MULTICAST ROUTING SUPPORT FOR REAL-TIME APPLICATIONS

by

Daniel Mark Alexander Zappala

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(Computer Science)

December 1997

Dedication

For Tracy, my eternal companion.

Acknowledgements

I am deeply grateful for Deborah Estrin, Scott Shenker, and Bob Braden, all three of whom have been partners in this effort. Deborah has provided consistent direction and guidance throughout my time at USC and has been willing to provide all the resources I could need to be successful. Scott has given me invaluable encouragement and feedback, inspiring clarity of both thought and prose. Bob led my efforts as part of the RSVP design team and helped me to become involved in the IETF. He has always been willing to discuss protocol design principles and details.

I am thankful for Peter Danzig, Rafael Saavedra and John Silvester for providing feedback on this work and for serving as members of my Qualifying Exam and Dissertation Committees. Many people have helped me to develop the ideas presented herein, including my fellow students and researchers at USC and ISI – Lee Breslau, Kannan Varadhan, Shai Herzog, Katia Obraczka, Sugih Jamin, John Heidemann, Ramesh Govindan and others – and my colleagues in the IETF working groups. I am also thankful for those who helped me during my internship at Xerox PARC, including Lixia Zhang, Scott Shenker, Steve Deering and Ron Frederick. I am particularly indebted to the late Kim Korner for developing my enthusiasm in operating systems as a new student at USC and for introducing me to Deborah and her network lab.

I greatly appreciate the financial support of the Air Force Office of Scientific Research under Award No. DAAH04-94-G-0378.

I am especially thankful for the love and support of my parents and for my son Dominic, who brings me joy every time I look at him. Most of all I am thankful for my wife Tracy for giving me her unqualified devotion and daily sustenance and for accepting my annual status report of "just one more year."

Contents

List Of Tables

List Of Figures

Abstract

One approach to improving the performance of real-time applications over the Internet is to design new adaptive techniques, similar to the use of TCP for elastic applications. These techniques include adapting a receiver's audio playback point and varying a receiver's subscription to levels of hierarchically encoded video. Another approach is to upgrade the best-effort service model of the Internet to include enhanced levels of service characterized by reduced delay or increased bandwidth. Researchers in this area have proposed an integrated services architecture that uses a combination of scheduling algorithms, admission control, and a reservation protocol to control access to these service levels.

While both of these approaches are promising, neither can be completely successful without a corresponding upgrade of the routing infrastructure. Real-time applications, whether they use adaptive techniques or integrated services, are currently limited to using shortest path, opportunistic routes. If an application is unable to obtain its desired service on the shortest path, routing does not supply it with an alternative route. Likewise, if an application is able to obtain its desired service on the current route, routing may opportunistically change to a new, shorter route, possibly resulting in a service disruption.

This dissertation explores several interdomain multicast routing enhancements that can improve the performance of real-time applications. We propose extending multicast routing protocols to include alternate routes and pinned (non-opportunistic) routes. Routing may use these extensions on-demand in support of receivers that need them. We have designed a simple, scalable route setup protocol that re-configures multicast trees using an explicit route. We describe how this protocol prevents loops and compare it to several other alternatives.

The utility of the route setup protocol depends in large part upon the ability of routers to construct alternate routes for multicast group members. It is particularly important that routers find routes using only partial knowledge of the network topology due to the size of the Internet. In addition, we distribute alternate path construction to leaf routers, to avoid the overhead of centrally computing routes. We have developed a set of heuristics for constructing alternate paths in this context. We describe the results of a simulation study that evaluates these heuristics and demonstrates the validity of our approach.

Given the ability to compute and install alternate, pinned routes, the network can better support real-time applications, both those using adaptive techniques and those using integrated services. Beyond these fundamental extensions, we explore a broad set of additional services that routing protocols may use specifically to support RSVP, a reservation protocol that is part of the integrated services architecture. We present designs of these additional services and qualitatively analyze their mechanistic complexity. Our goal is to help determine whether these features are feasible, as part of the ongoing investigation into the applicability of the integrated services architecture.

Chapter 1

Introduction

The Internet has proven to be highly successful in supporting elastic applications, which adapt to varying delays and packet loss. Much of this success arises from the Internet's use of the datagram as a building block, over which other services including end-to-end reliability are built [Cla88]. At the heart of the Internet are routers that implement *best-effort* service; routers do not guarantee delivery or performance. Routers simply service packets in first-come first-served order and packets are dropped from the tail of the router's queue as it overflows. To obtain end-to-end reliability out of these components, applications use TCP [Pos81b, Jac88] to adjust their sending rate and retransmit lost packets.

Recently, the Internet has begun using *multicast* delivery to support group communication [CD92, Eri94]. Multicast delivers packets from a sender to a group of receivers over a multicast tree. The primary advantage that multicast has over traditional unicast delivery is that the sender transmits a single packet to reach all of the group members, rather than sending a separate copy to each receiver. Replication of each packet is handled by the network and is done only when necessary, i.e. at the branching points in the multicast tree. Furthermore, the group model used by the Internet is *receiver-oriented*; receivers may join a group independently (i.e. senders do not control membership), and senders do not need to know the identities of group members. By avoiding the potential bottleneck at the sender, dynamic multicast applications may grow to encompass very large groups of receivers [Dee88b, MS94].

Another recent development has been the increasing use of real-time applications in the Internet. Real-time applications impose stringent delay and throughput constraints on the network, as compared with traditional elastic applications. When real-time applications communicate across a network, data must traverse the network in time for the application to use it. Likewise, a real-time application often needs a certain amount of throughput, below which it does not receive adequate service. Because of these characteristics, real-time applications require new mechanisms, beyond TCP and best-effort service, to cope with delay and loss.

One approach to improving the performance of real-time applications is to design new adaptive techniques. For example, receivers of an audio or video signal can adapt the point at which the signal is played back [BCS94] and can re-construct the timing of the signal produced by the sender [SCFJ96]. In addition, senders can use hierarchical encoding techniques to split a video signal into several composable layers, which receivers then use to reconstruct the signal. By varying the layers to which they subscribe, receivers can control their fidelity in response to congestion [MJV96].

Another approach is to upgrade the best-effort service provided by the network to accommodate real-time applications. The proposed integrated services architecture [BCS94] introduces preferential Qualities of Service (QoS) using a combination of scheduling algorithms, admission control, and a reservation protocol. Applications use the reservation protocol to request resources from routers and are either accepted or rejected by admission control at each router. The Internet community is in the processes of standardizing one such reservation protocol, RSVP [ZDE$^+$93, BZB$^+$97], which is designed for both unicast and receiver-oriented multicast sessions.

While both of these approaches are promising, neither can be completely successful unless the routing infrastructure is also upgraded. The current routing infrastructure of the Internet [Hed88, Mil84, Hed89, IDR93, RL94, Moy94b], designed to support best-effort service, has two primary characteristics, both of which pose problems for the two previously-cited approaches:

Shortest Path Current routing protocols use a single cost metric (i.e. hop-count) and then choose the least-cost path (i.e. the shortest path). An application has no alternative route to try if it does not receive acceptable service along this path. An adaptive real-time application, for example, can adjust its behavior while using the shortest path, but still may not be able to achieve acceptable delay on this path. Similarly, a real-time application using an integrated services network may have its resource reservation denied by some routers along the shortest path. Thus, many flows may be denied service even though other paths could accommodate their service requirements.

Opportunistic Current routing protocols always utilize the current shortest path, even if the previous shortest path is still functioning. This behavior is opportunistic in the sense that the routing protocol makes the switch assuming that the shorter path is always better. However, this behavior may cause a real-time application to experience a service disruption. For example, an adaptive real-time application may have adjusted its behavior to suit the current route and reached equilibrium; when the route changes, it must adapt to the new route and may not necessarily re-establish the same service level. Likewise, a reservation-based application may have reserved the current route, but may be denied making an equivalent reservation on the new route. In both cases, the duration of the service disruption may be indefinite.

This dissertation investigates solutions to these two problems, which are applicable to all real-time applications, and also examines other forms of routing support designed specifically for RSVP. We concentrate on interdomain multicast routing.

Many in the research community have investigated solutions to these two problems within the context of *QoS routing*. A typical QoS routing scheme globally distributes topology, link resource availability, group membership, and (in some cases) per-flow resource usage. A source's first-hop router then uses this information to compute a multicast tree that is known a priori to have available resources. Finally, this same router uses a source-initiated setup protocol to install the multicast tree in the network. The bulk of this work attempts to minimize tree cost, primarily for static multicast groups [BKJ83, Wax88, Cho91, KPP92]. The Internet and ATM communities have begun applying these results to link-state multicast routing protocols [RGW97, ZSSC96, PNN96].

Because these QoS routing approaches require global distribution and synchronization of such rapidly varying quantities, we do not believe they are applicable to interdomain routing, where issues of scale are paramount.[1] A global database of topology alone scales linearly with the size of the network; neither group membership nor the number of flows is limited by the size of the network. Additionally, the overhead required to maintain a consistent view of this data depends on application behavior, such as group membership

[1]QoS routing may still be used within a domain.

changes and resource usage. We believe this combination of storage and processing overhead rules out the use of centralized computation and installation of routes, as well as global optimization of these routes.

One alternative to QoS routing that does scale adequately for interdomain routing is to compute multiple paths for each destination based on relatively static routing metrics. In this approach, called *QoR routing*, the metrics reflect the "Quality of Route" using static service characteristics such as maximal bandwidth or minimal latency to indicate link capabilities.[2] The routing protocol maintains a separate routing table for each metric, and applications indicate the their desired QoR when sending data. This is similar to the congestion-sensitive type-of-service routing described in [MS95], but routes adapt only to topology changes, not resource usage. QoR routing could provide significant benefits for best-effort service by allowing, for instance, interactive applications to avoid routes involving satellite links, while enabling applications involving asynchronous bulk data transfers to seek out maximal bandwidth paths. For similar reasons, QoR routing could benefit real-time and other inelastic applications.

Because QoR routing metrics are static, this approach has none of the scaling problems of the more dynamic QoS routing proposals. However, the essence of the failed primary route problem remains; an application could only avail itself of one minimal-latency route, and one maximal-bandwidth route, etc. If a service requirement was denied along one of these pre-computed routes, there would be no way of utilizing available bandwidth along other routes with similar properties. Thus, while QoR routing can increase the chance that an application will be satisfied with the primary route, receivers need routing to install *alternate paths* into a multicast tree on demand. Likewise, QoR routing does not solve the opportunistic routing problem, since QoR routes adapt as the metrics change. Therefore, receivers need routing to also install *pinned routes* – routes that will not adapt unless they fail – into a multicast tree on demand.

The contributions of this dissertation are threefold. First, we propose a routing architecture in which alternate paths and pinned routes are installed by a multicast route setup mechanism. Existing route setup mechanisms use sender-oriented route setup or require routers to know the identities of downstream group members [DB95, FBZ94, UNI96], which renders them unusable for interdomain routing. We have designed a simple, scalable route setup mechanism named MORF and show that it prevents loops while establishing and re-routing a multicast tree.

[2]Scott Shenker coined this term and distinguished it from QoS routing.

The key to the viability of this architecture is whether routers (or route servers) can find adequate alternate paths. To scale to the interdomain level, we propose to use route construction that is both decentralized and query-driven. Routers with local receivers find alternate paths for their receivers on-demand. Moreover, these routers do not use global distribution of topology to find routes. Rather, they find routes using a partial map of the network, which they build using heuristics to query the routing tables of other nodes.

The second contribution of this dissertation is a simulation study demonstrating the viability of using localized route construction to find alternate paths around bottlenecks. Our intent in presenting these results is to demonstrate the utility of several low-cost, route construction heuristics, thus validating our architecture. As others find better route construction heuristics, routers may incrementally deploy the improvements.

In our approach to route construction, we are designing for the case when congestion is not widespread, and thus do not attempt to find paths where resource availability is known a priori. We view this as the most feasible approach for interdomain routing for several reasons. During times of high load when congestion is common, using alternate paths can degrade network utilization [KZ89, Aki84]. Moreover, it is impracticable to design a scalable route computation method for finding the proverbial needle in a haystack — the one route that has available resources among a very large number of routes that do not.

Finally, this dissertation explores a broad set of additional routing services that may be used to support RSVP within the integrated services architecture. Since there is no widespread agreement on the set of services that should be available to hosts, or even if the integrated services architecture itself should be adopted, we view our work as exploratory: we discuss the set of services routing *might* offer to support RSVP. We are not proposing that these services be adopted at this time; rather we are trying to understand what is needed mechanistically to provide these services. The design and analysis we perform can help to determine whether a particular feature is feasible from an architectural standpoint. For those features we deem feasible, the ultimate decision on adding it to future versions of RSVP and routing protocols depends on further evaluation of their utility.

The rest of this dissertation is outlined as follows. Chapter 2 discusses work related to this dissertation. In Chapter 3 we describe our proposed multicast routing and reservation architecture, giving the context in which a route setup protocol operates. Chapter 4 details our design of MORF, a multicast route setup protocol and analyzes its loop-freedom. Chapter 5 describes the interdomain route construction heuristics we have developed and presents simulations showing their effectiveness in finding alternate paths. Chapter 6 explores routing support of RSVP. Finally, Chapter 7 presents our contributions and areas for future work.

Chapter 2

Related Work

This dissertation is related to the work of many researchers. First we look at the development of multicast routing and reservation protocols. We then describe the history of alternate path and non-opportunistic routing, motivating our need to extend this work to receiver-oriented multicast. Finally, we look at route construction, including QoR and QoS routing, as well as other efforts to compute paths when the entire network topology is not known.

2.1 Multicast Routing

Multicast routing protocols form a multicast forwarding tree to deliver data to each member of a group, reducing forwarding overhead compared to using unicast routing to reach each group member. The multicast group model also provides a convenient method of logically naming an anonymous group of receivers. We are most interested in multicast routing because most of the interesting real-time routing problems become more complicated with the interaction of many receivers. In addition, the receiver-oriented model used by IP multicast is a scalable way of providing routing services on a per-receiver basis.

The IP multicast routing model, introduced by Deering [Dee88a], defines a multicast group as a collection of receivers. Each receiver independently joins the multicast group by advertising its membership to its local router. A host that wishes to send data to the multicast group does not need to be a member of the group, nor does it need to know the identity of each member. The sender simply addresses packets to the group using the multicast group address, and the network delivers the packets to each member. The network uses a multicast routing protocol to determine how routers forward packets toward receivers along the multicast delivery tree associated with the group address.

Deering developed multicast routing methods for both distance-vector and link-state protocols [Dee91]. Subsequently, Deering created DVMRP [Dee88b, WPD88] based on his distance-vector method; it is widely used in the MBone (the Internet multicast backbone) today [CD92]. DVMRP routers exchange routing metrics for each subnet in a network using a distance-vector protocol based on RIP [Hed88]. When a source begins transmitting data, the routers use the distance vector to compute on-demand a reverse shortest path multicast tree that spans the entire network. The routers then prune the tree to remove any branches that do not lead to any members. Periodically, routers graft pruned branches back onto the tree in order to capture membership and topology changes. These routing actions, termed *flood and prune*, occur only for groups whose senders are transmitting data. However, because *flood and prune* is expensive, DVMRP is best-suited for small networks and groups with dense membership throughout the network.

Based on Deering's link-state model [Dee91], Moy modified OSPF [Moy94b], a link-state unicast routing protocol, to perform multicast routing. Termed MOSPF [Moy94a, Moy94c], this protocol globally distributes membership status as part of its link-state advertisements. Routers calculate sender-based, shortest-path multicast trees on-demand and, due to their global membership knowledge, can avoid the periodic flooding and pruning of DVMRP. The multicast trees created by MOSPF are based on the shortest path from each source to each receiver, rather than on the shortest path from each receiver to the source. MOSPF aggregates routing information into areas in order to scale to large networks, at the expense of non-optimal routing between areas.

