. 1. Steiner tree generates less traffic compared to shortest path tree.

work, the total traffic increases more for source-based trees than for Steiner trees. With a large number of discussion groups, just as the total traffic is distributed throughout the network for source-based trees, so will the total traffic be distributed throughout the network for Steiner trees. In fact, using the law of large numbers, one would expect that any particular link would have a normally distributed traffic load with a mean that would be less for Steiner trees. Thus, source-based trees lose the much acclaimed advantages of distributing traffic in a real network with large numbers of multicast sessions running at the same time.

In our simulation study, CO Steiner trees using the Takahashi Matsuyama (TM) heuristic were compared with shortest path trees. Although we used the TM heuristic, we could well have used more efficient heuristics such as [31,33]. However, we used TM because it is readily available and is more commonly used as a Steiner tree heuristic. Both directed and undirected graphs were considered, but we report on only the undirected case.

Graphs were generated using a modified version of Waxman's [43] approach. In order to be closer to real world networks, graphs were generated by first generating a number of autonomous systems and then interconnecting a small percentage of the nodes in each autonomous system with nodes in neighboring systems. The average node degree ranged from 5 to 10 in the study. For each autonomous system, the nodes were generated by uniformly distributing them in a rectangular area. The probability of connecting two nodes depended on the distance between the two nodes:

$$P = e^{(-d(u,v)/L\alpha)},$$

where $d(u,v)$ is the distance between the two nodes u and v, L the maximum distance between any two nodes in the graph, and α is a parameters in (0,1]. As α decreases, the density of short edges relative to long edges increases. The average node degree was used to determine the density of edges in the graph. See Fig. 2 for a typical graph of 36 nodes distributed in four autonomous systems, with an average node degree of 5 and $\alpha = 0.2$.

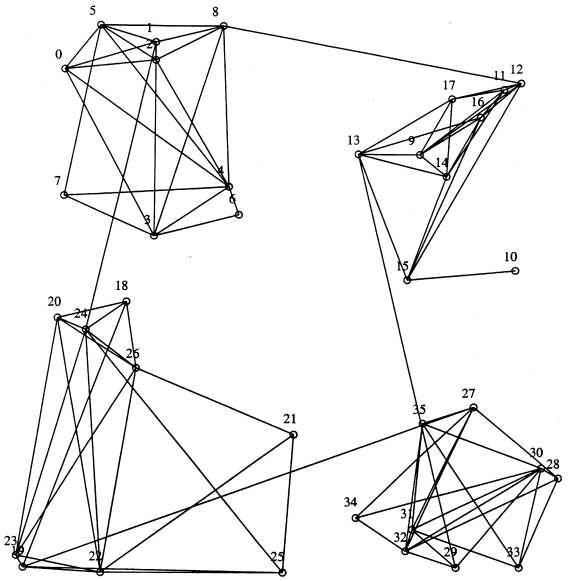


Fig. 2. Graph with 36 nodes and 4 autonomous systems.

Conferences were generated based on Waxman's construction with the first member of the conference being randomly chosen and the other members chosen with the probability

$$P = e^{(-d/L\delta)},$$

where d is the least distance between the new member to be chosen and existing members in the conference. If δ is small (of the order of 0.01) the conference group is called locally dense (all the members will be concentrated in a small region of the first member) while if δ is of the order of 0.5, the conference is called sparse (all members will be fairly far apart). L is again the maximum distance between two nodes in the graph. It was assumed that in a conference any member was equally likely to be a speaker. Note that a conference of a given number of members (usually between 10 and 20) is generated. Shortest path trees were computed for each speaker to the rest of the group, and the Takahashi Matsuyama heuristic was used to create the (unique) CO Steiner tree for the group.

Results are reported for a graph (network) with 800 nodes in 16 autonomous systems for a sparse conference. The graphs had an average node degree of between 5 and 8. The number of members in a conference was taken as 10. We plotted results

as a function of the number of conferences which ranged from 1 to 80. We first describe the results for loading in the network by considering the following metrics computed for both Steiner trees and shortest path trees:

- *Average load on a link*: This is defined as the average of the loads over all the links (edges) in the network. For each conference, each multicast packet that travels over a link increases the load of that link by 1 for the Steiner tree. For shortest path trees, with 10 conference members, there are 10 different shortest path trees (one for each sender in the group) and each shortest path tree increases the load of any link on its tree by 1/10.
- *Maximum load on a link*: This is defined to be the maximum load over all the links in the graph.
- *Standard deviation of the load on a link*: The standard deviation of the load over the n links of the graph.

The average load is shown in Fig. 3. As we have argued, the average load increases with the number of conferences and is significantly more for shortest path trees as the number of conferences becomes large. In a real sense this is a measure of concentration in the network.

It is widely stated that shortest path trees generate a lower traffic concentration as compared with shared trees. One possible meaning might be that the authors believe the maximum load on a link or the standard deviation of load on a link might be lower for shortest path trees. Figs. 4 and 5 show that this is not true as the number of conferences increases.

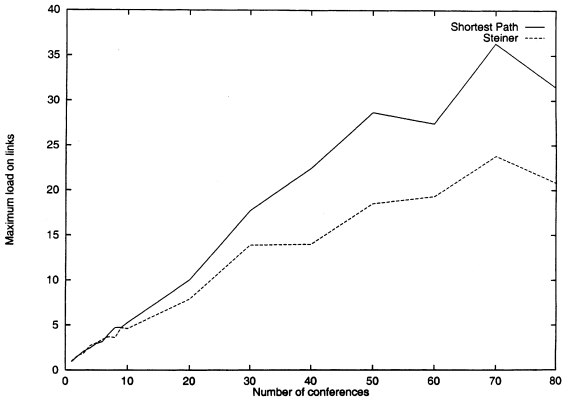


Fig. 4. The maximum load – sparse conferences.

The argument might hold only for a very small number of conferences. In fact our simulation results do indicate this. Interestingly, however, the 'small number' is really small. Fig. 6 shows the maximum load in more detail for 1 to 10 conferences and even here Steiner trees quickly have lower maximum load performance.

In [4] it was recognized that since the average load of a typical shortest path tree was more than for a shared tree, it might be better to define concentration as the ratio of maximum load to average load. Even here it can be seen that as the number of conferences increases, Steiner trees

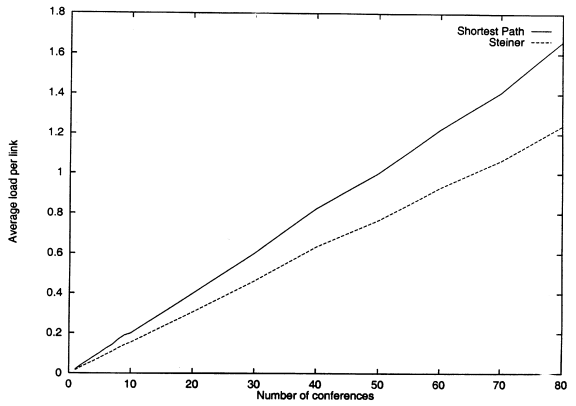


Fig. 3. The average load – sparse conferences.

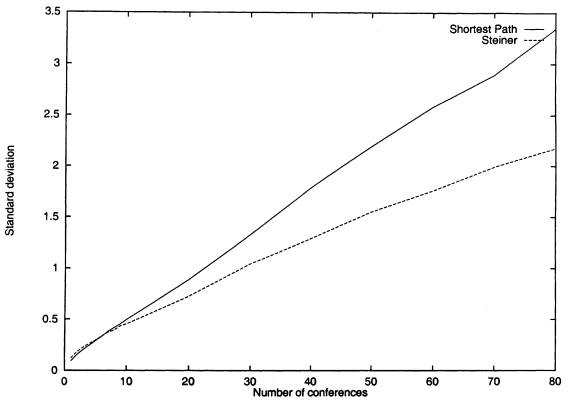


Fig. 5. Standard deviation of load per link – sparse conferences.

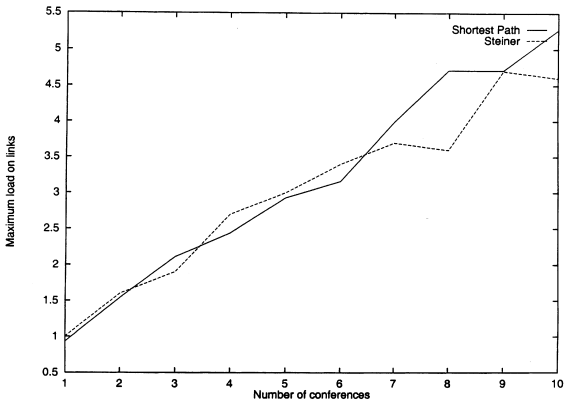


Fig. 6. The maximum load for 1–10 conferences – sparse conferences.

provide better (lower) concentration. In Fig. 7, the trade-off occurs after about 30 conferences.

The results above argue that whatever the notion of concentration is defined to be, as the number of conferences increases, Steiner trees becomes significantly more beneficial as compared with shorted path trees with respect to load on the network.

As indicated previously, it is certainly true that the maximum path delay for a Steiner tree would be greater than for a shortest path tree. This delay is the maximum path length between two nodes in the Steiner tree and the maximum path length from the sender to a receiver over all possible senders in the shortest path trees for the conference. Since this result does not depend on the number of conferences (all results are averaged over many simulation runs) we plot the maximum delay as a function of the number of members in a conference. See Fig. 8.

Note that the maximum delay for a Steiner tree for our detailed study of a conference with 10 members is about 1.5 times the maximum delay for a shortest path tree. This result is consistent with other such reports in the literature [4,12,14]. Further results relating to locally dense conferences and directed graphs can be found in [9]. We now turn to a description of the CSM protocol design.

3. The conference Steiner multicast protocol

The architecture of CSM is designed for use in an environment where the discussion group typically has more than one active speaker and the speaker is also a listener, although it is also appropriate for the situation of only one speaker and the rest listeners. The protocol has been designed so that it can scale to use in a network with a large number of such discussion groups. The scaling is possible because the functions of determining the admission policy and the shared tree for a particular session are distributed throughout the network. A basic aspect of the protocol is that members of the group can easily join or leave the discussion group. This feature of the protocol will be particularly exploited when we discuss the extensions needed for mobile CSM in Section 5. The protocol is most appropriate in support of sparse

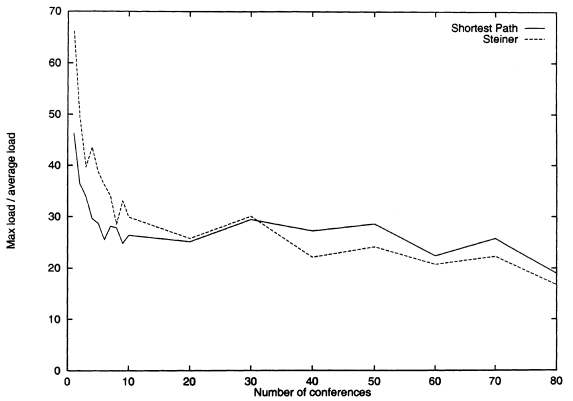


Fig. 7. Maximum load/average load – sparse conferences.

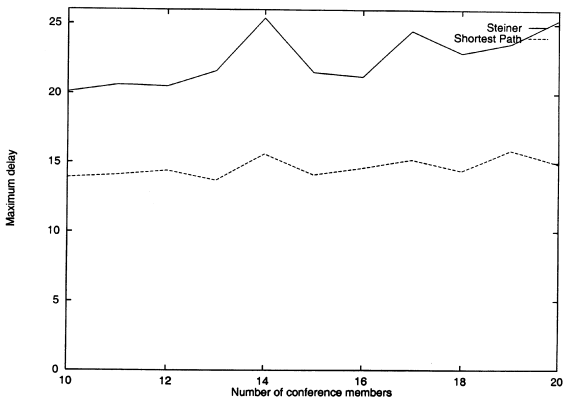


Fig. 8. End-to-end delay – sparse conferences.

groups, although we believe that the protocol could also be used for dense groups.

The protocol provides a low level of security through its registration mechanism that can filter members who are allowed to join the discussion group. The security that is provided is of two types: first, admission control permits refusing of join requests and secondly, and more importantly, routing tables can only be updated by request from other routers, thus preventing applications from causing havoc by unconstrainted joins or join requests. Of course, application level security mechanisms should still be used for privacy and higher level security.

A key aspect of the protocol is that it has an adaptive component that periodically determines how much the current multicast tree differs from the CO Steiner tree and provides a mechanism to switch to a more efficient multicast tree. See [1,3] and also more recently [10]. For example, if the current tree was 20% more costly than the CO Steiner tree, it might be preferable to switch. This feature of switching to another tree is done asynchronously with respect to the dynamic aspects of group membership modification due to the joining and leaving of members, and provides highly reliable delivery of packets during the critical transition phase using the Dualcast protocol [3]. The use of Dualcast during the switch allows a seamless transition to a new tree by multicasting on two trees simultaneously. In the context of mobile networks, a notion of multicasting on redundant links is used to effect handoffs between Mobile Support Stations. This can also be viewed as using a version of the Dualcast protocol.

In this section we discuss the architecture of the basic CSM protocol without the mobility aspects. The CSM protocol has three major phases: (1) *initialization*, in which a new discussion group is registered and a CO Steiner tree or other shared tree is determined for that group; (2) *ongoing operations*, in which members of the group send multicast messages to other members of the group, and during which time the membership of the groups changes as members join and leave the group; and (3) *switching*, in which a new shared tree is determined to be needed and dualcast is used to effect a fault-tolerant transition to a new tree.

Before discussing the details of the protocol, we present some assumptions on the network environment and the notation that we shall use.

Assumptions

1. We describe CSM for an autonomous system where the underlying unicast routing protocol is OSPF. Inter-domain operation can be supported using BGP. We expect that the limitations of CSM due to its reliance on an underlying link state protocol [30] will be less important as an increasing number of domains support OSPF.
2. The Msbone is implemented by having cooperating routers running a multicast Steiner routing daemon *msrouted* that implements the CSM protocol. The *msrouteds* are interconnected using tunnels just as the *mrouteds* in MBone. The *msrouteds* are assumed to have the complete topology of the autonomous system just as in OSPF.
3. The cooperating router over Msbone could actually run any shared tree algorithm using AAR based on input from the session initiator, but for the rest of this paper, unless otherwise indicated, we assume that the router computes a CO Steiner tree. We use the term multicast router or simply router for a cooperating router over the Msbone that is running *msrouted*.
4. We assume that there is some support at the application-layer to filter out duplicate packets. For many applications that use a higher level protocol such as RTP [36], this support is automatically included.
5. We assume that there is an application level daemon running in each area that is a multicast sessions directory (MSD) server. This application level daemon is accessible to anyone wishing to determine possible multicast sessions, and has the flavor of session directory (sdr) in Mbone [23]. This MSD server could be built in a decentralized hierarchical manner, but we do not address this issue in this paper.

Notation

1. A discussion group G consists of a set of n applications involved in the multicast session.

Each application can send/receive data multicast messages to/from all other applications in the group. The ith application is A_i , its corresponding host node is H_i and its designated multicast router is DR_i . The CO Steiner tree of our protocol will be a spanning tree covering the multiset $\{DR_1, \ldots, DR_n\}$. This tree will be termed *Tree*(*G*). The DR runs *msrouted*, and is on the same subnet as the host. An application will be assumed to be known by an *Application ID* that incorporates information such as the application's email address and UDP/TCP port number.

2. A multicast group becomes 'operational' once it has registered with its area MSD server. The MSD server maintains information on each group such as the following:
   (a) Multicast IP address and port number,
   (b) Type of group,
   (c) Membership of the group,
   (d) Description of the multicast session.

   The information entry for group *G* in MSD is termed Entry. We discuss the specific components of *Entry* when discussing registration.
3. Corresponding to each registered group *G*, some application, say A_1, initiates the registration process of the group. In the implementation section we shall see that this is done by a user running a software interface tool called *umsd*. The first action of creating a group is actually to start up a special application that executes the registration process and that will be responsible for authentication of new group members if the group is of a certain type (*Closed* group). This special application is termed *Gatekeeper Application* (*GA*) and usually runs on the same local area network (or even the same host) as A_1. For simplicity, we shall identify this gatekeeper application with A_1, although it can actually be a distinct application from the initiator. See the implementation section for more details. The Designated Router (DR) for *GA* is important and we term this router *Gatekeeper Router* (*GR*). The gatekeeper application is specific to a group *G* and hence a *GA* should be ideally represented as *GA*(*G*) and the corresponding *GR* should be represented as *GR*(*G*). However, the terms *GA* and *GR* will be used instead of the terms *GA*(*G*) and *GR*(*G*) unless it is necessary to do so.
4. The logical distinguishable entities in our architecture that can send and receive messages are thus *Application* (*A*), *Router* (*R*), *MSD*, *Designated Router* (*DR*), *Gatekeeper Application* (*GA*), and *Gatekeeper Router* (*GR*). For supporting mobile applications we will use one additional entity, a *MobileSupportStation* (*Mss*).
5. As a matter of convenience, the message primitives exchanged between the logical entities are named according to a convention. Each primitive is of the form *Source_Destination_Message* (*parameters*) followed by *request*/*reply* where *Source* is the sending entity and *Destination* is the target entity. *Message* is the name of a message, and *parameters* are specific to a given message. *Request* typically represents a request for a specific service, while *reply* is the response corresponding to a given request. The *Source*/*Destination* part of a message, or the *request*/*reply* part is omitted when there is no ambiguity based on the context. For instance, if the message is *A_GA_Join_sparse* (A_j, DR_j), then the message is from an application *A* (source) to the Gatekeeper Application *GA* (destination), the name of the message is *Join_sparse* and the parameters are A_j and DR_j.

The notation discussed above is summarized in Table 1.

3.1. Registration

Before a group can become operational, it is necessary for information about the group session to become generally available. As discussed previously, we assume this is done through an MSD

Table 1
Notation

A_i	ith application belonging to group G
H_i	Host machine running A_i
DR_i	Designated Router for H_i
T_i	Traffic descriptor corresponding to A_i
GA	Application registering a session for group *G*
GR	Multicast router serving *GA*
MSD	Multicast Session Directory

server that can be queried (using the interface tool *umsd*) to determine the available multicast sessions and their characteristics. We shall further assume that there is an MSD server in each domain (called *area MSD*) and that these servers communicate with one another to insure that information about multicast sessions is current and available to all potential participants.

Assume that an application wishes to register a potential group G with its area MSD. Because it is the initiator of a discussion group, this application is *GA* and its designated router is *GR*. *GR* plays a crucial role in the CSM protocol. The gatekeeper application contacts MSD by sending a GA_MSD_Register request message containing the IP address of *GR* and also containing the following information:

1. *Type*(*G*): *Type of group*. A group can be either open or closed. An open group is one where any one can join without any permissions from the gatekeeper. A closed group is one where the *GA* is responsible for admitting new participants. A group can also be *sparse* or *dense*.
2. *Membership*(*G*): *Membership of the group*. The membership of the group consists of the initial set of applications that will be involved in the discussion group.
3. *Description*(*G*): *Description of group*. This information might be needed to allow potential participants to decide whether or not they want to join a session.