Both DVMRP and MOSPF create sender-based trees. When multiple senders are involved in group communication, routing must keep state for each source-group pair. To counteract this state requirement, Ballardie et al. developed a multicast routing protocol, CBT [BFC93, Bal97, BJR97], that uses a single, shared tree for all sources that send to a multicast group. Routers with local receivers explicitly join the multicast tree by contacting one of a set of core routers. Formation of the shared tree is based on unicast routing, making CBT independent of the underlying unicast routing. Shared trees incur greater delay than shortest-path sender-based trees. The delay incurred by a shared tree depends on the placement of cores throughout the network; while optimal placement of cores is NP-complete, heuristics can be used to reduce tree cost [TR96, Shu94]. A shared tree may also suffer from traffic concentration when many sources are active. Wei and Estrin [WE94] give an in-depth analysis of the tradeoffs of sender-based and shared trees. Billhartz et al. have also compared PIM and CBT [BCFG$^+$97]. The shared trees used by

CBT are most useful for applications that have low bandwidth requirements, can tolerate larger delays, or require low cost.

Deering et al. subsequently developed a multicast routing protocol, PIM [DEF+94, EFH+97, DEF+97], to take advantage of the benefits of both sender-based and shared trees. PIM supports two complementary routing modes, *sparse mode* and *dense mode*, designed to adapt to different multicast group and wide-area network characteristics. Dense mode multicasting is similar to DVMRP; routers build on-demand multicast trees that use a broadcast-and-prune approach suitable to networks with dense membership representation. Routers performing sparse mode multicasting first create shared trees similar to CBT, using a rendezvous point toward which members join. Once the data rate for an active source is large enough, receivers may individually join a shortest path tree for the source. Sparse mode trees are created through explicit joining; both routing modes use unicast routing to construct multicast trees.

Recently, due to the growing use of multicast routing, several groups of researchers have begun designing a hierarchical multicast routing architecture. One of these efforts [DT95] describes hierarchical use of DVMRP, by defining regions similar to domains and specifying interactions between intra-region and inter-region border routers. Routing within a region is based on an encapsulation scheme using region identifiers in order to improve scaling. Similarly, a preliminary design describes the hierarchical use of PIM [DFE+95]. More recently, Thaler et al. are designing an interdomain multicast routing protocol that builds bidirectional, shared trees to connect multicast domains and reduce routing state [TEM97].

2.2 Reservation Setup Protocols

Reservation setup protocols reserve resources along a route by contacting admission control at each switch. For multicast applications, the reservation protocol establishes a reservation over a multicast tree. Some setup protocols use a *hard-state* approach, in which the network is responsible for reliably maintaining the reservation state. Others use *soft state* that times out after a period and must be refreshed by an endpoint. In this dissertation, the reservation protocol requests routing services on behalf of the application. We are particularly interested in *receiver-oriented* reservation protocols, in which the receiver initiates the reservation, because this matches the scalability of multicast routing.

In the telecommunications industry, Plain Old Telephone Service (POTS) involves in-channel setup of a point-to-point call. To provide end-to-end digital communications, the industry introduced the Narrowband Integrated Services Digital Network (N-ISDN) specification. N-ISDN uses out-of-band signalling to merge voice and data communications on a single circuit-switched network. More recently, the telecommunications industry has positioned Broadband ISDN (B-ISDN) as the network of the future. Vendors are designing B-ISDN to provide a range of multimedia services using a fixed-size, fast packet (cell) switching technique known as ATM.

ATM signalling protocols unify routing and reservation setup into a single mechanism for connection establishment. Because ATM switches perform forwarding based on connection identifiers, the signalling protocol must create forwarding state for each new connection. Simultaneously, the signalling protocol reserves sufficient resources at each switch to carry the call. Typically signalling is divided into user-network (UNI) signalling and network-network (NNI) signalling, due to a strong distinction between endpoints (usually devices) and the network. Most ATM signalling protocols specify user-network behavior; the network signalling required to support the specified mechanisms is not fully developed. Finally, ATM signalling protocols usually run over a reliable layer and establish hard state.

A number of standards bodies, vendors and universities have explored parallel efforts to provide multicast and QOS services in B-ISDN. The International Telecommunications Union (ITU) published recommendation Q.2931 for unicast B-ISDN calls, then upgraded this specification for multicast [IT94d] and QOS negotiation [IT94a, IT94b, IT94c]. The ATM Forum produced UNI specification 3.1 [UNI94] based on these protocols. The SPANS UNI and NNI [FS94b, FS94a] specify multicast channels by allowing a source to add multiple destinations to a call. GSP [MT92b, MT92a] and EXPANSE [Min91] both provide general signalling models for multiple-connection multimedia calling. Most significantly, CMAP [BDG91] defines a multicast reservation establishment protocol with receiver-initiated joining. The group initiator may specify one of three methods of controlling how members may join the group: at the invitation of the group initiator, via member request to the initiator, or via uncontrolled member joining. The cost of allowing this level of control is that each router must know how each multicast group is controlled. Most recently, the ATM Forum has approved UNI 4.0 [UNI96] to provide receiver-initiated joins, QOS negotiation and multiple connections per call, somewhat similar to CMAP. None of these protocols allow sharing of reservations among the senders to a multicast group and thus may have excessive overhead for some applications [MS94].

In the Internet community, Forgie developed ST [For79] for unicast reservations. Delegrossi and Berger later extended this in ST-II [DB95] to provide multicast resource reservations. The ST-II protocol, designed as a separate real-time network suite, creates its own source-based multicast trees based on unicast routing. A source initiates reservations over the tree on behalf of each receiver as the receiver joins the multicast group. The capacity of the resource reservation is established by the source and may be relaxed by individual receivers. Multiple senders transmitting data to the same group of receivers must make independent reservations.

Ferrari et al., in their work on the Tenet architecture [FBZ94], are primarily concerned with establishing real-time channels that provide *a priori* network guarantees. The Tenet reservation scheme uses two passes to dedicate resources for a unicast channel. The first pass, in the forward direction, makes a conservative reservation and the second pass, in the reverse direction, relaxes the reservation to fit the end-to-end parameters of the channel. The Tenet group is currently extending the scheme to provide multicast channels.

Cidon et al. have designed a fast reservation mechanism [CGS93] used in the plaNET architecture, which provides numerous types of routing, including source routing, label-swapping, and ATM routing. Their setup mechanism takes advantage of a selective copy mechanism, whereby the connection establishment packet is forwarded immediately via hardware to each router and copied into the processing subsystem. Each router performs connection establishment in parallel, and a subsequent signal commits the reserved resources to the channel. This scheme is also currently limited to unicast applications.

Zhang et al. developed RSVP [ZDE$^+$93, BZB$^+$97] as a receiver-oriented reservation protocol designed to match Deering's burgeoning receiver-oriented multicast model. Because it is receiver-oriented, RSVP can scale to large numbers of receivers by not requiring source interaction with each receiver. Routers forward reservation requests up a multicast tree toward the source, merging requests of various receivers at the branching points of the tree. RSVP also defines several reservation styles that indicate how reservations for different senders may interact, allowing receivers to more flexibly match their reservation requests to the style of a particular application [MS94]. RSVP does not have its own multicast routing protocol, but uses whichever routing protocol has defined the multicast tree. As such, RSVP works with both shared and sender-based multicast trees. RSVP also allows transparent operation through routers that do not support the protocol.

Recently, the developers of both the Tenet architecture and the ST-II protocol have extended their setup mechanism to include receiver-initiated reservation requests [BFG$^+$95, DHHS93]. The Tenet setup protocol is also being extended to include multicast channels, and both are implementing channel grouping mechanisms similar to RSVP's once-unique reservation styles. Once these enhancements are more fully developed, the primary difference between them and RSVP will be their *hard-state* nature. Using hard-state provides stable routing to real-time applications while making failure recovery mechanisms more difficult to implement. RSVP uses a *soft-state* mechanism that provides simple and robust failure recovery, but makes stable routing a more difficult service to implement. RSVP owes its soft-state design primarily to the Internet design philosophy discussed by Clark [Cla88]. Because the RSVP design follows the modular split between reservations and routing, it is more applicable to the routing work of this dissertation than the Tenet reservation protocol, for example.

2.3 Alternate Path Routing

Typically, networks use a preferred or default route for most traffic. Some networks have used alternate paths, in addition to the default route, to improve performance. In this dissertation, we integrate alternate path routing into adaptive multicast routing to provide reservable routes for multicast real-time applications. We are thus interested in observing how other researchers have used alternate paths in support of unicast routing and extending these results to multicast routing.

In order to provide paths sensitive to network conditions, the ARPANET routing algorithms used load based routing to construct lowest-delay paths [MRR95]. However, these algorithms were shown to lead to oscillations [KZ89]. When a link became congested, routes using the link adapted to a different path. This caused a new link to become congested while the original link became uncongested, leading to a new set of route changes. These results initiated a trend toward less load-sensitive routing.

Current routing protocols, such as OSPF [Moy94b], RIP [Hed88], IDRP [IDR93], and BGP [RL94], are based on topology and compute shortest-paths that adapt only to topology changes. IDPR [ES91] used source routes to enhance shortest-path routing with policy-based routing. Typically, policy routing is used to determine which entities may send data through a region of the network. However, this functionality is similar to that of real-time applications requiring a path that has adequate resources available.

The routing algorithms developed for virtual circuit routing, deployed primarily by the telecommunications industry, have used alternate path routing during congestion. Generally the primary route consists of a one-hop path and the alternate routes have two hops. Routing uses the primary path if it is available; otherwise, it chooses an alternate path. Algorithms such as DNHR [AKK81], DAR [GKK88], and FAR [MS91] select from a set of pre-computed alternate paths and differ only in their manner of path selection. DNHR computes alternate paths based on estimates of long-term traffic patterns. DAR randomly computes a set of alternate paths, while FAR computes alternate paths based on load balancing and maximization of profit. Other algorithms, such as DCR [HSS91], ALBA [MG90, MGH91] and RTNR [AH93] control routing based on real-time measures of network load. DCR uses a central processor to collect load measurements every 10 seconds and adjust switch routing tables based on these measures. Both ALBA and RTNR use decentralized algorithms to measure link utilization and classify each link according to a set of discrete levels. The goal of both of these algorithms is to perform load balancing for the network.

A common problem with alternate path routing is that it can decrease throughput at very high load [Aki84]. This decrease in throughput results when calls routed along alternate, two-hop paths block calls from being routed along a direct path. Virtual circuit routing algorithms solve this problem by reserving a portion of each link for calls routed directly on the link. This method, known as trunk reservation, limits the amount of alternate paths that routing may use during high load.

Similarly, several researchers from the Internet community have used adaptive routing to avoid congestion and improve throughput. Attar [Att81] developed a link-state routing algorithm that ranks paths according to preference. Nelson et al. [NSC90] used alternate paths that were triggered by a single router detecting congestion. Wang and Crowcroft [WC90] used a similar method to provide shortest path routing with “emergency exits.” Each of these algorithms separate route computation from route selection based on load measures.

Breslau [Bre95] developed a comprehensive alternate routing architecture based on source routing for alternate paths. In his model, sources select alternate routes based on load information that the network distributes in a limited fashion. His results indicate that this architecture can improve throughput, setup delay and route quality. Breslau also extended the benefits of trunk reservation in circuit-switched networks to the use of alternate paths in data networks.

Breslau’s work verifies the premise of the unified routing model [ERH92], which uses both hop-by-hop routing and source routing. Generic hop-by-hop routes are used for most traffic, while source routes are used for more specialized traffic requirements. The open issues with the unified architecture are route construction and the integration of a source routing protocol with common hop-by-hop routing protocols such as BGP and IDRP.

2.4 Non-Opportunistic Routing

The virtual circuit routing algorithms employed by the telecommunications industry do not adapt opportunistically to topology changes. For example, non-opportunistic routes work well in the telephone network because of its relatively simple service model and stable usage patterns. In addition, because a typical phone call lasts on the order of minutes, the overhead for setting up a virtual circuit is tolerable.

Through the use of source routes, the Internet also supports non-opportunistic routing [Pos81a]. Breslau's work on alternate path routing [Bre95], cited earlier, shows how the network can use source routing to improve throughput.

ST-II [DB95] specifies a way to pin the route of a unicast or multicast flow by following hop-by-hop the routes installed by the routing protocol. A source initiates a stream by identifying an initial set of receivers. The control protocol carries a joint reservation and routing setup message along the shortest path routes from the sender to these receivers. At each hop, the control protocol creates non-opportunistic routing state for the stream. The source may use a similar mechanism to add new receivers to the stream during the call. Receivers may also request that they be added to the stream by contacting a router on the tree. The branch is not established by the receiver however; the router that is already on the tree is responsible for extending the tree to the new receiver. Because ST-II does not have a mechanism for grafting receivers into the tree using source routes, it is restricted to achieving reservations over a small set of routes, without any recourse to alternate paths.

The method that ST-II uses to pin a route is somewhat similar to the method this dissertation evaluates in Chapter 4, albeit for a source-based model rather than a receiver-based model. ST-II identifies several failure scenarios that may lead to looping during installation of a route. Because routing setup is restricted to the forward direction from the tree to the receivers, however, ST-II's looping situations are simpler to detect and prevent. This simplicity comes with the significant cost of requiring routers on the multicast tree to know of all downstream receivers that they add to the tree.

Recently, Guerin et al. have proposed a similar method for pinning unicast routes that is integrated into RSVP [GKH97]. However, while ST-II removes a pinned route if looping occurs during setup of the route, the Guerin proposal tries to adjust by temporarily unpinning the route. This proposal has not yet been extended to multicast routing, and the group is now looking at source routing alternatives [GKR97b, GKR97a].

The Tenet architecture [FBZ94], similar to ST-II, uses only non-opportunistic routes to support real-time connections and establishes a connection using joint routing and reservation setup. Unlike ST-II, however, the routes are pre-computed at the edge of the network. This results in more expensive route computation, with the benefit of the opportunity to use alternate paths. The Tenet architecture does not specify how routes are precomputed however, nor how the network ensures tree integrity when receivers join the tree via alternate paths.

Both ST-II and Tenet use a conservative, hybrid network architecture. Real-time applications use non-opportunistic routing as their basic network service. Data applications use separate network protocols. There are several important ramifications that result from using this type of network architecture. Using non-opportunistic paths requires a "customized" route for each connection, based on the application's service requirements. By forcing all real-time applications to use this type of routing, the architecture loses the scaling benefit of allowing adaptive real-time applications to use hop-by-hop routing. In addition, strictly separating real-time and non-real time connections hinders multicast communication among heterogeneous groups of receivers.

2.5 Route Construction

One method for overcoming the limitation of using only a single shortest path is to compute multiple such paths distinguished by their Quality of Route (QoR). As a means for providing a variety of paths to meet application requirements, QoR routing is not a novel idea. Several routing protocols such as OSPF [Moy94b] allow the creation of multiple routing table entries per destination, and a Type of Service field in IP packets has been designed to index those tables [Alm92]. Recent work by Matta and Shankar [MS95] indicates that such an approach can improve overall end-to-end delays in the network. While their approach uses metrics based on measured delay and utilization, the formulations of their metrics result in relatively static measures that thus do not exhibit the oscillation seen in the ARPANET [KZ89].

Most of the research community has instead concentrated on using Quality of Service (QoS) routing to solve this same problem for small networks. The goal of QoS routing is to compute paths that are known to have available resources. This foundation of this work is the development of heuristics for minimizing the network cost of a multicast tree, which is an NP-complete problem. Kumar and Jaffe [BKJ83] show that heuristics based on shortest paths have comparable average performance to heuristics based on minimal spanning trees. Waxman [Wax88] extends the basic multicast routing problem to include a bandwidth constraint and dynamic joining and leaving by group members. Kompella et al. [KPP92] propose a heuristic for multicast trees with a delay constraint. Salama et al. do a comprehensive evaluation of both unconstrained and constrained algorithms, all for static multicast trees (i.e. the group members are all known in advance). The Internet and ATM communities have begun applying these results to link-state multicast routing protocols [RGW97, ZSSC96, PNN96]. For all of this work, network topology and link characteristics are known globally and the root of the multicast tree runs the routing algorithm.

In contrast, this dissertation explores several route construction heuristics for large-scale networks that do not require global distribution of topology. We are primarily concerned with techniques that can be implemented using today's routing infrastructure. Alaettinoglu has explored hierarchical extensions to routing that enable large-scale policy and ToS routing [Ala94]. This work includes route construction heuristics based on querying different levels in the hierarchy to obtain detailed information and compute policy-sensitive routes. Hotz has studied route construction heuristics based on route fragments, which are precomputed partial source routes, and triangulation of graph position [Hot94].