On receiving the Register request message, MSD replies by adding the group to its list of discussion groups (creating an *Entry*) and sending a MSD GA Register reply message back to the gatekeeper application. The MSD server also determines *Address*(*G*) which is a multicast IP address and port at which the group will operate. Thus, an *Entry* contains at least the following information: *Address*(*G*), *Type*(*G*), *Membership*(*G*), *Description*(*G*), *GA*(*G*), and *GR*(*G*). An entry in a typical MSD table is shown in Table 2.

Once the *GA* receives the *Register reply* message from MSD, it sends a GA_GR_Register request message to *GR* indicating that *GR* will be responsible for computing a CO Steiner tree for the group.

Table 2
MSD table

Entry[*i*]	*Address*(*G*) = ⟨Class-D IP Address, Port Number⟩
	Type(*G*) = Closed, Sparse
	Membership(*G*) = $\{A_i\}$
	Description(*G*) = "Discussion on Stock Prices"
	GA(*G*) = ⟨Host IP Address, port number⟩
	GR(*G*) = ⟨Router IP Address, port number⟩

Note that using application assisted routing, GA could also send a code-number (the AAR information provided by the gatekeeper application) to GR indicating the type of shared tree to create. This code-number will be used by GR to execute the appropriate routing algorithm for the group. A specific AAR code-number might also include additional information such as expected traffic or bandwidth limitations.

To review then, the message primitives for registration are:

1. *Register request/reply* (between GA and MSD) and
2. *Register request/reply* (between GA and GR).

3.2. The multicast CO Steiner tree

Given the set of *DRs* associated with the applications of a group, the gatekeeper router will generate the corresponding CO Steiner tree.

As discussed previously, we assume that the *GR* has the complete topology of its domain and is thus able to compute a CO Steiner tree using a heuristic such as Takahashi Matsuyama. The input to this algorithm is simply all the multicast routers in the domain and the associated link information, similar to the type of information that a protocol such as OSPF requires, plus the set of *DRs*.

Using the AAR code, an approximate Steiner minimal tree heuristic might also use the traffic information previously defined to help in defining an even more appropriate tree. We have not done a detailed study of the use of such traffic information, but we believe that any of the heuristics can be improved by using this information. The resulting spanning tree *Tree*(*G*) (when using the default shared tree algorithm) is a CO Steiner tree

that uses a common set of links over which to transmit packets. See Fig. 9.

In our implementation, the Takahashi Matsuyama heuristic is used to compute the CO Steiner tree. The Takahashi Matsuyama heuristic works as follows. Start with an arbitrary *DR* and its trivial (partial) spanning tree. Iteratively choose the closest *DR* to the current tree from the remaining set of *DRs* and update the partial spanning tree by adding the path from this *DR* to the tree. Note that the path may contain routers which are not *DRs*. These routers are needed to interconnect the *DRs* in the Steiner tree. The set of routers that constitute the CO Steiner tree for group *G* is referred to as *Routers*(*G*).

The fact that only the *GR* computes the CO Steiner tree and is responsible for subsequent updates is one important element in the scalability of our protocol. For each group, there will be a *GR*, but any multicast router can be a *GR*. Thus, for a large number of discussion groups, it is expected that the function of the *GR* would be fairly uniformly spread among the multicast routers. We have proposed that the *GR* for a group would be determined as being the closest router to the application that initially set up the group. However, it is possible that this function of load maintenance might not be spread out sufficiently among available routers. In such a case, it would certainly be possible to implement a slightly more sophisticated algorithm at the *GA* to choose some other *GR* if necessary. We have not investigated this load balancing issue further.

It should also be observed that calculating the CO Steiner tree is at worst similar in polynomial time complexity to finding all shortest paths in a network and is done infrequently as compared to the ongoing operational aspects of the protocol.

In our simulation study, we computed both CO Steiner trees and shortest path trees and the computation times were comparable.

The *GR* now sends *io-set* information [1] (discussed in the next section) to each router in *Routers*(*G*) using the *GR_R_Ioset_distribution* primitive. This *io-set* information is different for each router in *Routers*(*G*) and is unicast to *Routers*(*G*). We assume that this control information can be sent in a reliable manner, either by using TCP (as in our implementation) or by sending the information repeatedly.

Since the *io-set* information will be sent out in parallel to all the routers in *Routers*(*G*), the time to do this distribution will be the maximum one-way delay between the *GR* and a router in *Routers*(*G*). Typical one-way coast-to-coast delay is on the order of hundreds of milliseconds. Thus we believe that the *io-set* distribution time is not something to be concerned about.

3.3. Steady state CSM protocol operation

The steady state CSM protocol operation from the perspective of the *Routers*(*G*) is quite simple. Each router in *Routers*(*G*) has an entry termed the R_{entry} corresponding to each discussion group *G*. R_{entry} contains the IP multicast address for G, an *io-set*, and a *delivery-set*. The *io-set* (for G) is the set of the incoming/outgoing links (called virtual links) to other adjacent routers in *Tree*(*G*). The *delivery-set* is the set of links (called physical links) to hosts containing applications being serviced by the router. If there is a common interface between the io-set and the delivery-set, it is removed from the io-set to prevent looping. See Table 3.

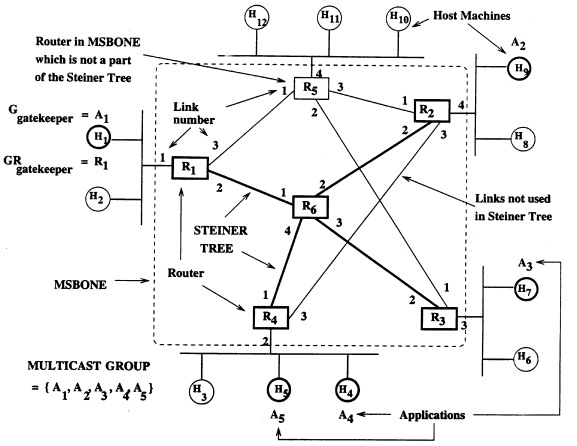


Fig. 9. Steiner tree and related terms.

[1] Can be thought of as routing table entries.

Table 3
R_{entry} in the routing tables

R_{entry} [1]	*Class-D address*	*io-set*	*delivery-set*
.	.	.	.
.	.	.	.
.	.	.	.
R_{entry} [*n*]	*Class-D Address*	*io-set*	*delivery-set*

As an example, the *io-set* for router R_1 in Fig. 9 is {2}, while *its delivery-set* is {1}. Similarly, the *io-set* for router R_6 in Fig. 9 is {1, 2, 3, 4}, while its *delivery-set* is empty.

Any IP packet arriving on an *io link* with the IP multicast address of G is simply forwarded on all the other *io links*. For instance, if a packet arrives for group G at router R_6 through *io link* 1, it will forward the packet on *io links* 2, 3 and 4 (Fig. 9). Note that if the router is a leaf node of the Steiner tree (that is, the *io-set* has only one element), the packet is no longer forwarded to other routers.

If the router is a *DR* (that is, the *delivery-set* is not empty), then it must deliver the packet to one or more hosts connected to this router by using the delivery set information, for final delivery to the applications it is directly serving (i.e., for which it is the closest multicast router). For example, when R_2 receives a packet through *io link* 2, it forwards the packet to application A_2 running on host H_9 (Fig. 9). We reiterate that only *GR* is responsible for computing the CO Steiner tree for G and for performing any updates. The *Routers*(G) operate in an extremely simple manner and maintain only minimal information. This contributes to the scalability of CSM.

3.4. Joining the discussion session

Suppose a new application, A_j, on host H_j wishes to join a discussion group. A_j first queries its area MSD server to determine if there is a discussion group of interest by scanning the entries in MSD. Assume that A_j is interested in joining discussion group G. A_j then obtains the relevant *Entry* from MSD. It is now necessary for A_j to join the discussion session G by having its nearest multicast router, DR_j, join the current shared tree via a path to a router, R_g, that is already a member of the *Tree*(G).

We outline two methods, Sparse Set Join and Dense Set Join, that will allow A_j to have DR_j join the shared tree for G by connecting to R_g. We shall assume that the join method is indicated in the *Entry* of MSD (for example, through the type field) and depends on the membership and type of the discussion group. Assume furthermore, that Dense Set Join is only used for Open type groups. We address the issue of how this method is determined later.

Our Sparse Set Join proposes that *GR* simply compute the shortest path from the new joining node to any node in the current shared tree. Note that this computation is also used in the Takahashi Matsuyama heuristic and in fact would generally require only a lookup as *GR* keeps track of all shortest paths between nodes. In [43], Waxman proposed a parametrized weighted greedy algorithm to determine a path for joining a new node to an existing source based tree. The algorithm minimized the following function over all nodes v in the existing tree:

$$W(v) = (1 - w) * \mathit{distance}(\mathit{newnode}, v) + w * \mathit{distance}(v, \mathit{source})$$

where the parameter w ranges from 0 to 0.5. Our Sparse Set Join heuristic is identical to this with $w = 0$.

Our Dense Set Join simply connects to the existing tree by trying the shortest path to the *GR* until a node of the current tree is hit. Note that this is like the other extreme of the weighted greedy heuristic ($w = 0.5$) or joining in naive multicast routing [14] (where one determines the shortest path from the source) with *GR* playing the role of source, but is actually more similar to the join in CBT (find the shortest path to the core router as target) because it is receiver initiated with *GR* as the target.

There are many interesting join mechanisms that have been proposed for dynamically adding nodes to existing trees or for dynamically modifying Steiner trees on-line. See for instance Kadire's paper [25] on his Geographic Spread Dynamic Multicast, or work on dynamic Steiner tree problems [8,22]. Using AAR, we could, in fact, choose any of these algorithms for dynami-

cally adding (or deleting) nodes to the current tree.

Sparse Set Join (Fig. 10)

- A_j sends a A_GA_Join_sparse (A_j, DR_j) request message to *GA*.
- *GA* checks whether A_j is authorized to join group *G*, and if so, it sends a GA_GR_Compute_src_route(DR_j) request message to *GR*.
- *GA* sends back a response to A_j using a GA_A_Join_sparse (yes/no) reply message.
- If the reply is 'yes', *GR* computes the shortest path to a current multicast router in the *Tree*(*G*), and updates the *io-sets* of the relevant routers. For example, in Fig. 10, the GR (router R_2), informs router R_8 that its neighboring routers in *Tree*(*G*) are R_7 and R_3. Router R_3 will translate this information into its local *io-set* link information {1, 3}.

The primitive messages that need to be implemented for a Sparse Set Join are:

1. *Join_sparse request/reply* (between an application and GA),
2. *Compute_src_route request/reply* (between GA and GR), and
3. *Ioset_distribution* (from GR to R, no reply necessary).

Dense Set Join. We do not present the Dense Set Join algorithm in detail as it is straightforward. The primitive messages that need to be implemented for a Dense Set Join are:

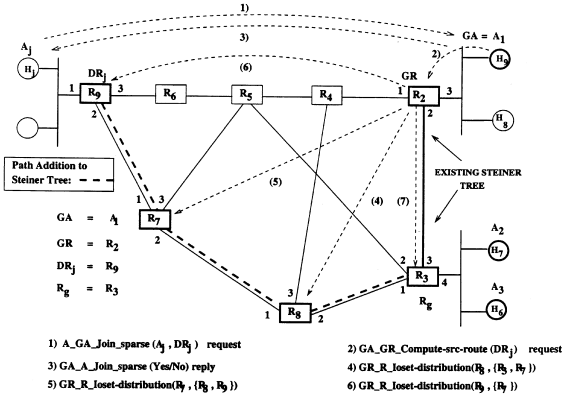


Fig. 10. Sparse join.

1. *Join_dense_request* (from an application to its designated router, no reply necessary),
2. *Join_dense_request* (from router to router, no reply necessary), and
3. *Application_joining_dense* (from router to router, no reply necessary).

3.5. Leaving the discussion session

The method of leaving a discussion group can depend on the exact information that *GA* is maintaining about the discussion group. The exact algorithms for the leave could also depend on the dynamic tree modification algorithm being used. Although we have not discussed the details, for sparse set operation, *GA* would maintain full information on applications and their closest multicast routers. For dense mode operation, however, *GA* need not maintain information on applications. We assume that routers would poll to determine if members on their directly connected LAN have dropped out.

For the sake of brevity, we only describe the primitive messages we have used in implementing leave.

Sparse Set Leave. The primitive messages that need to be implemented for a Sparse Set Leave are:

1. *Leave_sparse request* (from an application or DR to the GA, no reply necessary),
2. *Delete_from_tree sparse request* (from GA to GR, no reply necessary), and
3. *Ioset_distribution* (from GR to R, no reply necessary).

Dense Set Leave. The message primitives needed to implement the Dense Set Leave are:

1. *Leave_dense request* message (between an application and its designated router),
2. *Leave_dense poll* message (between DR and the corresponding application), and
3. *Node_leaving_dense request* message (between routers).

3.6. Computing an updated CO Steiner tree

So far, ongoing operations and joining and leaving the discussion group have been described. It is clear that as this continues, the resulting

shared tree will perhaps no longer be a 'good' tree for the multicast operations.

Recall that the initial Entry in the MSD server could have contained expected traffic information for the group G. The *GR*, however, is clearly equipped to monitor this traffic and could have a much better idea of the traffic situation on an ongoing basis. *GR* can use this and any information from the application using AAR to determine whether the current tree still meets performance requirements.

We did a simple study that shows the benefit of recomputing the CO Steiner tree periodically after a certain number of join/leave events. We assumed that a new member was added to the tree at the nearest node to the current tree. Group membership was modified (a member was added or removed) randomly but so as to approximately maintain the size of the group. We compared the case in which the shared tree was then reset to the CO Steiner tree (based on the heuristic algorithm) after some number of events as compared with a tree that was not reset. The inefficiency measures the percent increased weight of the current shared tree vs. a CO Steiner tree found from the heuristic. In Fig. 11, the upper graph is the result of not resetting the tree (naive), and the lower graph (reconfigured) resets every 10 modifications to the CO Steiner tree. The graphs are averages of 200 runs. Note that periodic transitions to a new tree reduces the average inefficiency from around 24 percent to about 3%. These results are similar to what others have reported [8,9]. In [9], using our sparse join and leave algorithm for the 800-node graph discussed in Section 2 with a conference size of 10, the inefficiency approaches 20% only after about 50 join/leave events. Thus, resetting to the CO Steiner tree need not be an extremely frequent operation.

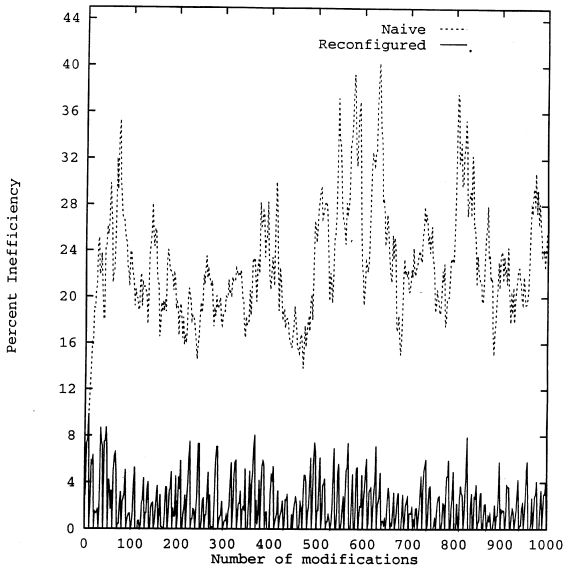


Fig. 11. Effect of periodically computing CO Steiner tree.

Based on the fact that computing the CO Steiner tree can be done fairly fast and that distributing the io-set is also fairly efficient, we believe that switching to a more optimal tree periodically is feasible.

We thus propose that after a certain number of events, *GR* recompute the CO Steiner tree. We call this the *proposed Tree*(G) tree, as compared with the *current Tree*(G). If the difference between the two is above a threshold (inefficiency of 20% for example), then the gatekeeper initiates an algorithm SWITCH to switch from the *current tree* to the *proposed tree*. Notice that the computation of *proposed* trees is done asynchronously with other activities, and can be done in the background. Although joins and leaves could be acknowledged during a switch, we only discuss the case where joins and leaves are queued until the SWITCH is completed.

Assume therefore that the *proposed Tree*(G) is sufficiently better than the *current Tree*(G). The *GR* then initiates the SWITCH tree algorithm described in Section 3.7.

3.7. The Switch Tree algorithm

The Switch Tree algorithm is intended to switch over from the current shared tree to the proposed CO Steiner tree.

The basis of the SWITCH Tree algorithm is that during the transition to the new tree, routers will be forwarding messages on both the current and the proposed tree. This provides a fault-tolerance to packets not being propagated correctly for some reason during the transition period. See Fig. 12.

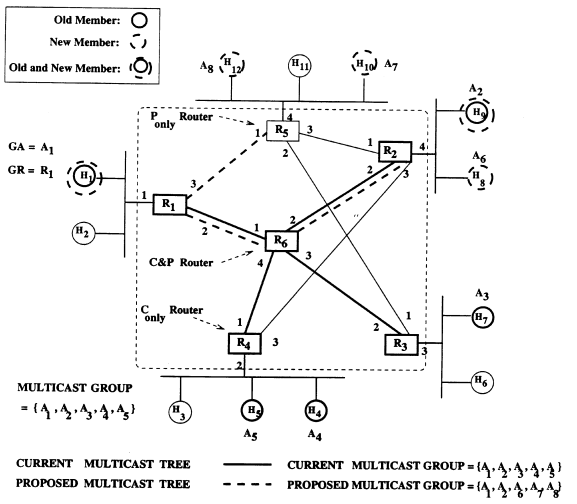


Fig. 12. Switching from current to proposed tree.

Routers on the *current tree* will be termed Current_G routers, and routers on the *proposed* tree will be termed Proposed_G routers. Of course, this set may have a large intersection. Notice that a proposed router that is also a current router might have a different *io-set* in the proposed tree. Call a router that is a current router but will not be a proposed router a C_{only} router. A router that is not a current router but is a proposed router is a P_{only} router. A router that is both a current and a proposed router is a *C* and *P* router. For example, in Fig. 12, R_3 and R_4 are C_{only} routers, R_5 is a P_{only} router, and R_1, R_2, R_6 are *C* and *P* routers. The SWITCH algorithm operates as follows.

1. *GR* sends *io-set* information to all proposed routers using GR_R_Ioset_distribution message as discussed previously in Section 3.2. P_{only} routers create a new R_{entry}, whereas *C* and *P* routers maintain two entries, one reflecting the current *io-set* and the other reflecting the proposed *io-set*. For instance, the *io-set* for R_5, which is a P_{only} router, is {1}. The *io-set* for R_4, which is a C_{only} router, is {1}. R_6, which is a *C* and *P* router, has two *io-sets*: {1,2,3,4} for the current tree and {1,2} for the proposed tree.
2. *GR* sends a GR_R_Start_switch message to routers associated with group *G*. Note that these will be both current and proposed routers. For reliability, this message is sent via TCP.
3. On receiving a GR_R_Start_switch message for *G* over an incoming link, a C_{only} and a P_{only} router propagates the message as usual on all other *io links*. The first *C* and *P* router getting a data packet needs to duplicate the packet and propagate it on each of the two trees by checking each of the two current and proposed R_{entry}'s and acting on each. It will also mark the data packet with an indication that it should go on the current or proposed tree. Subsequent *C* and *P* routers will simply propagate the packet on the correct tree based on the data packet marking. A simpler scheme, which we implement, does not require data packets to be identified and marked but rather has each *C* and *P* router simply forwarding the packets using both R_{entry}'s. This could potentially create some cycles in the graph during the switching time period but the cycles will be broken anyway after the switching is over and the circulating packets will soon be eliminated using the TTL value of IP packets.
4. After a sufficient time has elapsed such that all routers have received the message, *GR* sends an GR_R_End_switch message to routers of group *G*.
5. On receiving an GR_R_End_switch message, a C_{only} router deletes its *Entry*. A *C* and *P* router deletes the *Entry* corresponding to current tree.