Chapter 3

Proposed Multicast Routing and Reservation Architecture

The proposed integrated services Internet includes a multicast routing and reservation architecture designed to allow RSVP to interoperate with current routing protocols (Figure 3.1). The key aspect of this architecture is the split between routing and reservation protocol functions. The distinction between these two systems was defined in the original RSVP publication [ZDE+93] and has since been refined within the community. The general rule of this division of labor is that routing decides whether to forward packets, whereas RSVP influences how they are forwarded. Thus, unicast and multicast routing protocols install the routes that data traverses, and RSVP associates a Quality of Service with the data as an alternative to the default best-effort service. This architecture allows RSVP to operate over any routing protocol and also allows RSVP and routing enhancements to be developed independently.

To this architecture, we add a multicast route setup protocol for installing alternate and pinned routes. To find alternate paths, routers at the endpoints of the network (i.e. near hosts) may also incorporate a local route construction agent. This chapter discusses each of these two additions.

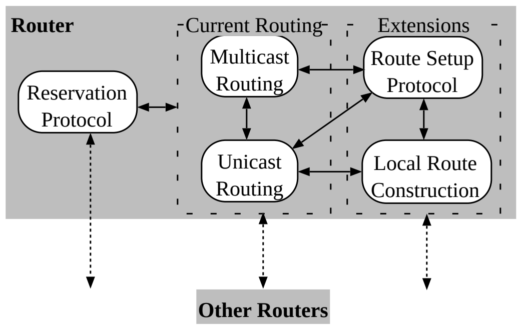


Figure 3.1: Multicast Routing and Reservation Architecture

3.1 Multicast Route Setup Protocol

The function of the multicast route setup protocol is to install alternate and pinned routes on behalf of receivers, overriding the opportunistic routes used by a multicast tree. Any route it installs is pinned so that it remains in place until it fails, at which time the multicast tree migrates back to an opportunistic route.

In keeping with the split between routing and reservation protocols, we separate route setup from reservation setup. Our primary reason for this is that applications not using reservations may want to utilize route setup. For example, use of video- and voice-conferencing over the multicast backbone is widespread today [CD92], but each receiver is limited to using the shortest path. When congestion occurs on this path, a receiver may want to use route setup to install an alternate path, yet still use best-effort service over that path if it yields adequate performance. In this context, applications could use a reservation protocol as yet another, separate enhancement of this service model. This requires that we not embed route setup in the reservation protocol itself, but rather incorporate it into the basic routing infrastructure. In addition, because both RSVP and the routing services described herein are still under development, it is particularly important that they be developed in a modular manner.

3.2 Local Route Construction Agent

One of the key challenges of this multicast routing architecture is to develop a route construction protocol that may be applied to the interdomain scale. We distribute route computation to routers located near receivers and do not attempt to find routes based on link resource availability. Instead, routers first use the shortest path and then, as needed, find alternate paths that avoid any bottlenecks. This design follows the unified routing model [ERH92], in which commonly-used routes are pre-computed and routes used less frequently are computed on-demand.

Using localized route construction reduces the problem of constructing multicast routes to that of constructing unicast routes. Each route construction agent only needs to find a route between a local receiver and the sender, rather than having to take into account the entire multicast tree. This approach scales well to large multicast groups and allows agents to use existing unicast routing protocols as the basis for a route construction algorithm.

In addition, using localized route construction has several other advantages. First, because a route construction agent does not need to coordinate its routing decisions with

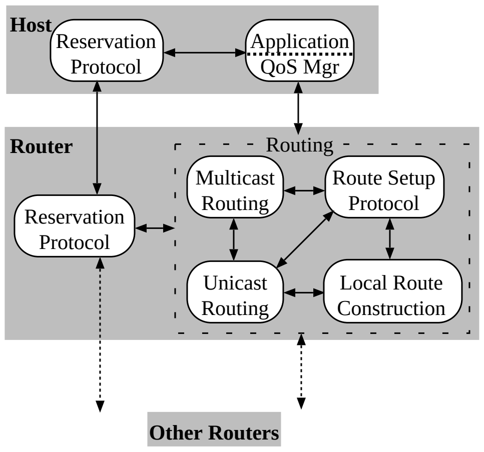


Figure 3.2: Application Interface to Architecture

other agents, it can utilize information that is not captured by the routing protocol's static metrics. This information can include the status of local reservation requests or resource availability information that is only known locally. In addition, route construction agents do not need to globally agree on a metric or algorithm. This allows diversity in the evolution of route construction techniques.

One consequence of using localized route construction is that routers may choose conflicting routes. Because any node in a multicast tree must have a single parent, routes using conflicting parents must be resolved. In the multicast routing architecture, the route setup protocol resolves such conflicts as it installs each route. Section 4 discusses this procedure in more detail.

3.3 Application Interface

Figure 3.2 shows how applications and the reservation protocol interact with our additions to the architecture. Applications access route setup and reservation setup through two separate interfaces, in contrast to many QoS routing proposals[FBZ94, DB95, Sti95] that use a single interface. Note that there is no application interface to the local route construction agent; the application's first-hop router contacts the agent when it needs a route. By not allowing the application to determine the routes being used, we prevent a malicious, malfunctioning, or misguided user from providing its own route and undermining the integrity or efficiency of a given tree.[1]

[1]We are indebted to our colleagues Steve Deering and Van Jacobson for emphasizing that end systems should not control routing.

In the most simplistic model of how this architecture would be used, applications would interface directly with routing and the reservation protocol, respectively. For the new services we propose, for example, they would ask routing for an alternate path or pinned route. In fact, given the complexity of the service options available, we assume that many operating systems will offer some form of support for managing quality of services. Such a *QoS manager* could act as an agent on behalf of an application, managing the reservation establishment and the routing services procurement process. This abstraction would allow applications to operate in terms of the service being presented to the user. If the service is unacceptable, a user (or a monitoring program) could indicate this to the QoS manager, which could ask for an alternate route or choose some other action. Unacceptable service could be indicated by a user's unhappiness with packet delays or an admission control failure along the shortest path. Likewise, acceptable service could be indicated by low packet loss or a successful reservation. In this case, the QoS manager may want to minimize the chance of disruption and ask routing to pin the current route.

The QoS management function could reside either in the application itself, or in a separate library, or in some other form of operating system support. All of these possibilities fit within our proposed architecture; we are defining the services available to an end-host, not the organization of software on an end-host. We are also not designing the QoS manager itself. We note that there are many possible actions the QoS manager may take upon after a reservation failure or success. After a reservation failure, a router may choose to downgrade its reservation level, downgrade its application type (i.e. switch from voice to text), or leave a multicast group for a higher level of video signal encoding. The reverse actions apply equally for a reservation success. Determining which of these actions the QoS manager chooses in each situation is beyond the scope of this dissertation.

Chapter 4

Designing a Loop-Free Multicast Route Setup Protocol

Within the architecture described in Chapter 3, the multicast route setup protocol installs pinned routes and alternate routes. Installation of a route may be initiated by any router on behalf of its local receivers, and computation of the route is not necessarily coordinated with any other routers. Thus, when installing a route, the primary tasks of the route setup protocol are to arbitrate among conflicting route setup requests and to maintain a loop-free multicast tree. For this chapter we assume that all requests carry equal weight, so the setup protocol resolves conflicts in first-come, first-served order. Chapter 6 discuss the complications that arise when some requests outweigh others.

In this chapter, we explore various distributed, receiver-oriented designs for a loop-free multicast route setup protocol. For the purposes of this discussion, we use the term *leaf router* to mean a router acting on behalf of local receivers. Routers, not hosts, are the entities that setup routes. Furthermore, for each of the designs we describe, the messages of the protocol operate on a multicast tree. This tree may be for a single sender [DEF$^+$94], or multiple senders may rendezvous via a core [DEF$^+$94, BFC93]. In either case, the protocol is the same; in the following discussion we refer to root of a tree for generality.

We begin by describing the MORF route setup protocol and compare it to several similar protocols. We then discuss extensions for bundling, recovering from failures, pinning routes on multi-access subnets, and doing unicast route setup. Finally, we discuss the pitfalls of relying on hop-by-hop routing decisions when performing route pinning.

4.1 The MORF Multicast Route Setup Protocol

MORF is a general route setup protocol that can install both alternate and pinned routes. A leaf router using MORF encodes the route it wants to use as a *strict explicit route*, which

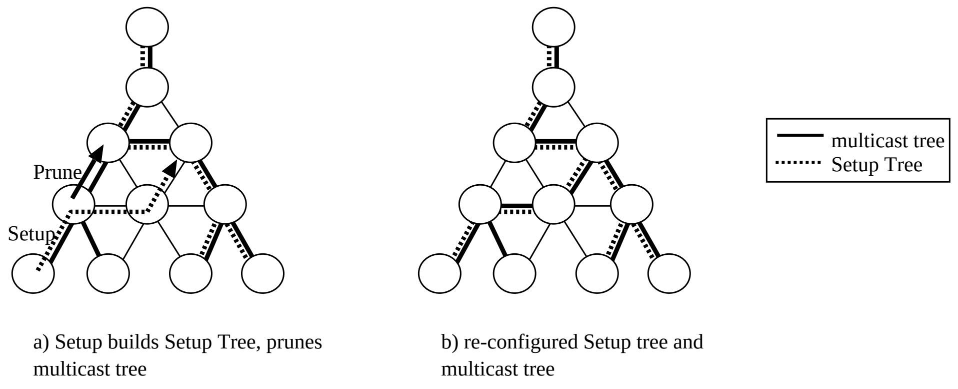

Prune
Setup
multicast tree
Setup Tree
a) Setup builds Setup Tree, prunes multicast tree
b) re-configured Setup tree and multicast tree

Figure 4.1: Using a *Setup* Message to Install a Route

lists all the routers to be traversed.[1] MORF then installs this route and re-configures the multicast tree as needed.

The installation of routes is the same for both alternate paths and pinned routes. As it is installed, an alternate path needs to be pinned as well, or else routing would immediately adapt back to the shortest path. Thus, we can view the route pinning and alternate path services as nearly equivalent; the only difference is in the way the route is obtained. For route pinning, a leaf router probes the network for the current route (i.e. with the *traceroute* utility or by querying its routing database). For alternate path routing, a leaf router may contact a route server to compute a suitable alternate path.

Once a router obtains an explicit route that it wishes to install, it generates a *Setup* message containing the route. MORF forwards the *Setup* message along the route, creating a Setup Tree that it maintains separately from the shortest-path tree built by the multicast routing protocol.[2] Where the Setup Tree conflicts with the shortest-path tree, MORF overrides the shortest-path tree, and the multicast routing protocol prunes the conflicting branches (Figure 4.1a). MORF also adjusts forwarding table entries so that the resulting multicast tree reflects the path installed by MORF (Figure 4.1b).

This first example is a simplified case when a *Setup* does not conflict with the rest of the Setup Tree. However, the setup protocol must also resolve *Setup* messages from different leaves that use conflicting routes, because leaf routers may use independent route construction agents. MORF resolves conflicts by choosing the first route that is installed

[1]We could use the term source route, but the route lists hops from the receiver to sender and is installed by a router near the receiver.

[2]To reduce the amount of state kept by routers, the multicast routing protocol could just flag incoming and outgoing interfaces that are pinned. Here we use separate multicast trees to simplify the description of the protocol.

```
Procedure Setup($child, root, group, route$)
begin
    if ($\exists timer^{Setup}$)
        if ($route^{Setup} \neq$ upstream($route$))
            send Failure($child, root, group, route, route^{Setup}$, “merge failure”)
        else
            $child^{Setup} \leftarrow child^{Setup} + child$
        endif
        return
    endif
    create $timer^{Setup}$
    $child^{Setup} \leftarrow child^{Setup} + child$
    $parent^{Setup} \leftarrow$ nexthop($route$)
    $route^{Setup} \leftarrow$ upstream($route$)
    if ($\neg$ end($route$))
        send Setup($parent^{Setup}, root, group, route$)
    endif
    return
end

Procedure Failure($parent, root, group, route_{tried}, route_{tree}, reason$)
begin
    if ($\nexists timer^{Setup}$)
        return
    endif
    foreach($c \in child^{Setup}$)
        send Failure($c, root, group, route^{Setup}, route_{tree}, reason$)
    endfor
    delete $timer^{Setup}$
end
```

Figure 4.2: MORF Route Setup Messages

for any given branch of the tree. Where subsequent routes meet this branch, they must conform to the route used from that point upward toward the source. If the setup protocol does not follow this restriction, then a number of looping scenarios may arise; Section 4.1.1 discusses these cases and the manner in which they are prevented.

Figure 4.2 shows the details of the MORF mechanism. Each router on the Setup Tree keeps state consisting of:

$route^{Setup}$	cache of upstream route,
$parent^{Setup}$	parent of Setup Tree, and
$child^{Setup}$	children of Setup Tree.

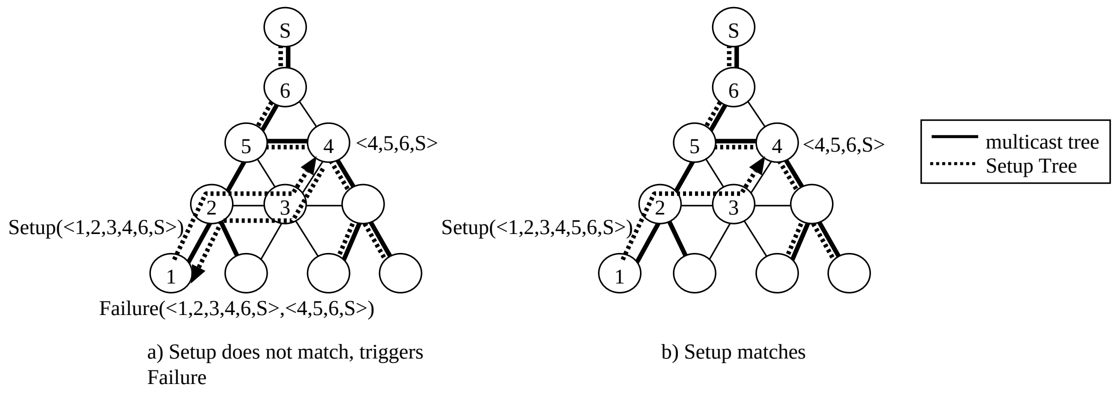


Figure 4.3: Matching *Setup* Messages

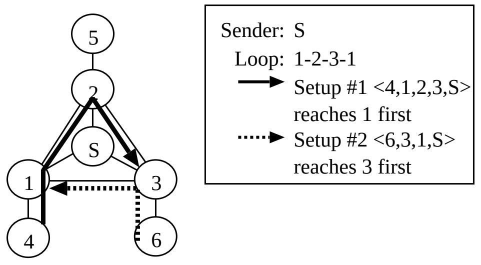
Note that separate state is kept for each multicast tree, but we omit the root and group in this notation for readability.

When a *Setup* message adds a node to the Setup Tree, it caches the route it will use to travel from that node upward toward the root. If a subsequent *Setup* message arrives at that node, it compares the remaining route it must travel to the route cached locally. If the routes match, the node merges the *Setup* message. If the routes do not match, the node stops processing the *Setup* and sends a *Failure* message downstream (Figure 4.3a). The *Failure* message contains the route used by the failed *Setup* and the route used by the tree from the rejecting node upward. A router receiving a *Failure* message merges the two routes it contains to construct a new route that will match the tree, then sends a second *Setup* with this route (Figure 4.3b).

It is from this mechanism – *Match or Fail* – that MORF derives its name. By using this restriction, MORF may increase the setup latency, but it prevents any loops from forming while the tree is constructed. The remainder of this section discusses potential looping scenarios and analyzes the tradeoffs of MORF versus other solutions for preventing loops.