Basically, the period between GR_R_Start_switch and GR_R_End_switch is when multicasts are being carried by both the current tree as well as the proposed tree. Other algorithms are also possible for synchronizing the switching period more closely if accurate global clocks at each router are assumed to exist.

The message primitives needed to implement the switch tree algorithm are:

1. *Ioset_distribution* message (from *GR* to *Routers*(*G*))
2. *Start_switch request* message (from *GR* to *Routers*(*G*))
3. *End_switch request* message (from *GR* to *Routers*(*G*))

4. Gatekeeper failures and inter-domain operations

In this section we briefly outline our approach to handling failures during the operation of the

CSM protocol. We also discuss extensions that are needed to the protocol for inter-domain operations.

4.1. Gatekeeper failures

In order to make the CSM protocol tolerant to faults, we follow the methodology discussed in [19] in which a set of software facilities are provided to help an application raise its level of fault tolerance in terms of the availability and consistency of the data. This approach to fault tolerance is different than simply providing, for instance, a primary and secondary domain name server. In CSM, the role of the gatekeeper (GA and GR) is very important. We advocate using a modified primary site approach to sofware fault tolerance in which the service that is to be made fault tolerant is replicated on one or more backup nodes. Thus we would implement a primary and backups for both GA and GR. The primary periodically checkpoints its state on the backups, and when the primary fails, a backup takes over as the primary. This approach has been successfully used in many practical distributed computing environments using reusable components [20] that provide the needed fault tolerance without extensive progamming effort. The computational overhead for providing this kind of fault-tolerance varies from 1% to 7% [20]. More details on our approach can be found in [2]. Note that the backup, by watching its primary, relieves an application from having to itself try the secondary when the primary fails. The burden of implementing the fault tolerance is thus not on the application.

4.2. Inter-domain operations

Our approach to extending CSM to operate in an inter-domain environment is to duplicate and coordinate all the basic functions on an as needed basis in all the domains that have members of group G. There is a primary domain and a number of secondary domains each with a MSD, GA and GR. The entities in the secondary domains are created *on demand* and they go away if the no members remain in the domain. GR in each secondary domain computes its own tree and grafts that tree to the primary tree through the border router of the domain. Due to space considerations, we refer the interested reader to [2] for additional details.

5. Mobile CSM

As discussed in the introductory sections, one of our goal in designing CSM was to make it fairly easy to develop a multicast protocol that involves applications that are mobile. See also [34]. We now show how our CSM architecture and the primitive messages we have previously defined can be extended to an architecture for mobile CSM (called MCSM in this section). For concreteness of the discussion, imagine a mobile application to be an application that is running on a computer in a car or running on a portable pc that an individual is carrying around.

The logical distinguishable entities that were able to send and receive messages in CSM were *Application*, *Router*, *MSD*, *Designated Router*, *Gatekeeper Application*, and *Gatekeeper Router*. We have one new logical entity in CSM that we term *MobileSupportStation* (*Mss*). An *Mss* is a fixed entity on the network that provides necessary support to a mobile application. An *Mss* can be equated to a Base Station in the cellular architecture. Our protocol related to handoff is based on the assumption that there is substantial overlapping between adjacent cells [16], such that a mobile may be assumed to have adequate communication with more than one base station during transition from an old to a new base station.

The Mobile CSM protocol will be described in reference to the architecture given in Fig. 13, in which there are several mobile support stations, connected to a fixed packet-switched backbone network. Each cell (represented by a hexagon) has a *Mss* which acts as the access point for the mobiles in the cell. In Fig. 13, M_j is a mobile host running an application A_j and the mobile host accesses the backbone network through the mobile support station Mss_1. As long as the mobile host M_j remains in the cell of Mss_1, it interacts with Mss_1 exactly in the same way as host H_j does with its designated router DR_j in the non-mobile con-

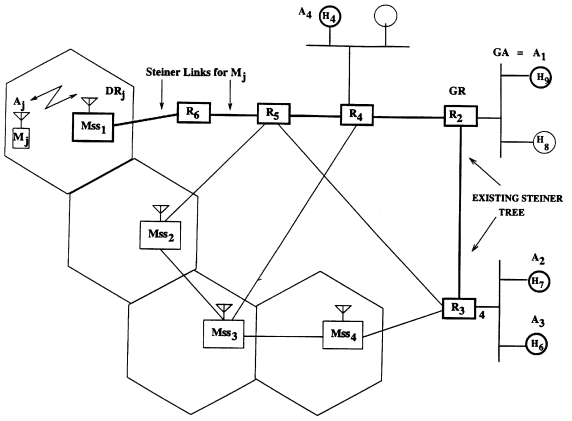


Fig. 13. Architecture for mobile CSM.

ferencing mode. As the mobile M_j starts to move away from Mss_1, there is a period, when it has adequate communication [16] with both Mss_1 and say, Mss_2, which is a neighbor of Mss_1 (Fig. 14).

During this handoff period, M_j will inform the new mobile support station Mss_2 that it wants to join the multicast session, and that control information will be propagated through the routers resulting in the addition of a new branch from R_5 to Mss_2 in the existing shared tree (Fig. 14). Note that during this handoff period, the mobile M_j has two paths, one through Mss_1 and the other through Mss_2, for receiving (sending) packets from (to) the multicast session in progress. Thus the notion of redundancy is used for reliable delivery of packets to the mobile station.

After the mobile M_j moves away from Mss_1, and starts to communicate only through Mss_2, the control information is propagated through the routers, resulting in the removal of the branch R_5–R_6–Mss_1 (Fig. 15).

Thus the multicast tree gets altered dynamically as the mobiles move in and out of cells. In particular during handoff, the existing tree is augmented with new branches to provide redundancy, and after handoff is over, unnecessary branches are pruned off the multicast tree.

Additional details of the Mobile CSM architectue can be found in [2]. Although the logical design is straightforward using our approach, without an implementation and experimentation, it is not possible to assess overhead and efficiency issues. We expect that the level of dynamicity of members leaving and joining will affect the efficiency of Mobile CSM. For example, if mobiles are very highly mobile, it could be wise to anticipate this and prepare all neighboring cells for receiving the multicast by setting up a tree that pretends that there are mobiles in all adjacent cells.

6. Implementation

Using the CSM design, a virtual network called *Msbone* was developed for use in the Internet. In

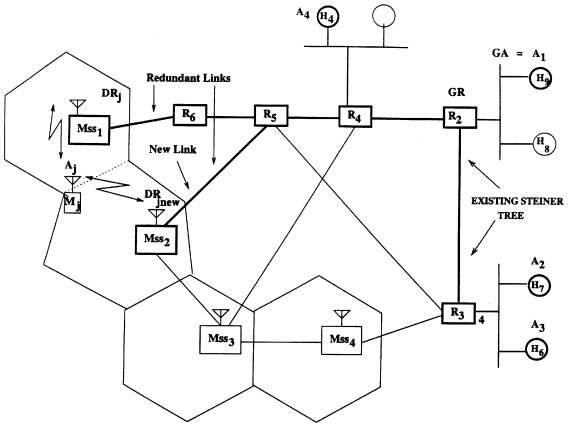


Fig. 14. Redundant connections during handoff.

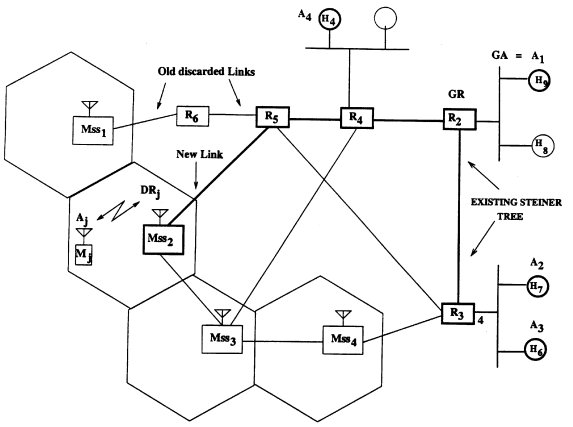


Fig. 15. Switching to a new cell after handoff.

the Msbone, group admission and routing is handled in a manner described by the CSM protocol. The Msbone supports both CSM style and traditional multicast applications and can interoperate with the Mbone. In our implementation, we developed two major software components: (1) *umsd* (user multicast session directory) and (2) *msrouted* (multicast Steiner routing daemon). The umsd is the user's interface to CSM and permits users to create (and delete) group sessions and join and leave sessions, and to launch (multicast) applications such as audio/video conferencing tools. The msrouted implements the routing, communications, and GR functions of CSM. In this section, we discuss our CSM implementation and discuss how multicast applications can be implemented over CSM. Since Msbone has been implemented using UDP tunnels and corporate firewalls do not let UDP traffic in or out, our experimentation has so far been limited to within the Lucent intranet. Hence performance comparison with MBone has not yet been done.

To form the virtual network, each LAN selects one host to serve as the designated "multicast router". This host runs msrouted. Msrouted is responsible for forwarding multicast packets to and from the LAN. Msrouted's exchange multicast packets by encapsulating them within unicast packets. Two msrouted's which exchange packet via this encapsulation method are said to be connected by a tunnel. The collection of LANs, msrouted's, and tunnels constitute the Msbone. Note that in our implementation, there is a one-to-one identification with a CSM capable LAN and a msrouted. See Fig. 16.