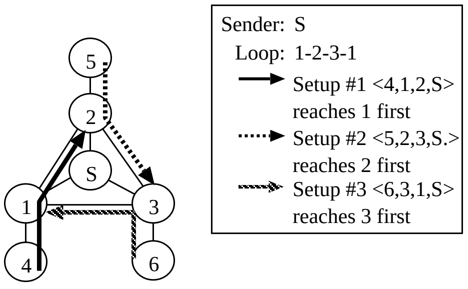
4.1.1 Preventing Loops

When *Setup* messages are not restricted to matching the rest of the Setup Tree, a number of possible looping scenarios arise. Figure 4.4a shows two *Setups*, each using an explicit route. Based on their order of arrival, as shown, if the *Setups* merge they form a loop. This loop can be prevented if nodes 1 and 3 compare the routes and detect the loop will form. However, when three joins are involved, as in Figure 4.4b, a single node cannot prevent the loop from forming without having more information available.



5
2
S
1
3
4
6
Sender: S
Loop: 1-2-3-1
Setup #1 <4,1,2,3,S> reaches 1 first
Setup #2 <6,3,1,S> reaches 3 first

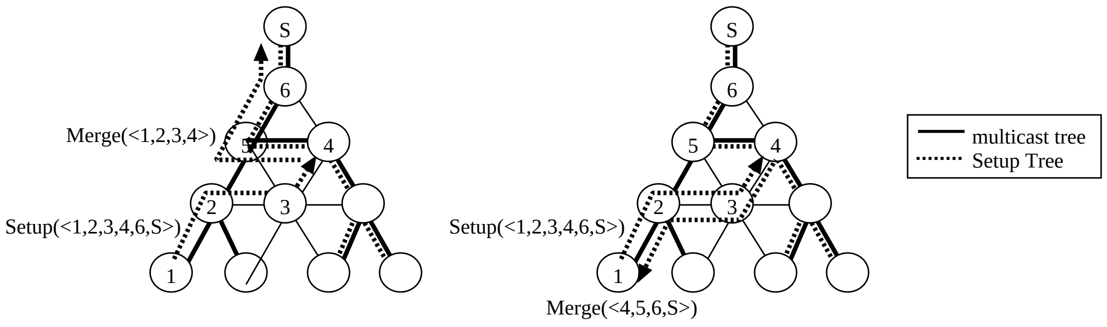
a) Loop formed by two Setups



5
2
S
1
3
4
6
Sender: S
Loop: 1-2-3-1
Setup #1 <4,1,2,S> reaches 1 first
Setup #2 <5,2,3,S.> reaches 2 first
Setup #3 <6,3,1,S> reaches 3 first

b) Loop formed by three Setups

Figure 4.4: Loops Formed by Naive *Setup* Messages



a) Setup triggers Merge sent upstream

b) Setup triggers Merge sent downstream

Figure 4.5: Merging *Setup* Messages Instead of Matching

To prevent loops, a node can use one of two strategies:

1. Before adding a parent, the node checks all its descendants to be sure the parent is not already a descendant.

2. Before adding a child, the node checks all its ancestors to be sure the new child is not already an ancestor.

We discuss each of these in turn.

Due to the dynamic nature of multicast trees, a node may not know all of its ancestors or descendants. While a node knows the route embedded in the *Setup* message it has sent upstream, that message may have merged with another *Setup* carrying a different route. Likewise, other *Setups* may have merged downstream, adding new descendants.

One approach to keep nodes updated concerning upstream and downstream merges is to distribute information after each merge. Following solution (1) above, each *Setup* that merges can send a Merge message upstream containing the portion of the route it has travelled (Figure 4.5a). Every node can then know all its descendants and thereby detect

Table 4.1: Comparison of Setup Mechanisms

Mechanism Name	Message Overhead	Storage Overhead	Setup Latency	Loop Handling
MORF	O(1)	O(a)	1 or 3 trips	Prevent
Merge Down	O(1)	O(a)	1 trip	Detect/Break
Merge Up	O(d)	O(d)	1 trip	Detect/Break

any loops. Alternatively, in keeping with solution (2) above, each *Setup* that merges can send a Merge message downstream containing the upstream portion of the route it merged with (Figure 4.5b). This allows every node to detect loops by knowing all its ancestors. We denote these two mechanisms as *Merge Up* and *Merge Down*, respectively. In both of these approaches, information distributed by the Merge messages may be stale, so loops such as that shown in Figure 4.4 may still form temporarily before being broken.

As opposed to these solutions, which in some cases will only detect loops after they have been formed, the strategy we use in MORF prevents any loops from forming. By requiring each *Setup* to match the upstream route already in place on the tree, MORF in effect enforces solution (2) by requiring that each node know its ancestors before it is added to the tree. Once a node is added to the multicast tree, its ancestors do not change. The cost of this strategy is that each *Setup* may take an extra round-trip between itself and the rest of the tree.

4.1.2 Comparison of Setup Mechanisms

Table 4.1 compares MORF to the Merge Down and Merge Up mechanisms, when building a single multicast tree, assuming there is no packet loss and that one receiver joins the tree at a time. The columns listing message and storage overhead consider the behavior of each mechanism at a single node. Overhead in these cases is expressed in terms of a, the number of ancestors of a node, or d, the number of descendants of a node. The setup latency column lists time in terms of the number of trips taken between a leaf router and the multicast tree.

Clearly the Merge Up mechanism does not scale well because each node must store each descendent as well as send one message upstream for each descendant. The advantages of using a receiver-oriented mechanism are lost with Merge Up; a sender-oriented mechanism has the same message overhead, but only the sender must store the descendants.

The MORF and Merge Down mechanisms have a similar overhead in this situation. The MORF mechanism may have a longer setup latency, but in return has the distinct advantage that it may prevent rather than just detect loops, as discussed above.

When we relax the assumption that one receiver joins the tree at a time, thus allowing multiple simultaneous *Setups*, the other tradeoffs of these two mechanisms become more apparent. In this situation, MORF must take into account conflicting *Setups*. We assume that it will use a binary exponential backoff to prevent thrashing. If we also assume a message transmission takes a uniform time t when sent over any link, then the dynamic setup latency for MORF is:

$$Latency_{MORF} = 3Lt(c+1) + \sum_{i=1}^{c} b * 2^{i-1},$$

where L is the average length of the branch from a leaf router to the rest of the tree, b is the backoff constant, and c is the number of conflicts the *Setup* encounters.

When considering these dynamic conditions, each node using the Merge Down mechanism may potentially send $O(a)$ messages downstream, since each ancestor may send the node one Merge message. In addition, the setup latency for Merge Down must take into account the time required to break loops. The worst case time to break a loop of m nodes is $(m-1)t$, so the setup latency can be given by:

$$Latency_{MergeDown} = 2Lt + \sum_{i=1}^{l}(m_i - 1)t,$$

where l is the number of loops encountered and m_i is the number of nodes in loop i.

As can be seen from this analysis, the Merge Down mechanism requires a robust protocol design to ensure that loops are quickly detected and broken. The more merges that occur simultaneously, the longer it will take for the mechanism to distribute the information needed to break the loops. The Merge Down mechanism will also have to detect when a Merge message is lost, as that event can cause a loop to persist. In contrast, MORF uses a simpler protocol to prevent loops and uses more complexity only at the edges of the network.

4.2 Recovering From Failures

A route setup protocol should be resistant to failures, which can be a router crashing, a process on the router crashing, or any other event that causes the router to lose state associated with the route setup protocol. A common approach to handling lost state is for

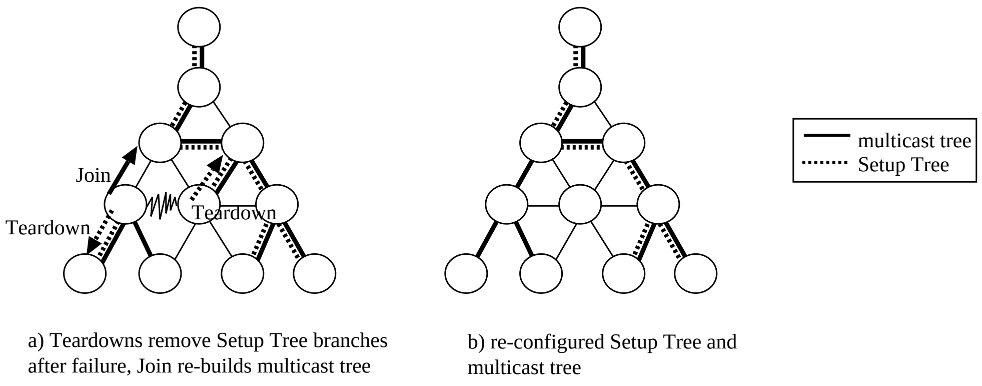

Join
Teardown
Teardown
multicast tree
Setup Tree
a) Teardowns remove Setup Tree branches after failure, Join re-builds multicast tree
b) re-configured Setup Tree and multicast tree

Figure 4.6: Using a Teardown to Remove a Failed Route

each node to periodically send a copy or a refresh of the previous message sent upstream to rebuild any lost state. This approach – often called *soft state* – greatly simplifies a protocol because it does not need to distinguish between sending a new message or sending a copy. A *hard state* protocol, on the other hand, uses separate messages to establish and then maintain protocol state.

RSVP is often cited as an example of the soft state approach. With RSVP, a refreshed reservation can be sent upstream to re-establish a lost reservation or to adapt to a route change. If it were to use a hard state approach, RSVP would need to use additional mechanism to discover when state is missing and then use some recovery mechanism to re-establish the reservation or else remove the reservation altogether.

A route setup protocol does not make opportunistic routing changes, but it can use refresh messages to rebuild lost state if the router with the failure is still operational. The same techniques that MORF uses to prevent loops when initially installing an alternate or pinned route apply equally to refreshes of a *Setup* message.

However, MORF still needs a mechanism to detect and recover from failures when the failed router is no longer operational. MORF may rely on a unicast routing protocol to exchange query messages with its neighbors to determine whether they are alive, or it may use its own similar mechanism. Once a failure is detected, MORF sends a *Teardown* message both upstream and downstream of the failure to remove failed branches from the Setup Tree (Figure 4.6a). At each hop, MORF notifies the multicast routing protocol of the branches it is removing. The multicast routing protocol re-builds the multicast tree to reflect its metric, often the shortest path to the root (Figure 4.6b).

There will be some delay between the time a pinned router fails and the time downstream nodes receive a *Teardown* to remove the pinned branch. During this delay, other

Setup messages could merge with the orphaned routers downstream of the failure. The newly-established branch will not actually be connected to the rest of the multicast tree and will be removed as soon as the upstream failure is detected.

The MORF protocol prevents any loops from forming in this situation. However, it can use a fast teardown mechanism to both reduce the chance of orphan merging occurring after a failure and to quickly clean up after orphan merging does occur. Because this mechanism is not needed to prevent loops, it is an optimization.

For the fast teardown mechanism, when a router sends a *Teardown* downstream it expects that it will be acknowledged with a *TearAck* by all of its pinned children. Until the router receives all of the *TearAcks*, it re-sends the *Teardown* to its non-responding children at a relatively frequent rate. The router also sets a limit on the maximum time it will wait for its children to respond, and assumes they have failed if that limit is reached. This process of collecting acknowledgements repeats itself, hop-by-hop, down the orphaned subtree until all of the branches are removed. Figure 4.7 details the processing for this mechanism. Note that a more basic mechanism can be derived by removing the lines that reference sending a *TearAck* or maintaining a timer.

4.3 Pinning Routes on Multi-Access Subnets

MORF needs extra mechanism to prevent duplicates when pinning a route on a multi-access local-area subnet (or LAN), such as an Ethernet. If one router is pinning the multicast tree so that it forwards packets onto the LAN, then the other routers must not forward packets onto the LAN. When a router enters a state where it explicitly does not forward packets onto a LAN, it is doing *route excluding*, the opposite of route pinning. The problem of ensuring that only a single router forwards packets onto a LAN is also addressed by the PIM design [EFH+97] and has a similar solution for route pinning.

Figure 4.8 details the mechanism routers use to determine when they should do route excluding. All routers on a LAN must share state about which of them (if any) is doing route pinning. Whenever a router pins a route on a LAN, it periodically sends a *Notify* message on the LAN indicating the tree it is pinning for. Routers receiving a *Notify* begin route excluding and set a timer for the excluding state to expire. This timer is reset every time another *Notify* is received.

Due to errors or lost state, several routers on a LAN may both decide to do route pinning for the same tree at the same time. To resolve such conflicts, a router that receives a *Notify* when it is pinning for the same tree compares its own IP address to that of the

```
Procedure Teardown($node, root, group, router_{failed}$)
begin
    send TearAck($node, root, group$)
    if ($\nexists timer^{Setup}$)
        return
    endif
    if ($node \in child^{Setup}$)
        $child^{Setup} \leftarrow child^{Setup} - node$
        if ($child^{Setup} = \emptyset$)
            send Teardown($parent^{Setup}, root, group, router_{failed}$)
            delete $timer^{Setup}$
        endif
    endif
    if ($node = parent^{Setup}$)
        foreach($c \in child^{Setup}$)
            send Teardown($c, root, group, router_{failed}$)
        endfor
        $n^{Setup} \leftarrow 1$
        $timer^{Setup} \leftarrow timer$
        while($child^{Setup} \neq \emptyset$)
            wait for event
            if (receiveTearAck(c,root,group))
                $child^{Setup} \leftarrow child^{Setup} - c$
            endif
            if ($timer^{Setup}$ expires)
                if ($n^{Setup} \geq$ maxtries)
                    $child^{Setup} \leftarrow \emptyset$
                else
                    foreach($c \in child^{Setup}$)
                        send Teardown($c, root, group, router_{failed}$)
                    endfor
                    $n^{Setup} \leftarrow n^{Setup} + 1$
                    $timer^{Setup} \leftarrow timer$
                endif
            endif
        endwhile
        delete $timer^{Setup}$
    endif
end
```

Figure 4.7: Fast *Teardown* Mechanism

```
**Procedure Notify**$(peer, root, group)$
**begin**
    **if** $(\nexists timer^{Setup})$
        create $timer^{Setup}$
        $excluding^{Setup} \leftarrow 1$
        $timer^{Setup} \leftarrow timer$
        **return**
    **endif**
    **if** $(excluding^{Setup} = 1)$
        $excluding^{Setup} \leftarrow 1$
        $timer^{Setup} \leftarrow timer$
        **return**
    **endif**
    **if** $(myIPaddress > peer)$
        **send Teardown**$(parent^{Setup}, root, group, myIPaddress)$
        delete $parent^{Setup}$
        delete $child^{Setup}$
        delete $route^{Setup}$
        $excluding^{Setup} \leftarrow 1$
        $timer^{Setup} \leftarrow timer$
    **else**
        **send Notify**$(peer, root, group)$
    **endif**
**end**
```

Figure 4.8: *Notify* Mechanism

sender of the message. If its address is larger, then it relinquishes its pinning. Otherwise, it sends another *Notify* message to force the other router to relinquish its state and begin excluding.

4.4 Bundling

Some applications may divide a stream of data into components and send each component to a different multicast group. For example, a video application may hierarchically encode the video stream [Sha92] and send each encoding level to a separate group. Recent work in this area has demonstrated that receivers may adapt to congestion in the network by subscribing to different encoding groups and varying the quality of the signal they receive [MJV96].

Current multicast routing protocols would route these multicast groups independently and, depending on the protocol, there is a chance that the trees would take different paths.

In the typical case, this may not be a problem and may actually provide the benefit of load-sharing. However, if each encoding level is marked with a different priority, then a receiver may prefer that lower priority packets are dropped when congestion occurs. If the encoding levels take different paths, higher priority packets might be dropped when they would otherwise get through.

To address this concern, routing could offer *bundling* of multicast groups so that they each use the same route. A multicast routing protocol could accomplish this by listing a set of groups – called the *bundling set* – in control messages that build a multicast tree. For example, when PIM sends a Join message upstream, the message can contain the bundling set, and routers processing the message can install forwarding entries for all of the groups. Likewise, when MORF sends a Setup message, it can list the bundling set and install the alternate path for all the groups.