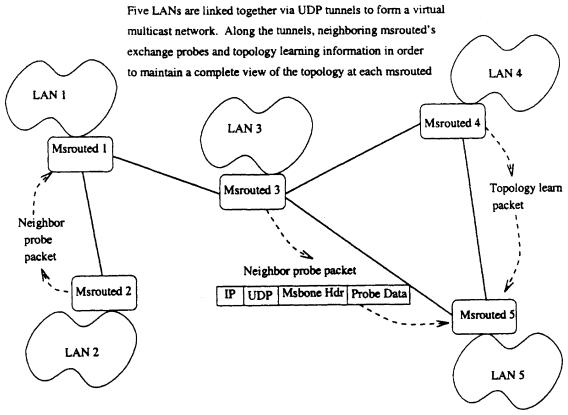


Fig. 16. Msbone architecture.

6.1. Msbone applications

Unlike other multicast schemes, the CSM design allows a multicast application to assert a great deal of influence over the group via the Gatekeeper Application concept. Applications designed for CSM can utilize this additional influence for improved operation. However, a large base of applications already exist that do not incorporate CSM style concepts. Msbone was designed to support both types of applications.

In the CSM model, there is the notion of a Gatekeeper Application, GA, that controls admission to the group and influences the group's routing. Typically the group's initiator acts as or initiates the GA that registers the group with MSD. The GA's (unicast) address is advertised along with the group address and to join a group, a user sends a request (via unicast) to the GA (see Fig. 17). If the request is approved, the GA must inform the Gatekeeper Router so a route will be established to the new member (see Fig. 18).

While the gatekeeper concept adds to the complexity of creating and joining a group, it also provides substantial advantages. The gatekeeper

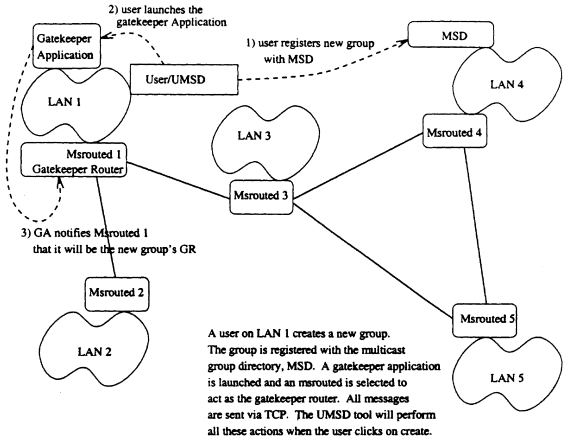


Fig. 17. Registration of a new session.

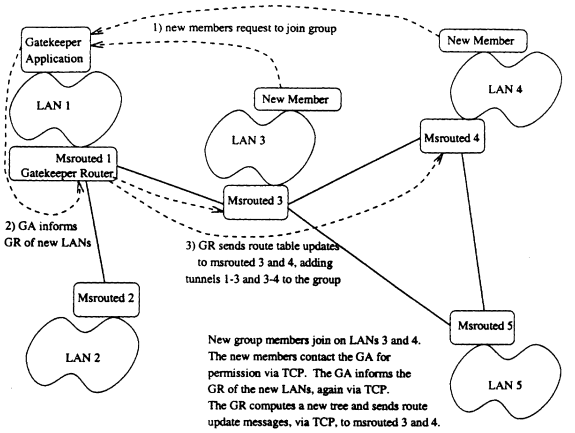


Fig. 18. Joining an existing session.

application can assert control over group membership. In many applications, there is a desire to know and perhaps restrict the group membership. With the GA concept, membership control is managed cooperatively within the network and the application can be assured that the routes are being computed based only on the valid members. The GA can also control when new members are allowed to join the group by delaying when the join is sent to the network. This allows the multicast application to enforce a rule that only allows new members to enter at specified times, if so desired. In addition to membership control, the GA can influence routing decisions. At any time the GA can instruct the network to recompute the routes or even switch to an entirely new algorithm for computing routes. In a multicast for two speakers, the GA could first optimize the routes for the initial speaker and later change the routes to optimize for the second speaker using AAR.

This GA functionality can be incorporated into the multicast application itself, it can be handled by a separate GA process, or it can be assigned to one of several default Gatekeeper Applications that run at well known addresses throughout the network.

6.1.1. Applications with built-in gatekeepers

An application designer could chose to build the Gatekeeper Application functionality directly into the multicast application code. A simple command line option could indicate the application is to be run in 'gatekeeper mode'. When the application is run in non-gatekeeper mode, it could automatically generate a join request and send it to the 'gatekeeper mode' application. This join procedure can occur without the user's knowledge or could pop up windows asking for information such as passwords to complete the join. Some applications naturally have the concept of a gatekeeper and these concepts are not only easy for the application designer to implement, but also highly desirable. For example, an application to hold committee meetings may naturally want to designate one user as the chairperson and the chairperson's application should allow control over who attends the meeting.

6.1.2. Separate gatekeeper application processes

Some application designers may find it desirable to locate the GA functions in a separate process or program. Typically the group's initiator would start this GA process. The application could be designed so it contacts the GA process automatically or the user might first have to run a join process or wrapper that only starts the main application after the join has succeeded. The GA process can define its own packet formats and control scheme or it could make use of Msbone GA control packet formats. The Msbone defines packet formats for an *open*, *closed*, and *secure* admission control scheme. Open format allows any host to join, closed restricts membership to a set of hosts, and secure requires a password to join the group. No restrictions are placed on how admission control is to be handled. The only requirement is that the GA and potential members have some agreed upon message format for joining the group.

6.1.3. Support for standard applications

Applications such as vat, vic, wb, and nevot [23] have become network standards for multicast conferences. These type of applications do not support the concept of a GA, but can still be used on the Msbone. Several default Gatekeeper Applications can run at well known sites and can

serve as the GA for any application that does not wish to provide its own GA. This allows non-CSM applications to operate on the Msbone and benefit from Msbone admission control. An application launching program is used to first contact a default GA (or first create the group through the GA). Once the GA has been contacted and a join succeeds, the launching program can exit and the application is run as it normally would be. When creating the group, the default GA can be instructed to use *open*, *closed*, or *secure* admission control. This allows tools such as vat to make use of the Msbone's admission control if desired.

Mbone users will be accustomed to launching multicast applications via the sd or sdr tool. Msbone has a similar tool, *umsd* for User Multicast Session Directory (see Fig. 17). With umsd a user can create a group and join a group in a fashion nearly identical to that of Mbone and sd. The user need not even be aware of the GA concept. The umsd tool handles all the details relating to the GA and then starts the multicast application.

6.1.4. Designing a gatekeeper application

Application designers that want to design their own GA may be uncomfortable having to deal with the network level multicast issues. Admission control is something the application designer is likely to be aware of, but multicast routing is often not an area of expertise for the application designer. The GA is responsible for communicating membership to the network (via the Gatekeeper Router concept of CSM) and application designers may find this troublesome.

To solve this problem, Msbone provides designers with a network interface library. Adding a member to the group is as simple as calling a function called 'addmember'. Various routing algorithms are listed by number and selected via a 'recomputetree' function. Much the way a socket is controlled by simple flags, multicast routing can also be controlled by well-defined functions and option flags. The application designer need not understand the complexities of the multicast routing problems and need only call functions from the multicast control library. We omit the details of this control library.

6.2. Msbone routing

Since modifying existing routers is often not practical, Msbone uses host software to perform routing tasks. A LAN selects one host to run msrouted. Each msrouted is configured to exchange packets with a fixed set of remote msrouted's. These remote sites are specified in a configuration file that is read when msrouted is started.

The current release of msrouted runs in user level space without requiring any special permissions. Better performance can be achieved by running with superuser privileges, but the prototype was chosen to run without this for easier initial deployment. Readers familiar with encapsulation methods should note that this requires that the msrouted tunnels encapsulate multicast packets in IP and UDP headers rather than just IP headers. Changing to only IP encapsulation (and thus also superuser privileges) is primarily a technicality and presents no fundamental changes to the Msbone design.

Msrouted is logically divided into four modules: (1) Tunnel Message Handler, (2) Control Message Handler, (3) Gatekeeper Router Module, and (4) Topology Learning Module. See Fig. 19.

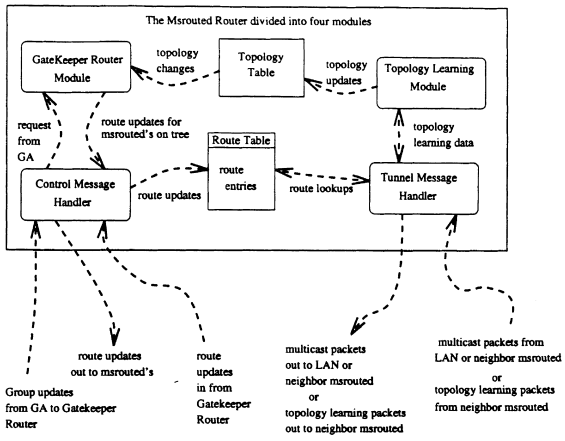


Fig. 19. Msrouted architecture.

6.2.1. Tunnel message handler

The Tunnel Message Handler focuses on reading and writing local multicast packets and exchanging packets with other msrouted's. Once an application has joined a group via the GA and a route has been constructed, the application simply writes a multicast packet onto the LAN. The multicast packet is received by all local group members as well as msrouted. Msrouted looks up the group's entry in the route table and determines which adjacent msrouted's should receive the packet. The multicast packet is encapsulated within an IP and UDP header and sent (unicast) to the appropriate adjacent msrouteds. When an encapsulated packet arrives at an msrouted, msrouted checks the multicast address of the encapsulated packet and forwards it to the appropriate (adjacent) msrouteds. If there are local group members, msrouted decapsulates the packet and multicasts it locally on the network. Any local members receive this multicast as if they were on the same LAN as the original sender. See Fig. 20.