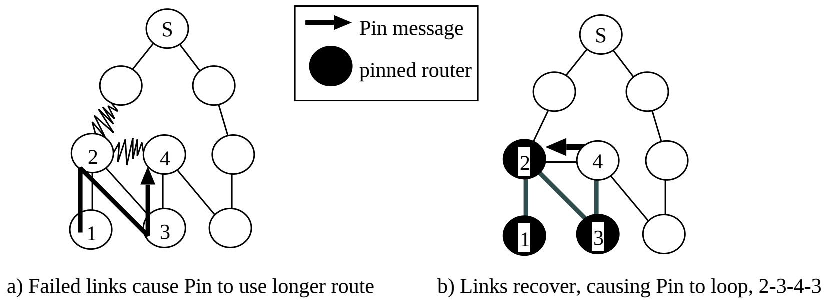
To request this service, receivers need to notify routing of the bundling set. We may reasonably expect that an application would enforce a consistent bundling set among all receivers requesting the service. Without this homogeneity, routing could not construct a meaningful service from a set of conflicting requests. Any router performing bundling that encountered a conflicting bundling set would thus return an error indicating that it could not provide the requested service.

4.5 Unicast Route Setup

Previous work has studied the efficacy of using explicit routing to support unicast real-time applications [Bre95]. One way to use explicit routes to provide alternate paths or pinned routes is to embed the explicit route in an application's packets [EZL$^+$96, HLFT94, DH95]. Assuming the route will be inserted by the sender's nearest router, no modifications to applications will be needed. However, because many routers currently delay processing of explicitly routed packets, this mechanism may not be applicable to applications with strict delay requirements.

An alternative is for the sender's nearest router to insert a label in the application's packets rather than an explicit route. This label would reference an alternate path or pinned route that is installed using MORF.[3] Because unicast applications involve only one receiver, the setup latency will only be 1 trip.

[3]The label could in fact be a multicast group address, reducing unicast alternate path routing to a special case of multicast.



Pin message
pinned router
S
2
4
1
3
a) Failed links cause Pin to use longer route
S
2
4
1
3
b) Links recover, causing Pin to loop, 2-3-4-3

Figure 4.9: *Pin* Message Loop

4.6 Pitfalls of Pinning Using Hop-by-Hop Routing

An alternative design for route pinning is to use hop-by-hop routing rather than explicit routes. With this design, a leaf router initiates route pinning by sending a *Pin* message upstream. This message does not carry an explicit route, but instead follows the currently-installed route toward the root of the multicast tree. Each router processing the message pins the child link over which it arrived and then queries its routing table for the next hop toward the root. It then pins the parent link toward the next hop router and forwards the message to the next hop. The *Pin* message stops when it reaches the root or a node that has already been pinned. This section discusses the pitfalls of this approach.

4.6.1 Preventing Loops

The route a *Pin* message follows may change before the route is completely pinned. If this happens, the *Pin* message may loop back on itself. Looping becomes even more likely if several routers send *Pins* at the same time because the messages can form loops by merging into each other.

Figure 4.9 shows an example of a single *Pin* message may loop back on itself. Two links have failed, causing routing to adjust the multicast tree to that shown in Figure 4.9a. Router R1 begins pinning its current route and sends a *Pin* message toward the root of the tree. By the time the *Pin* reaches router 4, the failed links have recovered and the multicast tree has changed changed. Router 4 forwards the Pin Request along this new path, forming a routing loop, shown in Figure 4.9b.

To prevent such loops from forming, we can add a status flag to each node on the multicast tree. When a router on an adaptive branch receives a *Pin*, it sets the flag to "pending", signifying that pinning not yet been established (Figure 4.10a). If the request reaches the root, routing sends a *Connect* message downstream to set the flag

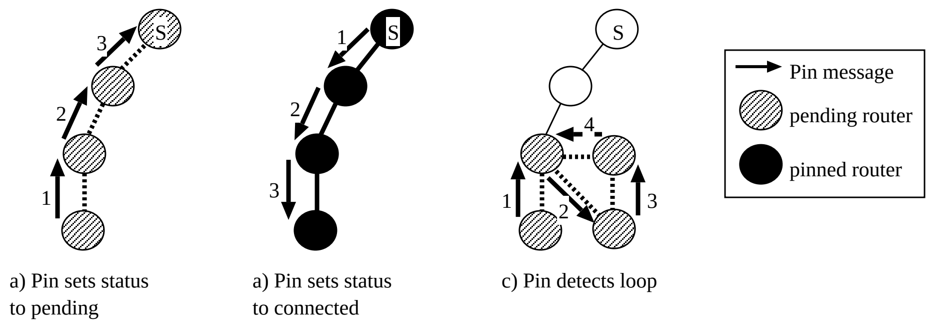


Figure 4.10: Connecting a Pinned Branch

to "connected", confirming that the branch has been added to the tree (Figure 4.10b). Likewise, if the request reaches a connected portion of the tree, routing sends the *Connect* downstream from that point. However, if a *Pin* reaches a pending branch, it may have looped back on itself (Figure 4.10c). Routing must reject the request and send a *Teardown* message downstream to remove the branch and any associated state. Upon receiving a *Teardown* message, the router that originated the *Pin* should then back-off before re-sending it.

4.6.2 Comparison to Explicit Routing

The hop-by-hop route pinning protocol, as outlined above, is less robust than MORF or the other explicit route setup mechanisms. If protocol state is lost but the router with the lost state is still operational, then MORF can rebuild the lost state and still prevent loops. In contrast, the hop-by-hop route pinning protocol can form a loop in this situation. Thus, it can pay a large penalty for not detecting a failure and recovering properly and needs additional mechanism.

Consider a time t_0 when pinning state upstream of node n is lost. Node n and all of the nodes downstream of n have the status flag set to "connected." This means that if n re-sends the *Pin* message, and the message reaches some connected node downstream of itself, it can form a loop, the same as in Figure 4.9. As can be seen from this example, setting the status flag to "connected" means that node is connected to the root of the tree. That condition is violated when state is lost and is difficult to rectify.

A hop-by-hop route pinning protocol is thus more amenable to a hard state approach, where separate messages are used to maintain a pinned branch once it is established. Figure 4.11 shows how such a protocol would work when state is lost. *Pin* and *Connect* messages have established the pinned portion of a multicast tree shown in Figure 4.11a.

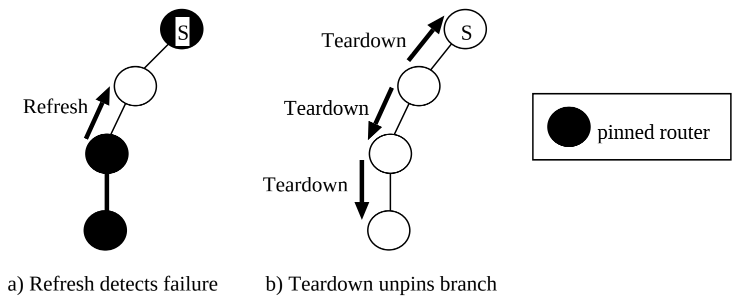


Figure 4.11: Separate *Refresh* Message

Receiver R periodically sends a *Refresh* message to keep this state alive. Suppose that R sends a *Refresh* after a pinned router fails. When the *Refresh* reaches a router without pinning state (or a router that is pinning on a different branch of the tree), it detects a failure (Figure 4.11b). The router sends a *Teardown* both upstream and downstream to unpin the branch (Figure 4.11c). In order for this solution to work, the *Pin* and *Connect* messages must carry a unique branch identifier to reference the branch they are pinning. The *Refresh* carries this identifier so that it can detect when the branch has failed.

We have written a prototype route pinning protocol using this latter approach and integrated it into the DVMRP multicast routing protocol [WPD88]. The prototype was successful in demonstrating that route pinning can be added to a multicast routing protocol, even one that uses a flood-and-prune mechanism. However, the protocol proved difficult to implement because of the complicated state machines and numerous message types needed for the hard state approach.

Chapter 5

Interdomain Route Construction Heuristics

Given a scalable interdomain multicast route setup protocol, such as MORF, the important issue we must address is whether we can construct useful alternate paths at a reasonable cost. Due to the scaling problems inherent in distributing global topology information at the interdomain level of the network, we propose that a route construction agent use heuristics to partially explore the interdomain topology and find routes.

We have developed several low-cost heuristics that do not require changes to routing protocols, thus allowing incremental deployment at the edges of the network. To determine their effectiveness in finding alternate paths, we have conducted a simulation study over various types of random topologies. For the purposes of our simulations, we have concentrated on validating our approach by trying to find routes around a single overloaded interdomain-level link.

5.1 Approach

Our approach relies on route construction agents to serve local receivers (Figure 5.1a). When a receiver needs an alternate path, its local agent uses heuristics to find a route around any bottlenecks. If the local agent is unable to find an alternate path, it may contact an agent near the sender for a route (Figure 5.1b).

The agents do not have full topology information; they instead build a partial map of the topology to find alternate paths. We have focused on methods for gathering topological information that is available with existing routing protocols.

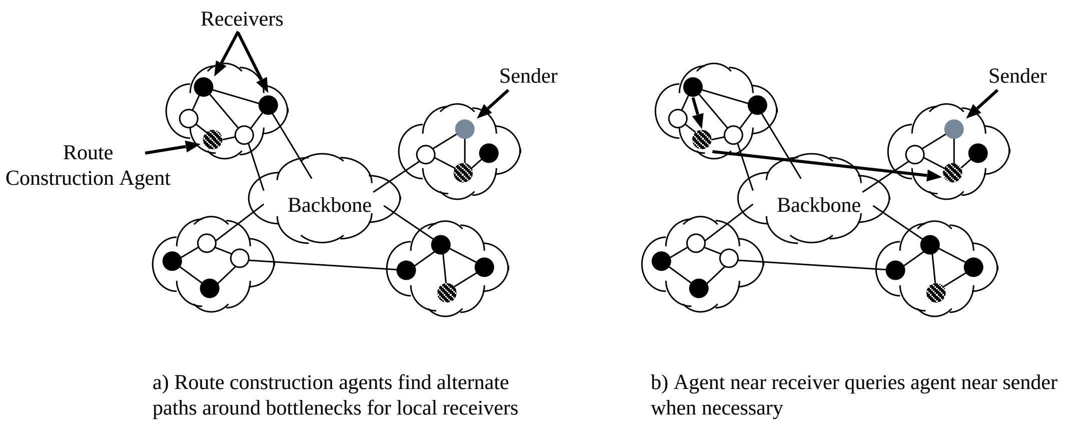

Receivers
Sender
Route Construction Agent
Backbone
Sender
Backbone
a) Route construction agents find alternate paths around bottlenecks for local receivers
b) Agent near receiver queries agent near sender when necessary

Figure 5.1: Route Construction Agents Serve Local Receivers

We have developed the following algorithm for exploring paths:

1. An agent explores the current path from itself to a small set of *Initial Nodes* in the network. These nodes may be randomly chosen or may consist of all the nodes within n hops. The agent initializes its map with these paths.

2. When the route setup protocol requests an alternate path from the agent, the request identifies a multicast tree (a group and a sender or core) and a bottleneck link. The agent probes the multicast tree for the requesting receiver's current path to the sender, adds this information to its map, and marks the bottleneck link.

3. The agent then runs a Dijkstra computation over its current map to find an alternate path around the bottleneck link. If one is found, it returns this path to the route setup protocol.

4. If a path is not found, the agent augments its map by exploring the current path from some other node in its map – a "third party" – to the sender. Third parties are chosen from the *Initial Nodes* in order of their proximity to the receiver until one is found that has a route around the bottleneck link. Any links explored during this process are added to the partial map. If no alternate path is found, then the agent may optionally contact an agent that serves the sender of the multicast tree.[1]

The variable parameters in this algorithm are the method by which nodes are selected to initialize the agent's map – either random or within n hops – and the option to query

[1]To find a route around multiple bottlenecks, the agent should re-run the Dijkstra computation if all third parties have been exhausted. This is necessary because collectively all of the third party explorations may succeed even if individually none of them do.

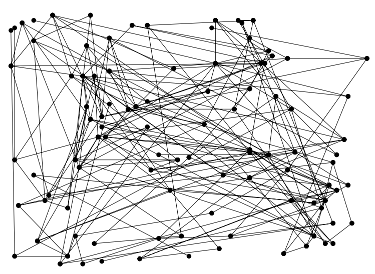
Table 5.1: Overhead Bounds for Route Construction Heuristics

Heuristic	Overhead Bound
N-Hop	$c(c-1)^{N-1}$
N-Hop+Sender	$2c(c-1)^{N-1}$
M-Random	M
M-Random+Sender	$2M$
N-Hop+M-Random	$M + c(c-1)^{N-1}$
N-Hop+M-Random+Sender	$2M + 2c(c-1)^{N-1}$

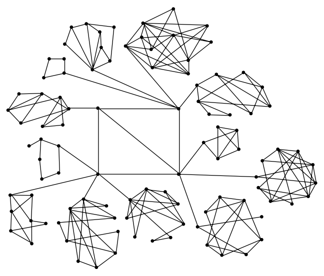
another agent near the sender of the tree when a route cannot be found locally. Combining the variations of these parameters yields the following set of heuristics:

N-Hop	Initialize using all nodes within N hops,
N-Hop+Sender	Same as above, but query the sender's agent if unable to find a route locally,
M-Random	Initialize using M random nodes,
M-Random+Sender	Same as above, but query the sender's agent if unable to find a route locally,
N-Hop+M-Random	Initialize using all nodes within N hops and M random nodes (not including any nodes within N hops),
N-Hop+M-Random+Sender	Same as above, but query the sender's agent if unable to find a route locally.

We can bound the overhead of each of the above heuristics in terms of the number of third-party queries performed for a single alternate path search. Table 5.1 lists these bounds in terms of c, the maximum degree of connectivity of the network; N, the number of hops explored initially; and M, the number of random nodes explored initially. In the algorithm given above, third-party queries are limited to the set of *Initial Nodes*, thus overhead is bounded by the size of this set. In the case of the N-Hop heuristic, this bound is $c(c-1)^{N-1}$; deriving the bounds for the other heuristics is straightforward. By keeping N small (i.e. 1 or 2 hops), we can limit the overhead of all of the heuristics to a small number of third-party queries.



a) 100-node flat random network



b) 100-node transit-stub network

Figure 5.2: Flat Random and Transit-Stub Networks

5.2 Simulation Model

We have implemented the route construction algorithm given above within the LBNL network simulator [MFF] to evaluate effectiveness in finding alternate paths. Our primary goal is to characterize the performance of the heuristics according to a varied set of topologies. We are also interested in measuring the path length of alternate paths computed using the heuristics; they should not be many hops longer than the shortest path.

5.2.1 Topology

We generated various large topologies using the Georgia Tech ITM topology generator [ZCB96, CZ]. We used one flat random network of 100 nodes (Figure 5.2a), using the Doar-Leslie edge-connection method [DL93] to generate edges that mostly connect nodes "near" each other. The average degree of connectivity for this network is 4.26. We also created transit-stub topologies, which consist of a backbone network and connected stub networks, with each sub-network generated randomly. Figure 5.2b shows a 100-node transit stub network we used, having an average degree of connectivity of 3.74. To determine how well our heuristics scale to larger topologies, we also generated three 1000-node transit-stub networks.

5.2.2 Workload

Given a single topology, the performance of the route construction heuristics will depend on the locations of the agent, the sender, and the bottleneck link. We have designed a workload that characterizes the effectiveness of a heuristic for a given topology by varying the placement of these three entities. For each topology, a simulation begins by randomly choosing a sender and a receiver. It then iterates through each of the links between the sender and receiver, assuming in turn that each link is the bottleneck link. The route construction agent, which we assume is co-located with the receiver, then tries to find an alternate path around that link.

The output of a single simulation, using a single sender-receiver pair and for a single link, is a binary indication of whether an alternate path was found, the shortest path distance, the alternate path distance (if any is found), and the number of third-party queries needed for the computation. If the heuristic is based on *M-Random*, then we repeat each simulation 100 times to compute averages for the relevant numbers. Other heuristics only need to be run one time. For each of the generated topologies, we ran simulations for 100 sender-receiver pairs.

5.3 Simulation Results

We ran a battery of simulations using the above workload for each of the route construction heuristics. We then evaluated each of the heuristics based on success rate and path length.

5.3.1 Success Rate

To determine the effectiveness of the heuristics, we compute the success rate by dividing the number of successes versus the number of attempts. We do not count attempts where there is no alternate path available, i.e. where even an algorithm with full knowledge of the topology will not find an alternate path. It is important to note that our experiments have underestimated the success rate because a given route server is finding alternate paths to only a single sender. In practice, a route server may handle requests for routes to a large number of senders, and any single request may benefit from routes learned for other requests. This will particularly be true when finding routes around local bottlenecks.

Table 5.2 shows the success rate of some of the route construction heuristics for a 100-node flat random network. For the flat random network, the 1-Hop and 1-Random heuristics are able to find an alternate path 77% and 68% of the time, respectively. Any combination of these heuristics with each other or with querying the sender is generally effective over 90% of the time.

Although the average success rate for the 1-Hop and 1-Random heuristics is similar, the distribution of the success rate among various sender-receiver pairs is quite different. Figure 5.3 shows a histogram of the success rate for these two heuristics. For these histograms, we group the alternate path attempts of each sender-receiver pair and calculate the overall success rate of each pair. Based on these histograms, querying local routers

Table 5.2: Success Rate of Heuristics on 100-node Flat Random Network

Heuristic	Average Success Rate
1-Hop	0.77
1-Hop+Sender	0.93
1-Random	0.68
1-Random+Sender	0.88
1-Hop+1-Random	0.90
1-Hop+1-Random+Sender	0.98
2-Hop	0.97
2-Hop+Sender	1.00

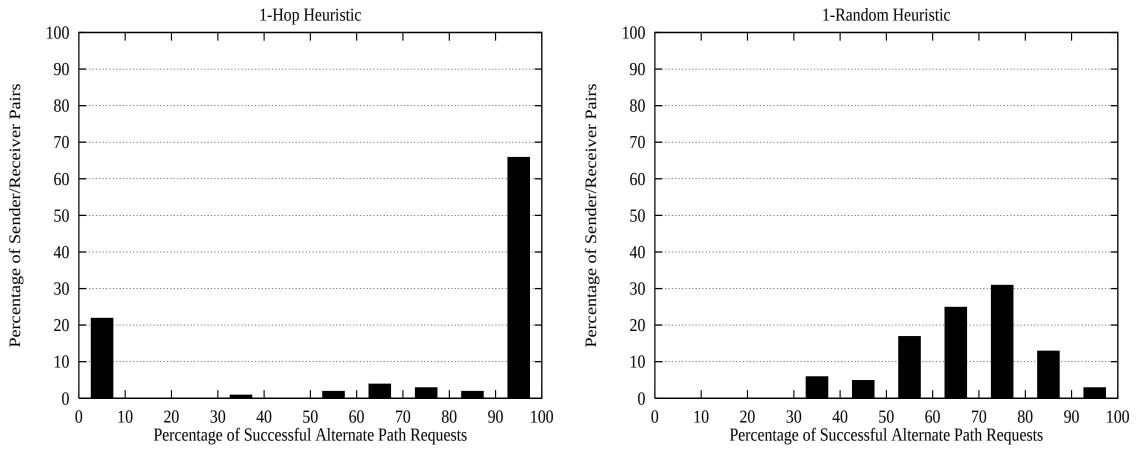


Figure 5.3: Histogram of Success Rate on 100-Node Flat Random Network

will always be helpful for some sender-receiver pairs and will never help for others. On the contrary, querying a single random router will always help to find some alternate paths in a flat random network.

While the histograms are useful for determining the distribution of a single heuristic, graphing the cumulative distribution of the sender-receiver paired success rates is useful for comparing many different heuristics. For ease in reading these graphs, we have converted the cumulative distribution function $F_x(a) = P(x <= a)$ into a *diminutive distribution function* (DDF) $F_x(a) = P(x >= a)$. Thus, for a given point on the graph, the y value represents the percentage of sender-receiver pairs whose success rate is greater than or equal to the x value. For example, Figure 5.4a shows that, for the 1-Hop heuristic, 75% of the sender-receiver pairs find an alternate path 50% of the time, and 66% always find an alternate path. This figure also shows that, for the 1-Random heuristic, all of the pairs

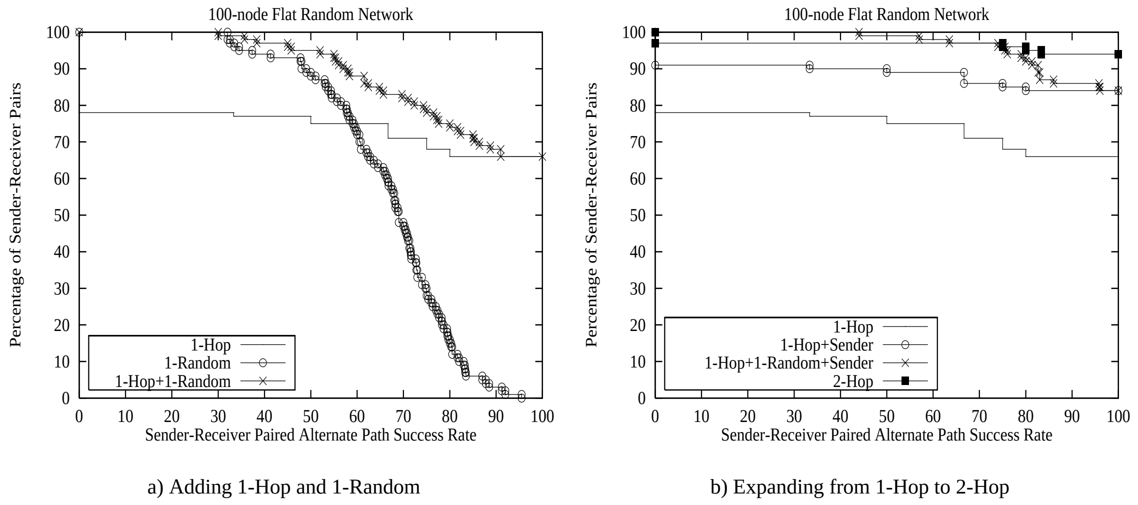


Figure 5.4: DDF of Success Rate on 100-Node Flat Random Network

are successful at least 32% of the time, but only 3% have a success rate higher than 90%. The 1-Hop+1-Random heuristic combines the potential for high success of 1-Hop with the lower bound of 1-Random.

Because 1-Hop has such a high potential success rate for a sender-receiver pair, we are interested in the value of combining it with other heuristics. Figure 5.4b compares the effectiveness of adding 1-Random and a query to the Sender to 1-Hop, versus expanding 1-Hop to 2-Hop. Adding just a query to the Sender to 1-Hop increases the percentage of pairs that always find an alternate path from 66% to 91%. Adding 1-Random to this combination raises the lower bound on success rate from 0 to 44%. While 2-Hop still has a lower bound of 0%, only 3% of the pairs fall in this category. Almost all of the other pairs always find an alternate path.

While many of the heuristics perform well on the flat random network, they all perform substantially worse on the 100-node transit-stub topology. Table 5.3 shows the success rate of the same heuristics run over the transit-stub network, along with the difference between this rate and that for the flat random network.

In particular, the 1-Random heuristic finds an alternate path only 7% of the time, compared to 68% for the flat random network. The reason for this decline can be seen by by examining the 100-node transit stub topology in Figure 5.2b. Nearly all of the nodes (96 out of 100) are located in one of the stub networks, each of which has an average of 8 nodes. Thus when the simulation randomly chooses a sender and a receiver, most likely they will be located within two different stubs. Then, when the agent explores the path

Table 5.3: Success Rate of Heuristics on 100-node Transit-Stub Network

Heuristic	Average Success Rate	Decline From Flat Random
1-Hop	0.35	0.42
1-Hop+Sender	0.68	0.25
1-Random	0.07	0.61
1-Random+Sender	0.15	0.73
1-Hop+1-Random	0.41	0.49
1-Hop+1-Random+Sender	0.74	0.24
2-Hop	0.41	0.56
2-Hop+Sender	0.77	0.23

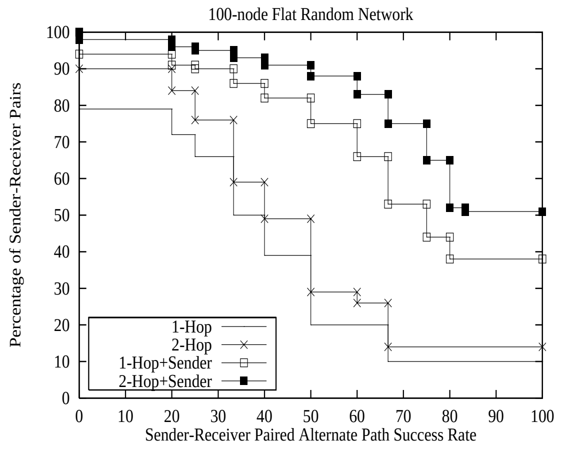


Figure 5.5: DDF of Success Rate on 100-Node Transit-Stub Network

to another random node, it is likely to choose a node that is in yet a third stub. With this map, the agent will not explore any part of the sender and receiver's stub networks except for the shortest paths through them. Thus the only times the agent is likely to find an alternate path will be the rare occasions when the third party is within the same stub as the sender or receiver or when the bottleneck is in the backbone.

The N-Hop+Sender heuristic, on the other hand, is better able to find alternate paths around local bottlenecks, either within the vicinity of the receiver or near the sender. Thus, this heuristic is much more successful on the transit-stub network than those using random nodes. Figure 5.5 demonstrates the effectiveness of querying the Sender in this topology, showing the DDF for 1-Hop and 2-Hop both with and without a query to the sender. Clearly the benefit of querying the sender far outweighs the benefit of adding an extra hop to the heuristic. Likewise, increasing the number of hops has more impact when

Table 5.4: Generation Parameters For Transit-Stub Networks

Network Name	Number Nodes	Number Transits	Transit Size	Stubs/ Transit	Stub Size
T100	100	1	4	3	8
T1000-1	1000	1	40	3	8
T1000-2	1000	10	4	3	8
T1000-3	1000	1	4	3	83

the heuristic also queries the sender, since both the sender and receiver are expanding the radius of their search. To further confirm the suitability of N-Hop+Sender for finding local bottlenecks, we re-analyzed the data for this topology considering only bottlenecks within a stub. In this scenario, the success rates of 1-Hop+Sender and 2-Hop+Sender rise to 85% and 93%, respectively.

Our simulations on 100 nodes thus confirm the following results:

- Topology affects the performance of the heuristics.
- Querying local nodes helps find alternate paths around local bottlenecks.
- Querying the sender always helps to find alternate paths, particularly for local bottlenecks near the sender.

We believe that finding routes around local bottlenecks is an important case because we expect the backbone of the network to be well-engineered, whereas a connecting domain may experience temporary overload.

To observe how well these heuristics scale to larger networks, we repeated our simulations using three 1000-node transit-stub networks. We generated each of these networks by building on the 100-node transit stub topology and making a different modification for each of the three networks. Table 5.4 lists the generation parameters for all of the transit-stub networks, which have an average degree of connectivity of 3.74, 4.35, 3.60, and 4.45 respectively. For network T1000-1, we specified a larger transit network, resulting in a larger, highly-connected backbone and more stub networks (each node in the backbone retains the same number of average stub networks). For network T1000-2, we specified more transit networks, resulting in a larger, hierarchical backbone and likewise more stubs. Finally, for network T1000-3, we specified larger stubs, which retains a very small backbone and the same number of stubs. To keep the node degree low for these stubs, we used the Doar-Leslie edge connection method.

Table 5.5: Success Rate of Heuristics on 1000-node Transit-Stub Networks

Heuristic	Average Success Rate			
	T100	T1000-1	T1000-2	T1000-3
1-Hop+Sender	0.68	0.60	0.46	0.76
2-Hop+Sender	0.77	0.73	0.59	0.86
1-Hop+1-Random+Sender	0.74	0.90	0.81	0.80
1-Hop+Sender, Stubs	0.85	0.87	0.88	0.87
2-Hop+Sender, Stubs	0.93	0.97	0.97	0.99

Table 5.5 shows the success rate of some of the heuristics run over these larger topologies as compared with the 100-node network. Because the 1-Hop+Sender and 2-Hop+Sender heuristics find alternate paths around local bottlenecks, they perform best on network T1000-3, in which the size of the stubs dominate the network. Likewise, these same heuristics do not perform as well when the number of transit networks is increased in network T1000-2. These two heuristics continue to find alternate paths around local bottlenecks, either within a stub or a transit network, but do not find alternate paths around distant bottlenecks when the connection between networks is hierarchical. On the other hand, when the backbone consists of a large, flat transit network, as with network T1000-1, these heuristics can also find alternate paths within the backbone. The last two lines of the table emphasize the ability of the N-Hop+Sender heuristics to find routes around local bottlenecks. If overall performance on a variety of hierarchical networks is a consideration, then the 1-Hop+1-Random+Sender heuristic, by combining local queries with random queries, is able to consistently find routes around both local and distant bottlenecks.

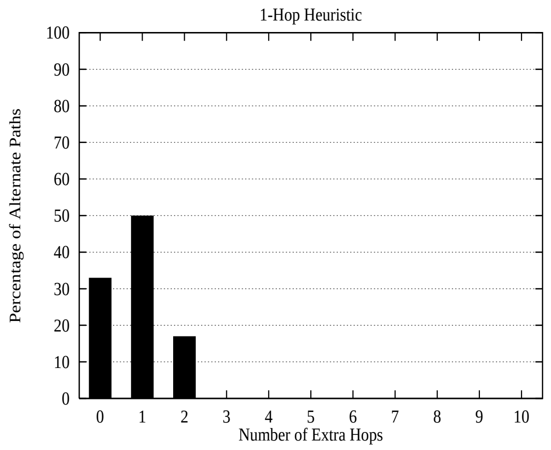
5.3.2 Path Length

We compared the length of each alternate path found by the heuristics to the shortest path used in each case. By taking the difference in path length we can determine the number of "extra" hops in the alternate paths found by the heuristics. We compare this average to the average alternate path length found by algorithm using global knowledge of the topology, which will always find the shortest available alternate path.

Table 5.6 lists the average number of extra hops for some of the heuristics on each of the topologies. We have grouped the data into several categories for ease in comparing the results. The first group lists the average number of extra hops for the global algorithm. Compared to this, the 1-Hop heuristic generally has comparable paths, with 1-Random

Table 5.6: Path Length of Alternate Paths

	Average Extra Hops per Alternate Path				
Heuristic	F100	T100	T1000-1	T1000-2	T1000-3
Global	0.67	0.69	0.62	0.79	0.63
1-Hop	0.84	0.79	0.80	0.85	0.86
1-Random	2.43	1.07	1.14	2.31	2.18
1-Hop+Sender	0.82	0.82	0.80	0.70	0.73
2-Hop+Sender	1.02	0.86	0.83	0.80	0.87
1-Hop+1-Random+Sender	1.15	0.83	0.91	1.57	0.82



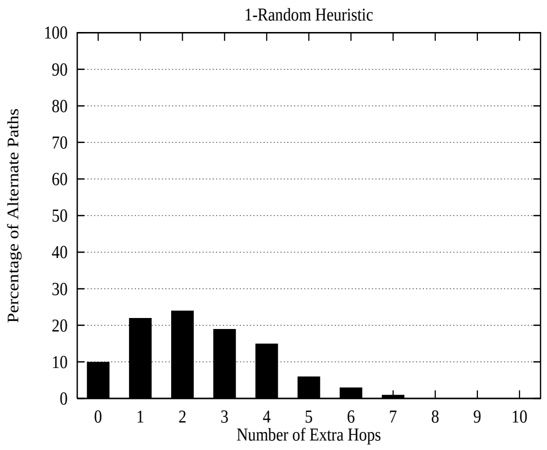


Figure 5.6: Histogram of Extra Hops per Alternate Path on 100-Node Flat Network

having much longer paths. The last group of data includes the three heuristics whose success rates are the highest.

These results indicate that the N-Hop heuristics often find an alternate path whose length is equal to that of the shortest path. The heuristics that use random selection of third parties, however, will often find paths that are much longer. These heuristics may select a third party that is distant from both the sender and receiver, so it is possible for them to find a path that has $2d$ extra hops, where d is the diameter of the network. Figure 5.6 shows a histogram of the extra hops per alternate path for the 1-Hop and 1-Random heuristics on the flat random network. While the longest path for the 1-Hop heuristic is 2 hops longer than the shortest path, the 1-Random heuristic finds paths that are as much as 7 hops longer. In many cases, a route server using 1-Random can do extra exploration of third parties (i.e. by using as a third party every node between itself and the randomly-chosen node) and find shorter paths. However, this could increase the overhead of the exploration beyond the bounds derived in Section 5.1.