6.2.2. Control message handler

The Control Message Handler module of msrouted will receive route updates from Gatekeeper Routers. In CSM terminology, these route updates are referred to as *io-sets*. An *io-set* is associated with a particular group and tells the msrouted which neighboring msrouteds are part of a group's multicast routing tree. Route entries may be explicitly cleared by receiving a NULL *io-set* for a group or the entry will expire if refresh messages are not periodically received. The control unit is responsible for processing *io-sets*, maintaining the route table, and periodically expiring old entries.

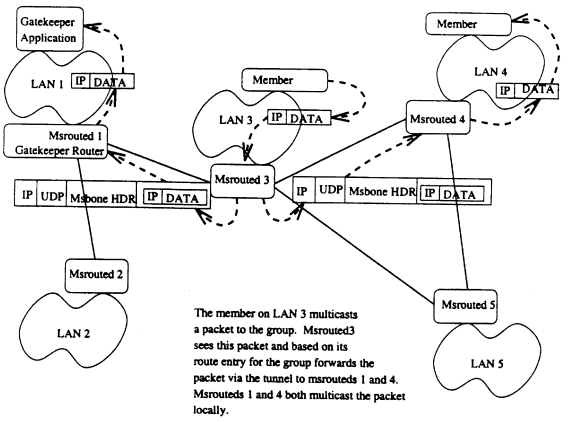


Fig. 20. Multicast packet forwarding in Msbone.

6.2.3. Gatekeeper router module

Recall that CSM designates a specific router (per group) known as the Gatekeeper Router (*GR*) that computes and distributes the route entries (*io-sets*) for that group. As discussed previously, for scalability, any msrouted may act as a *GR* and may serve many groups simultaneously. The choice of gatekeeper router is typically the nearest router to the *GA* that registers the group, but can really be any router in the Msbone. The Gatekeeper Router Module performs these functions. It receives group membership updates from the GA and computes a route accordingly. This module implements application assisted routing and a number of shared tree heuristics can be used to compute the tree. Each algorithm is associated with a code-number and the GA decides which algorithm number to use. These modules are designed so that additional algorithms can easily be added to msrouted. Adding a new algorithm is as simple as adding the algorithm function name to an included file, recompiling, and linking in the object code for the function.

6.2.4. Topology learning module

Since CSM is based on computing CO Steiner trees, topology information about the network is essential. This fourth msrouted unit is responsible for maintaining topology information. The precise algorithm for doing this is not fundamental to the design and could be replaced by OSPF. All that is required is that the Topology Learning Module provide up-to-date topology information to the Gatekeeper Router Module for computing the shared trees. The Topology Learning Module also notifies the Gatekeeper Router Module in the event of topology changes.

6.3. Interoperation

It is increasingly unlikely that any one multicast algorithm/architecture will gain complete acceptance. Interoperation between algorithms is essential. The current Msbone can interoperate with any other system such as Mbone in the following basic manner. An Msbone 'gateway' runs on a LAN with a non-Msbone router. For example, the Msbone gateway can run on any LAN in the Mbone. The gateway acts as a proxy group member. If any Msbone host joins a Mbone group, the gateway also joins the same group. The non-Msbone algorithm will deliver packets to the gateway just as it would any other host. The gateway then routes the packets onto the Msbone as if it had generated the packet. Packets leave the Msbone by arriving at the Msbone gateway and then being multicast onto the gateway's LAN where the other (non-Msbone) multicast router will see the packets and distribute them as if they had been issued by any local group member. In a similar way, Msbone applications could allow non-Msbone participants to join Msbone sessions.

An initial version of Msbone is operational in the Bell Labs research network, and can be used to create new multicast sessions or join existing sessions in a manner similar to Mbone. Each session can run Mbone applications, namely, vat, vic, and wb [23] at present. Work is ongoing to develop a more complete prototype that can be used for experimental studies.

7. Related protocols

Two main protocol architectures have been developed for dense multicast groups. DVMRP is an extension of a distance vector unicast protocol and MOSPF is an extension of a link state unicast protocol. DVMRP has been successfully deployed over the Mbone. Both of these protocols were designed for multicast routing using source-based trees where the unique sender is the source. The problems with these protocols with respect to scaling and use for sparse groups are well known and have been discussed by several authors. In particular, the mechanism that DVMRP employs to prune and regrow trees is not practical for sparse groups. MOSPF requires that each router must receive and store membership information for each multicast group and requires that each router compute the delivery tree for each group. Thus, the information storage and processing requirements for MOSPF cause difficulty when scaling to a large number of multicast groups. It can be seen that CSM does not have these scaling problems as it uses a shared tree and handles joins and leaves differently from these protocols.

Two major protocol architectures have been proposed for sparse multicast groups. CBT proposes the use of a shared tree centered at a core [5]. CSM is similar to CBT in that it scales in the same manner as CBT and also uses a shared tree for the group multicast. However, CSM precisely defines how the shared tree is set up and modified as necessary by the gatekeeper for the group. In contrast, CBT does not discuss the placement of the core and initially suggests to either use statically configured cores or to have the nearest router to a host act as a core. It should be clear that although the notion of a gatekeeper in CSM is somewhat akin to a core router, the gatekeeper plays the crucial role in CSM of computing the close-to-optimal shared tree and is unlikely to be a 'core router' in the sense of CBT. Furthermore, the fact that the initiator of a group is usually the GA for the group and its DR is the GR for the group implies that any router can become a GR. Therefore, for a large number of multicast groups, the gatekeepers would likely be distributed uniformly across the network of multicast routers, and there is no single router which will become the bottleneck. This is the key to the scalability of CSM.

A second major difference in CSM as compared with CBT is that CSM discusses in detail how to switch to another more optimal tree and provides a detailed method as to how to do so reliably. Other differences include details on how joins and leaves are implemented.

CSM also differs from CBT in its handling of failure recovery. CBT attempts to integrate failure recovery directly with the CBT protocol. In contrast, CSM separates the failure recovery issue into

two parts. Failures of gatekeepers are handled by a mechanism discussed in this paper. Failures of links and nodes, on the other hand, are assumed to be handled by the network of multicast routers in a manner similar to the way failures are currently handled by the Internet routers. (See for instance our implementation of the msrouted Topology Learning Module.) Thus, multicast routers would exchange link state information assuming an OSPF type of protocol was implemented at that level. Finally, CSM defines extensions to multiple administrative domains.

The second major architecture that has been proposed for sparse multicast groups is PIM [12]. PIM describes the use of rendezvous points (RP's) that initially allow the creation of a shared tree. PIM then discusses how to move to a shortest path source based tree if desired. In contrast, CSM starts out with a CO Steiner tree (if an initial set of participants is known) and then continuously evaluates the membership set to see if it is wise to move to another shared tree. CSM precisely defines how the shared tree is initially setup by the gatekeeper and CSM also clearly allows for a choice of shared tree algorithms to be used by a gatekeeper. CSM also differs from PIM in that the multicast routers in CSM have a simpler architecture for ongoing multicast sessions. CSM only uses the multicast address and the *ioset* to determine how to route a packet (it uses a *delivery set* if a locally attached host wants that packet). In contrast PIM requires more processing by the routers, requiring them to maintain information on incoming and outgoing links for each group as well as the RP addresses. Furthermore, additional complexity is entailed by having the routers responsible for switching from a shared tree to a source-based tree. Although PIM describes the use of a shared tree, it is clearly designed to quickly switch to a source-based tree. The premise of CSM is to always be operating on an appropriate shared tree.

PIM is also different from CSM in separating the senders from receivers. Although for fixed sender sources this may be advantageous, there is no simple way in PIM of converting a sender into a receiver in an existing multicast tree. Therefore, PIM may have trouble with applications such as conferencing and chat groups, in which the senders and receivers interchange their roles very often in the middle of a session.

CSM, as contrasted with PIM and CBT supports application assisted routing and clearly discusses extensions that allow it to be used in a mobile environment.

8. Conclusion and future work

In this paper we presented a flexible multicast routing protocol targeted towards conferencing and online discussion groups. We have discussed how this new protocol architecture differs from other architectures previously proposed. Some of the key ideas of the paper are the distributed dynamic computations of the CO Steiner trees for the multicast groups and the support for application assisted routing, the fault-tolerant transition from an old tree to a new tree, extension of the basic protocol to an inter-domain environment, and the introduction of the watchdog concept to make the basic protocol robust in a hostile network environment. The basic protocol primitives can be also be used to support mobile multicasting. Handoffs in mobile CSM are also handled in a novel way as compared to other mobile protocol approaches.

Our current implementation of CSM offers the following features:

- Group admission control in cooperation with the network.
- Application dependent group admission schemes.
- Ability for each group to select its own routing algorithm.
- Application driven recomputation of routes.
- Application control over adjusting routes for new members.
- Ability to easily add new routing algorithms to the system.

The current implementation is local to the Lucent intranet. In future work, we intend to deploy Msbone more broadly and do a detailed performance analysis.

We believe a key lesson learned from the construction of the Msbone is that the application and

network can be integrated in a simple way. This allows the application to gain better performance and prevents situations where the network must guess the application's desires.

Acknowledgements

The authors are grateful to Amritansh Raghav for several early discussions on multicasting.

Appendix A. List of abbreviations used in the paper

Abbreviation	Expansion
AAR	Application-Assisted Routing
BGP	Border Gateway Protocol
CBT	Core-Based Tree
CO	Close-to-optimal
CSM	Conference Steiner Multicast
DR	Designated Router
DVMRP	Distance Vector Multicast Routing Protocol
GA	Gatekeeper Application
GR	Gatekeeper Router
Mbone	Multicast Backbone
MOSPF	Multicast Open Shortest Path First
MSBone	Multicast Steiner Backbone
MSD	Multicast Sessions Directory
MSS	Mobile Support Station
OSPF	Open Shortest Path First
PIM	Protocol Independent Multicast
RTP	Real-Time Protocol

References

[1] S. Aggarwal, S. Paul, A flexible protocol architecture for multi-party conferencing, in: Proceedings of the ICCCN'96, October 1996, pp. 81–91.