From the data shown above it is also clear that the topologies we generated have multiple equal-cost paths to many destinations. If this also holds true for real-world networks, then a distance-vector unicast routing protocol like BGP [RL94] could pass through some equal-cost paths to route servers, simplifying route computation. If these paths are used frequently, then it will be cheaper to distribute them rather than compute them individually at each route server. A route server would still need to use the route querying heuristics described within this paper to find less-frequently used and slightly longer alternate paths when other paths are not adequate. This division of labor between pre-computed paths and on-demand computed paths has been proposed earlier as a part of the unified routing architecture [ERH92].

Chapter 6

Routing Support for RSVP

As described in Chapter 3, one of the possible uses of alternate path routing and route pinning is to support a reservation protocol, such as RSVP [ZDE+93]. For example, when a router receives a reservation failure, one of its possible actions is to ask routing for an alternate path around that bottleneck. Likewise, when a router determines that it has successfully reserved a route, it may request route pinning.

The purpose of this chapter is to describe other mechanisms that routing may use to support RSVP. We examine the tradeoffs of these support services in terms of the utility they offer versus the cost of the mechanism. Many of the services explored herein are equally applicable for any reservation protocol that is receiver-oriented.

When discussing the architecture for alternate path routing and route pinning, we have been careful to emphasize the separation of the routing and reservation protocols. The general rule of this division of labor is that routing decides whether to forward packets, whereas RSVP influences how they are forwarded. However, the services discussed in this chapter do not lend themselves to such a clean separation, as they require much more cooperation between RSVP and routing. Thus, in this chapter we combine routing and reservation setup whenever it significantly simplifies the mechanisms required to implement a particular service. In keeping with the general rule separating the two protocols, we propose that routing perform this combined signalling whenever a mechanism determines how a multicast tree is routed. Otherwise, RSVP should incorporate the indicated functionality.

We begin this chapter by giving a brief overview of RSVP as background for subsequent discussions. We then describe a number of routing services for supporting RSVP. We conclude by discussing the tradeoffs of the mechanisms.

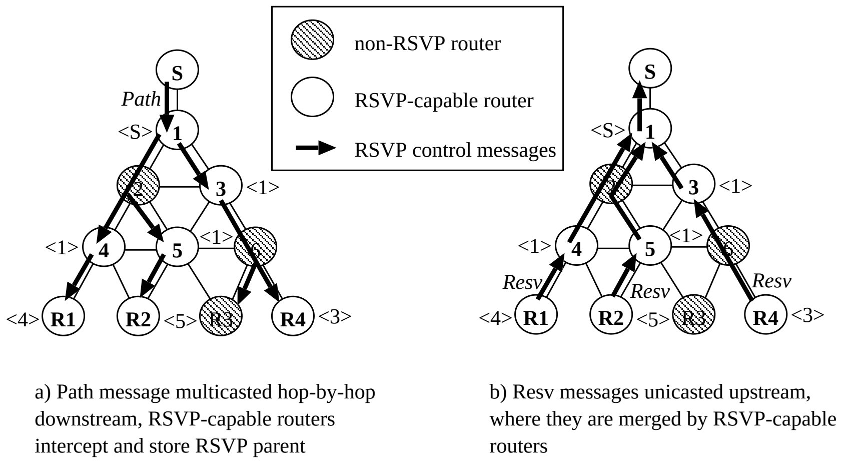

non-RSVP router
RSVP-capable router
RSVP control messages
Path
S
1
2
3
4
5
6
R1
R2
R3
R4
<S>
<1>
<4>
<5>
<3>
Resv
a) Path message multicasted hop-by-hop downstream, RSVP-capable routers intercept and store RSVP parent
b) Resv messages unicasted upstream, where they are merged by RSVP-capable routers

Figure 6.1: RSVP *Path* and *Resv* Messages

6.1 RSVP Overview

RSVP is designed to scale to very large multicast groups. It uses receiver-oriented reservation requests that merge as they progress up the multicast tree. The reservation for a single receiver does not need to travel to the root of a multicast tree; rather it travels only until it reaches a branch of the tree with an equal or larger reservation.

When forwarding reservations upstream toward the root of a multicast tree, RSVP needs to follow the route that multicast data would travel when sent by the root. To allow incremental deployment, RSVP must additionally cope with the possibility that some routers may not support the protocol. RSVP overcomes both of these obstacles by having an RSVP sender transmit a *Path* message downstream. The *Path* message is sent as a multicast packet that is forwarded normally by non-RSVP routers, but is intercepted at each RSVP router on the multicast tree for processing. Each intercepting RSVP router stores state needed to reach the root of the multicast tree, including the parent RSVP router.

Using this state, RSVP can then forward reservation messages upstream along the reverse path of the multicast tree. When a receiver wishes to make a reservation, it transmits a *Resv* message, which RSVP unicasts to the parent RSVP router. At each hop, RSVP attempts to make the requested reservation and returns an error to downstream receivers if the reservation fails. If the *Resv* meets an already-reserved portion of the multicast tree, it stops if the reservation in place is larger than or equal to its own. Otherwise, the *Resv* message is forwarded to the root of the tree.

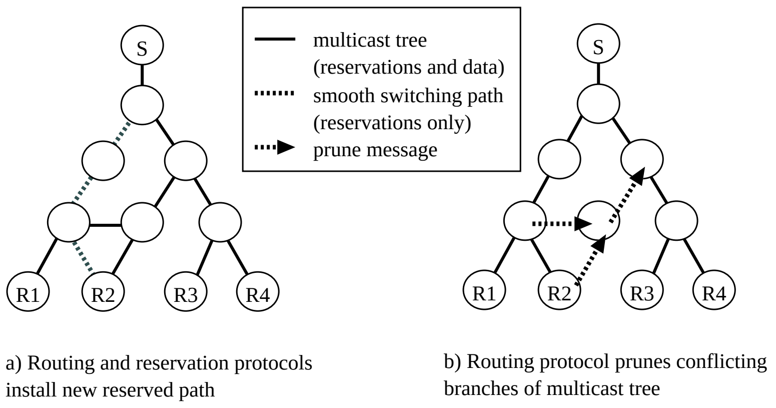
Figure 6.1 shows an example of RSVP message processing. In Figure 6.1a, routers 2 and 6 are automatically bypassed by the *Path* message as it is muilticasted hop-hop-by hop downstream. Note that the *Path* message is also received by non-RSVP receivers that subscribe to the multicast group, such as receiver R3. Each router in the figure is shown storing the parent RSVP router in the multicast tree. In Figure 6.1b, each RSVP receiver unicasts a *Resv* message to its parent RSVP router. These routers in turn unicast the message upstream toward the root, again bypassing non-RSVP routers. When multiple reservations are received at router 1, they are merged and a single reservation is forwarded to the root. Note that more merging would take place at router 2 if it was RSVP-capable.

6.2 Smooth Switching

Because a pinned (alternate) route does not adapt opportunistically, it may at times be longer than the shortest path. While any router can limit the amount of extra hops its pinned route uses, the use of paths longer than shortest paths may decrease network utilization. If alternate routes are commonly used, this may lead to significantly inefficient usage of network resources. To alleviate this problem, we may prefer that the architecture enables all receivers to migrate to the shortest path if adequate resources are available to support their reservations.

However, changing paths indiscriminately can cause a service disruption for receivers with a reservation. As with opportunistic routing changes, receivers have no guarantee that their reservations will be accepted on the shortest path once a switch is made, and even if they are, they face some period of time when there is no reservation state. To prevent service disruptions, we may require that the multicast routing protocol not switch to the shortest path until after a reservation has been established there. Furthermore, the reservation on the shortest path must be large enough so that it provides the same level of service as the old path to all affected receivers. We call this type of service *smooth switching*; receivers switched to the shortest path should experience a smooth transition, without service interruption.

Thus, with the smooth switching service, the routing and reservation protocols cooperate to migrate receivers from a reserved alternate path to a reserved shortest path. While the shortest path is being prepared, the reservation protocol keeps a reservation in place on both paths, but data flows only on the installed multicast tree (Figure 6.2a). When the transition is made, the shortest path is incorporated into the multicast tree and the old path is pruned from the tree (Figure 6.2b). At the same time, the shortest path is itself



multicast tree (reservations and data)
smooth switching path (reservations only)
prune message
S
R1
R2
R3
R4
a) Routing and reservation protocols install new reserved path
b) Routing protocol prunes conflicting branches of multicast tree

Figure 6.2: Smooth Switching Service

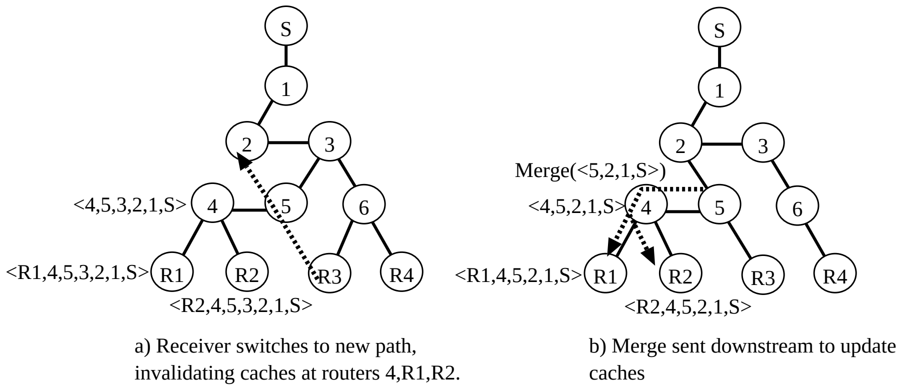


Figure 6.3: Smooth Switching Requires *Merge* Message

pinned, to prevent any future disruption from opportunistic route changes. Likewise, the reservation protocol drops its reservation along the old path.

Because smooth switching implies that pinned branches in the multicast tree can be re-routed, a more complex setup protocol than MORF is needed to perform this service. When the smooth switching service migrates from an old route to a new one, the ancestors cached within the re-routed subtree will become invalid (Figure 6.3a), thus allowing temporary loops to form. These nodes can either be removed from the tree (similar to a *Failure* with MORF) or have their caches updated (similar to a *Merge* with Merge Down). While MORF is appropriate for new branches being added to the tree, it can result in service disruptions when applied to this scenario. In addition, since the nodes with invalid caches can already participate in temporary loops, the loop-prevention advantage of MORF is lost. Using a *Merge* message updates nodes with corrected caches (Figure 6.3b) while keeping them attached to the tree. Unlike the Merge Down mechanism, however, the message needs to be sent down the re-routed subtree, rather than on the new branch.

The Smooth Switching protocol also has to incorporate additional complexity to ensure that the new branch is reserved at an adequate level. The Merge Down mechanism can be modified for this purpose by adding two additional messages: *Standby* and *Switch*. The *Standby* message travels upstream from a receiver to install and reserve a temporary branch in the multicast tree. The reservation amount is set so as to provide service to affected receivers that is equal to what they are currently receiving. When making the reservation on the smooth switching route, routers should coordinate with RSVP so that they can use any current reservations for the tree, i.e. where the smooth switching route overlaps the current route. Once the reservation is in place, a *Switch* message is sent downstream to make the temporary branch a permanent branch of the multicast tree. Any conflicting branches of the multicast tree are safely pruned because equivalent reservations on these branches have already been made on the new branch. It is important to switch the path on the way downstream, to avoid service disruption if a *Switch* message is lost between routers.

To begin smooth switching, a router generates a *Standby* message that contains the shortest path route from itself to the root of the multicast tree, along with the combined reservation of any local receivers. The router gets the shortest path route by querying its routing tables or probing the network (i.e. with *traceroute*), and it obtains the needed reservation objects from RSVP. The router then forwards the *Standby* message upstream toward the root of the multicast tree.

Figure 6.4 details the processing of the *Standby* message. Each router along the smooth switching path temporarily keeps extra state consisting of:

$route^{Standby}$	smooth switching (Standby) route
$parent^{Standby}$	parent (next hop) of Standby route
$child^{Standby}$	child (previous hop) of Standby route
$resv_{parent}^{Standby}$	reservation for parent of Standby route
$resv_{child}^{Standby}$	reservation for child of Standby route

Note that separate smooth switching state is kept for each multicast tree, but we omit the root and group in this notation for readability. Similar nomenclature is used for Setup Tree state and RSVP state.

```
Procedure Standby($child, root, group, route, resv$)
begin
    if ($\exists state^{Standby}$)
        if ($route^{Standby} \neq route$)
            send Failure($child, root, group, route, resv,$ “merge failure”)
        else
            if ($\neg$ end($route$))
                send Standby($parent^{Standby}, root, group, route, resv^{Standby}_{parent}$)
            endif
        endif
        return
    endif
    if ($resv$ rejected on $child$)
        send Failure($child, root, group, route, resv,$ “reservation failure”)
        return
    endif
    if (end($route$) $\vee$ (upstream($route$) $= route^{Setup} \wedge resv \leq resv^{RSVP}_{parent}$))
        $child^{RSVP} \leftarrow child^{RSVP} + child$
        $resv^{RSVP}_{child} \leftarrow max(resv, resv^{RSVP}_{child})$
        $child^{Setup} \leftarrow child^{Setup} + child$
        send Switch($child, root, group, route, resv$)
        return
    endif
    create $timer^{Setup}$
    $child^{Standby} \leftarrow child$
    $parent^{Standby} \leftarrow$ nexthop($route$)
    $route^{Standby} \leftarrow route$
    $resv^{Standby}_{parent} \leftarrow max(resv, resv^{RSVP}_{parent})$
    $resv^{Standby}_{child} \leftarrow resv$
    send Standby($parent^{Standby}, root, group, route, resv^{Standby}_{parent}$)
    return
end
```

Figure 6.4: *Standby* Message Processing for Smooth Switching Mechanism

The first thing a router must do when processing a *Standby* message is prevent the messages from crossing paths or merging. Several receivers may utilize smooth switching for the same multicast tree at the same time only if the routes being switched do not overlap. This simplifies the smooth switching mechanism. An alternative would be to allow *Standby* messages to create a Standby tree in parallel to the multicast tree, and to use MORF or Merge Down to prevent loops in this tree. If this were the case, then the largest reservation level would need to be carried upstream whenever a *Standby* message merged. Moreover, additional mechanism would be needed whenever a *Switch* message had already started to switch over to a new route when a *Standby* message merged downstream with a different route or a larger reservation. The mechanism described herein avoids this complexity.

The next processing step a router takes is to attempt the reservation listed in the *Standby* message. When attempting the reservation, the Smooth Switching mechanism coordinates with RSVP so that it can use any current reservations for the tree, i.e. where the smooth switching route overlaps the current route. If the reservation succeeds, then the *Standby* message is forwarded upstream. As the message travels upstream, it attempts to reserve the Standby route using the largest reservation of any of the branches that it is re-routing. This ensures that the reservation level is not downgraded for any of the receivers that are re-routed. However, during the delay it may take to switch to the new route, any upgrade of the reservation on the old route may be lost. To avoid this problem, a node may reject any upgraded reservation requests while a Standby route has not yet been smoothly switched.

The *Standby* message stops being forwarded if it reaches the end of its route. The *Standby* message may also stop if its route matches what is in place on the Setup Tree and if its reservation level is less than or equal to the reservation maintained by RSVP. In either case, the *Standby* message stops and the router sends a *Switch* message downstream.

Figure 6.5 details the processing of the *Switch* message. If the parent for the Standby route differs from that of the Setup Tree, then the router must prune the conflicting branch of the Setup Tree. It must also send a *Merge* message downstream to all children in the Setup Tree to prevent loops.