[2] S. Aggarwal, S. Paul, D. Massey, D. Calderaru, A flexible protocol architecture for multi-party conferencing: from design to implementation, Technical Memorandum, Lucent Technologies, April 1998.

[3] S. Aggarwal, A. Raghav, DUALCAST: a scheme for reliable multicasting, in: Proceedings of the ICNP'94, October 1994, pp. 15–21.

[4] T. Billhartz, J.B. Cain, E. Farrey-Goudreau, D. Fieg, S.G. Batsell, Performance and resource cost comparisons for the CBT and the PIM multicast routing protocols, IEEE Journal on Selected Areas in Communications 15 (3) (1997) 304–315.

[5] A.J. Ballardie, P.F. Francis, J. Crowcroft, Core based trees, in: Proceedings of the ACM SIGCOMM, 1993.

[6] K. Bharat-Kumar, J.M. Jaffe, Routing to multiple destinations in computer networks, IEEE Transactions on Communications COM-31 (3) (1983) 343–351.

[7] T. Ballardie, R. Perlman, C. Lee, J. Crowcroft, Simple scalable internet multicast, Technical Report, University College, London, April 1999.

[8] F. Bauer, A. Varma, ARIES: A rearrangable inexpensive edge-based on-line Steiner algorithm, IEEE Journal on Selected Areas in Communications 15 (3) (1997) 382–397.

[9] D. Caldararu, Heuristic Steiner trees for networks with large numbers of multicast conferences, M.S. Thesis, Department of Computer Science, SUNY, Binghamton, August 1997.

[10] J. Donahoo, K.L. Calvert, E.W. Zegura, Center selection and migration for wide-area multicast routing, Journal of High-Speed Networks 6 (2) (1997).

[11] S. Deering, Host Extensions for IP multicasting, RFC 1112, May 1988.

[12] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, An architecture for wide-area multicast routing, in: Proceedings of the ACM SIGCOMM, August 1994, pp. 126–135.

[13] S. Deering, D. Cheriton, Multicast routing in datagram internetworks and extended LANs, ACM Transactions on Computer Systems (May 1990) 85–111.

[14] M. Doar, I. Leslie, How bad is naive multicasting, in: Proceedings of the IEEE INFOCOM, 1993, pp. 82–89.

[15] H. Eriksson, Mbone: the multicast backbone, Communications of the ACM (August 1994) 54–60.

[16] L.G. de R. Guedes, M.D. Yacoub, Overlapping cell area in different fading conditions, in: Proceedings of the 1995 IEEE 45th Vehicular Technology Conference, July 1995, pp. 380–383.

[17] S. Hakimi, Steiner's problem in graphs and its applications, Networks 1 (1971) 113–133.

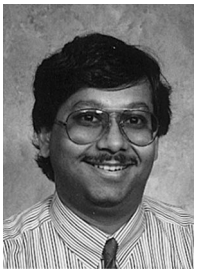
[18] H. Holbrook, D. Cheriton, IP multicast channels: EXPRESS support for large-scale single-source applications, in: Proceedings of the ACM SIGCOMM, August 1999, pp. 65–78.

[19] Y. Huang, C. Kintala, Software implemented fault tolerance: technologies and experience, in: Proceedings of the 23rd International Symposium on Fault Tolerant Computing (FTCS-23), June 1993, pp. 2–9.

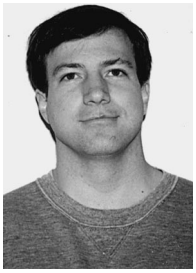
[20] Y. Huang, C. Kintala, Software fault tolerance in application layer, in: M. Lyu (Ed.), Software Fault Tolerance, Wiley, New York, 1995, Ch. 10.

[21] F.K. Hwang, D.S. Richards, Steiner tree problems, Networks 22 (1992) 55–89.
[22] M. Imase, B.M. Waxman, Dynamic Steiner tree problem, SIAM Journal of Discrete Math 4 (3) (1991) 369–384.
[23] V. Jacobson. Multimedia conferencing on the internet, Tutorial 4, ACM SIG-COMM 94, August 1994.
[24] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum, New York, 1972.
[25] J. Kadirire, G. Knight, Comparison of dynamic multicast routing algorithms for wide-area packet-switched (ATM) networks. in: Proceedings of the IEEE INFOCOM, 1995, pp. 212–219.
[26] L. Kou, G. Markowski, L. Berman, A fast algorithm for Steiner trees, Acta Informatica 15 (1981) 141–145.
[27] V.P. Kompella, J.C. Pasquale, G.C. Polyzos, Multicasting for multimedia applications. in: Proceedings of the IEEE INFOCOM, 1992, pp. 2078–2085.
[28] V.P. Kompella, J.C. Pasquale, G.C. Polyzos, Multicast routing for multimedia communication, IEEE/ACM Transactions on Networking 1 (3) (1993) 286–292.
[29] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, M. Handley. The MASC/BGMP architecture for inter-domain multicast routing, in: Proceedings of the ACM SIGCOMM, August 1998, pp. 93–104.
[30] J.J. Garcia-Luna-Aceves, J. Behrens, Distributed, scalable routing based on vectors of link states, IEEE Journal on Selected Areas in Communications 13 (8) (1995) 1383–1395.
[31] L. Lou, K. Makki, An even faster approximation algorithm for the Steiner tree problem in graphs, Congressus Numerantium 59 (1987) 149–154.
[32] J. Moy, Multicast routing extensions for OSPF, Comm. ACM 37 (8) (1994) 61–66.
[33] K. Makki, N. Pissinou, O. Frieder, Efficient solutions to multicast routing in communications networks, Mobile Networks and Applications 1 (1996) 221–232.
[34] C. Perkins, IP Mobility Support', RFC-2002.
[35] V. Rayward-Smith, The computation of nearly minimal Steiner trees in graphs, International Journal of Mathematics and Educational Science and Technology 14 (1983) 15–23.
[36] RTP: A Transport Protocol for Real-Time Applications," Internet Draft, March 1995, ⟨draft-ietf-avc-svc-rtp-07.txt⟩, October, 1995.
[37] H.F. Salama, D.S. Reeves, I. Viniotis, Comparison of multicast routing algorithms for high-speed networks, IBM Technical Report, September 1994.
[38] H. Takahashi, A. Matsuyama, An approximate solution for the Steiner problem in graphs, Mathematics Japonica 6 (1980) 573–577.
[39] D.W. Wall, Mechanisms for broadcast and selective broadcast, Ph.D. Thesis, Stanford University, June 1980.
[40] P. Winter, Steiner problem in networks: a survey, Networks 17 (1987) 129–167.
[41] D.W. Wall, S. Owicki, Construction of centered shortest path trees, Networks 13 (1983) 207–231.
[42] D. Waitzman, C. Partridge, S. Deering, Distance Vector Multicast Routing Protocol, RFC 1075, 1988.
[43] B.M. Waxman, Routing of multipoint connections, IEEE Journal on Selected Areas in Communication, 6–9 December 1988, 1617–1622.
[44] L. Wei, D. Estrin, A comparison of multicast trees and algorithms, CS Department Technical Report, University of Southern California, September 1993.

Sudhir Aggarwal is Head of the Internet Systems Research Department at Bell Laboratories, Lucent Technologies, in Murray Hill, NJ. Prior to his current position, he was professor and chairman of the Computer Science Department at the State University of New York, Binghamton, NY. His interests are in computer networks, design and development of distributed software systems and communication protocols, real-time systems, and distributed interactive simulation. He received the Ph.D. degree from the University of Michigan in Computer and Communication Sciences in 1975, and M.S. and B.S degrees in Mathematics from the University of Michigan and Stanford University in 1971 and 1969, respectively.



Sanjoy Paul (M'92-SM'97) received a B.Tech. degree from Indian Institute of Technology, Kharagpur, India, followed by M.S and Ph.D. degrees from University of Maryland, College Park, in 1988 and 1992 respectively. He joined AT&T Bell Laboratories after graduating from University of Maryland. He is currently a principal investigator and a distinguished member of technical staff in networking software research department in Bell Laboratories. His research interests include caching; multimedia streaming; multicasting; transport, network, and link-layer protocol issues; formal methods; security; and mobile networking. Dr. Paul holds nine US patents, has published widely in international conferences and journals, and has been on the program committees of several IEEE conferences. Dr. Paul served as the guest editor of IEEE Network Magazine Special Issue on Multicasting. He is the co-recipient of 1997 William R. Bennett award from IEEE Communications Society for the best original paper in IEEE/ACM Transactions on Networking in 1996. Dr. Paul is author of the book "Multicasting on the Internet and its Applications", which was published by Kluwer Academic Press in May 1998. He is an adjunct faculty of the Computer Science Department and the Internet Technical Institute of Rutgers University. Dr. Paul is a senior member of IEEE and a voting member of the ACM.



Daniel Massey received a B.A. in Mathematics/Computer Science and an M.A. in applied mathematics from U.C. San Diego. He expects to complete a Ph.D. in Computer Science from UCLA in summer 2000. He previously worked at the San Diego Supercomputer Center and has been a summer researcher at Bell Labs and Xerox PARC. His research interests include routing protocols, infrastructure fault tolerance, and network security.

Daniela Caldararu was born in Bucharest, Romania. She holds a B.Sc. in Mathematics and Computer Science from Bucharest University, Romania. In 1997 she received her M.Sc. in Computer Science from SUNY Binghamton, with a thesis in Distributed Systems. She is currently working as a consultant for New Era of Networks.