As the *Switch* message travels downstream, it deletes the smooth switching state. The routing state is converted to Setup Tree state and the reservation state is relinquished to RSVP. From this point on, MORF and RSVP assume responsibility for maintaining their respective state. If a *Standby* message is lost, the originating router will not receive a *Switch* message. If this occurs, then this router may re-transmit the *Standby* message

```
**Procedure Switch**$(parent, root, group, route)$
**begin**
    **if** $(\nexists state^{Standby})$
        **return**
    **endif**
    **if** $(route \neq route^{Standby})$
        **return**
    **endif**
    **if** $(parent^{Setup} \neq parent^{Standby})$
        **send Prune**$(parent^{Setup}, root, group)$
        **foreach**$(child \in child^{Setup})$
            **send Merge**$(child, root, group, route^{Setup}, \mathbf{upstream}(route^{Standby}))$
        **endfor**
    **endif**
    $parent^{RSVP} \leftarrow parent^{Standby}$
    $child^{RSVP} \leftarrow child^{RSVP} + child^{Standby}$
    $resv_{parent}^{RSVP} \leftarrow resv_{parent}^{Standby}$
    $resv_{child}^{RSVP} \leftarrow resv_{child}^{Standby}$
    $route^{Setup} \leftarrow \mathbf{upstream}(route^{Standby})$
    $parent^{Setup} \leftarrow parent^{Standby}$
    $child^{Setup} \leftarrow child^{Setup} + child^{Standby}$
    $child \leftarrow child^{Standby}$
    delete $state^{Standby}$
    **if** $(\mathbf{start}(route))$
        **return**
    **endif**
    **send Switch**$(child, root, group, route)$
**end**
```

Figure 6.5: *Switch* Message Processing for Smooth Switching Mechanism

after a period of time. State created by the mechanism will eventually timeout if no re-transmission is made.

We have described the smooth switching mechanism using combined signalling, since this reduces the signalling time. This is important for this mechanism, since competing *Standby* messages and upgraded RSVP reservations will be blocked during the switching time. Separating the routing and reservation mechanisms in this function would require three round-trip times to do the signalling instead of one: one trip to setup the path, one to make the reservations, and one to switch the path. In addition, combining the mechanisms makes sense because the routing state created by the mechanism is only needed temporarily to carry reservations upstream; it is not needed for data. Because smooth switching re-routes multicast trees, we assume the combined signalling will be provided by the route setup protocol, with an interface to RSVP to obtain reservations.

One drawback of the mechanism is that receivers are susceptible to a *one-upmanship* problem:

1. Assume Router A has reserved a route.
2. Router B overrides the route using smooth switching with an incrementally larger reservation.
3. Router A retaliates by returning to the original route using smooth switching and an incrementally larger reservation than B's new one. Etc...

While it is not clear whether there is an incentive for routers to exhibit this behavior, well-behaved routers could eliminate this problem by only using shortest-path routes in smooth switching. This is a reasonable restriction since the service is designed to improve network utilization by reducing path lengths.

6.3 Advance Advertising

Before making a reservation, some applications may prefer to know the end-to-end characteristics of a route. RSVP currently collects *advertising* information that describes the service characteristics of the current multicast tree [SS95, BZB$^+$97]. Having this information allows the application to determine, before making a reservation, whether the current route suits its needs. For instance, an interactive application may want to know the latency along the path before deciding if the route could provide the required low-latency service.

While RSVP's current advertising capability is limited to the currently-installed route, both routing and RSVP may want to use *advance advertising* to collect information along routes that are not yet installed. For example, if a receiver's reservation is rejected along the current path, then either RSVP or a route server may want to determine whether an alternate path is appropriate (e.g., has low latency) for a receiver before the route setup protocol installs the path. Similarly, when routing is already using an alternate path, RSVP or a route server could use this service to probe the shortest path and determine whether it should initiate smooth switching. Using advance advertising does not affect the forwarding of data. Note that advance advertising does not give dynamic information about whether a reservation request will be accepted; it only passes on the relatively static parameters advertised by routers (such as latency).

Within a single multicast group, differing receivers may want to obtain advance advertising over differing paths. Some may be trying various alternate paths, while others may be checking the shortest path. While each node in the tree can use only one path to the sender, it is not possible to know ahead of time which of the advance advertising requests will find a path RSVP can utilize. Thus, merging advance advertising requests into a single tree is impractical. Likewise, routing can not install each receiver's route, as this would result in $O(n^2)$ state. Therefore, advance advertising should use combined signalling, where a single *Collect* message carries the explicit route being probed and collects the advertising information while traversing the route. Because advance advertising does not affect the route used by a multicast tree, this combined signalling should be provided by RSVP. Since RSVP may not be available at each router for processing of the *Collect* message, a receiver will first need to probe the network to determine which routers along the target path support RSVP.

While in many cases the node initiating advance advertising will be the receiver or its last-hop router, this may not always be true. For example, a route server may check the characteristics of a route as part of its route computation process. The route server may not even be a part of the route being probed and thus would need to turn on and off collection of advertising information (Figure 6.6).

To maintain flexibility when collecting advertising information, the *Collect* message carries a *routing script*. In addition to listing the hops in the route and a counter for the current hop, the routing script contains a bitmask that can turn on and off the collection of advertising information. By setting the bitmask in the script, a router can collect advertising information from any of the hops listed in the route. This functionality is similar to that of GRE explicit routing [HLFT94]. For instance, the route server in Figure 6.6

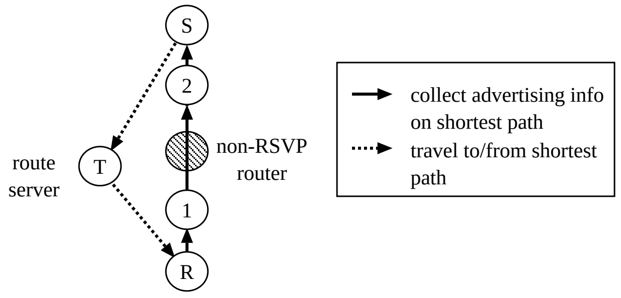

S
2
non-RSVP router
route server
T
1
R
collect advertising info on shortest path
travel to/from shortest path

Figure 6.6: Advance Advertising Service

```
Procedure Collect($root, group, route, adv$)
begin
    if (end($route$))
        return
    endif
    if (bitmaskset($route, myIPaddress$))
        $adv \leftarrow adv +$ advertising info
    endif
    send Collect(nexthop($route$), $root, group, route, adv$)
end
```

Figure 6.7: Advance Advertising Mechanism

would use a routing script with an explicit route of $< T, R, 1, 2, S, T >$ and a collection bitmask of $< 001110 >$. This would collect advertising information for traffic being sent from S to R from routers 1, 2 and S.

Figure 6.7 details the processing of a *Collect* message. At each hop, the router examines the bitmask to determine if it should collect advertising information at that hop. If it should, then it adds this information for the link leading to the previous hop in the route. After appending any advertisements, the router continues forwarding the message. The last hop listed in the routing script should be the router that originated the *Collect* message, so that processing stops where it began.

Any route can be used with advance advertising as long as it is encoded as an explicit route, listing all the hops that will be traversed. For example, a router may want to determine which route it would ultimately be using if it tried to join the multicast tree with a given alternate path. It can use MORF to do this type of probing by carrying a *Setup* message from the receiver until it reaches the rest of the Setup Tree, then automatically generating a *Failure* message, regardless of whether the route matches. The router can determine which nodes on this route support RSVP, then use advance advertising to determine resource availability at those nodes.

6.4 Sender Deactivation

RSVP incorporates different reservation styles that determine how a reservation applies to the senders in a multicast group. The dynamic filter style[1] allows a receiver to make a reservation for a set of senders, then select which senders use that reservation over time. The reservation is always kept in place for all senders, regardless of whether their data is needed. This service is useful for receivers that are limited by cost or by a bottleneck link to receiving only a few streams at a time. For example, the receiver may want to use a dynamic filter reservation to switch among the available speakers in a video conferencing application. The dynamic filter style ensures that when the user wants to watch a new speaker, a reservation will already be in place for the speaker.

The original design of RSVP called for routers to drop the data of the senders that are not currently selected by downstream receivers. However, the designers soon realized that this would prevent non-RSVP receivers from getting data they wanted, i.e. an RSVP-capable receiver could deny non-RSVP receivers from watching a particular sender in a videoconference. Likewise, routers should not always forward all the data from all of the senders. In the case where there aren't any receivers that want the data – RSVP-capable or otherwise – the unneeded data could overburden downstream links.

In keeping with the relative roles of RSVP and routing, some routing mechanism is thus needed to drop the data of unwatched senders as soon as possible. Pruning the multicast tree of an unwatched sender is not acceptable, as this would prevent RSVP from maintaining its reservation for the sender. We suggest that routing can instead use a similar service called *sender deactivation*. When a receiver uses this service on a branch of a multicast tree, the multicast routing protocol *deactivates* the sender by no longer forwarding its data over the branch. The multicast routing protocol still retains its own protocol state so that it can help RSVP to send control messages along the deactivated branch. This in turn allows RSVP to use the dynamic filter reservation style while only forwarding data where it is needed.

The sender deactivation service requires cooperation between hosts, RSVP, and routing. First, hosts (both RSVP and best-effort) need to report to routers their interest in

[1]This is described in the original paper [ZDE+93], but is not part of the current specification. [BZB+97]

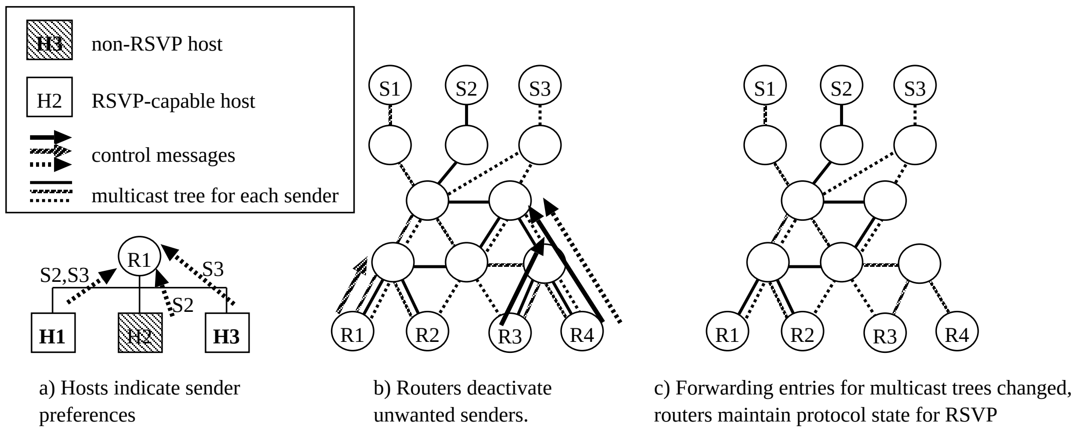

H3 non-RSVP host
H2 RSVP-capable host
control messages
multicast tree for each sender
S2,S3
R1
S3
S2
H1
H2
H3
a) Hosts indicate sender preferences
S1
S2
S3
R1
R2
R3
R4
b) Routers deactivate unwanted senders.
c) Forwarding entries for multicast trees changed, routers maintain protocol state for RSVP

Figure 6.8: Sender Deactivation Service

specific senders to the multicast group. Extensions to IGMP that provide this functionality are already being developed [CDT96]. Second, an RSVP router that receives a dynamic filter reservation needs to tell routing that it wants to use sender deactivation for that multicast tree. When all hosts on a link have indicated that they are not interested in data from a given sender, then the router on that link can deactivate forwarding for that sender using a simple multicast routing protocol extension.

Figure 6.8 shows how sender deactivation works. In Figure 6.8a, each of the hosts indicates to router R1 the senders they want to receive data from. In this example, none of the hosts are interested in sender S1. If RSVP indicates to R1 that it wants to use sender deactivation, then R1 deactivates forwarding for S1 (Figure 6.8b). Likewise, router R3 deactivates sender S2 and R4 deactivates both S2 and S3. After these actions, the multicast trees for each sender are deactivated up to the point where other receivers need the data (Figure 6.8c). The routers continue to maintain protocol state internally so that RSVP may forward *Path* and *Resv* messages along the trees.

Figure 6.9 details how the multicast routing protocol supports sender deactivation using two new messages: *De-Activate* and *Re-Activate*. When hosts tell a router they do not want data from a sender, the router sends a *De-Activate* message upstream. The *De-Activate* message operates like a multicast routing prune message, except that at each hop it leaves routing protocol state in place and only removes state from the forwarding engine. Later, if hosts want data from the sender, a router can re-install the forwarding state along the branch by sending a *Re-Activate* message upstream.

```
Procedure De-Activate($child, sender, group$)
begin
    if ($\nexists tree^{Multicast}$)
        return
    endif
    if ($child \notin child^{Multicast} \vee status_{child}^{Multicast} = deactivated$)
        return
    endif
    if (local receivers interested in sender)
        return
    endif
    $status_{child}^{Multicast} \leftarrow deactivated$
    remove child from forwarding entry
    if ($status_{parent}^{Multicast} = deactivated$)
        return
    endif
    foreach($c \in child^{Multicast}$)
        if ($status_{c}^{Multicast} = activated$)
            return
        endif
    endfor
    remove $parent^{Multicast}$ from forwarding entry
    $status_{parent}^{Multicast} \leftarrow deactivated$
    send De-Activate($parent^{Multicast}, sender, group$)
    return
end

Procedure Re-Activate($child, sender, group$)
begin
    if ($child \notin child^{Multicast} \vee status_{child}^{Multicast} = activated$)
        return
    endif
    $status_{child}^{Multicast} \leftarrow activated$
    add $child$ to forwarding entry
    if ($status_{parent}^{Multicast} = activated$)
        return
    endif
    $status_{parent}^{Multicast} \leftarrow activated$
    add $parent^{Multicast}$ to forwarding entry
    send Re-Activate($parent^{Multicast}, sender, group$)
    return
end
```

Figure 6.9: Sender Deactivation Mechanism

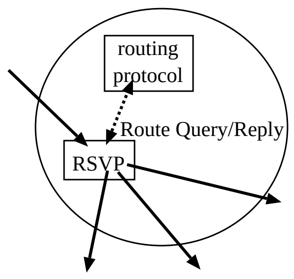


Figure 6.10: Route Change Notification Service

6.5 Route Change Notification

When RSVP is used over opportunistic routing, it can adapt to routing changes using periodic refreshes of its *Path* messages. At each router on the multicast tree where RSVP processes a Path message, it sends a *Route Query* to the routing protocol to get the latest routing entry. In response, RSVP receives a *Route Reply*, listing the children for the delivery of the *Path* message. RSVP then sends the Path message to those children (Figure 6.10). The speed by which RSVP can adapt to routing changes using this mechanism is limited to the refresh timer period – a default of 30 seconds.

One of the early directions of this dissertation was to develop the *Route Query* and *Reply* interface and then to determine how to improve RSVP's response time to route changes. Because RSVP also uses *Path* message refreshes to keep its state alive, using them to discover route changes overloads their functionality. If RSVP uses a short refresh timer so that it responds quickly to route changes, then its state maintenance overhead increases correspondingly. Likewise, lengthening the timer decreases overhead but worsens adaptivity.

We developed a *route change notification* service by which routing notifies RSVP immediately as route changes occur. This frees RSVP from discovering route changes, allowing it to adjust the refresh timer solely to reduce messaging overhead. RSVP can also use the route change notification to trigger an immediate refresh of a *Path* message and adapt to the new route. Then when normally processing a refresh timer, RSVP can assume the routing entry it has is still current and does not need to query routing each time.

This service is extremely simple; no new routing messages are required, and the multicast routing protocol needs to maintain only a single bit of information (indicating notification is either on or off) for each routing entry. To use the service, RSVP sets a bit in the *Route Query* message indicating that it wants route change notification. It does this at each hop where it has received a *Path* message. Then, when a route change occurs, routing sends an unsolicited *Route Reply*, in addition to what was immediately sent in response to the *Route Query*. This *Route Reply* lists the new set of children when the route has changed.

Route change notification decreases the recovery time for RSVP after a route change by up to t_r, RSVP's refresh rate. Assume the routing protocol provides route change notification at time t_0. This triggers an RSVP *Path* message that travels n hops along the new route with propagation time t_p for each hop. Each *Path* message also incurs t_q time for a route query at each hop, except for the last hop. Once the *Path* message reaches the receiver or some downstream reserved portion of the multicast tree, RSVP sends a *Resv* message upstream. This also travels n hops with propagation time t_p for each hop. Thus the total time for RSVP to recover its state after a route change is

$$t = 2n * t_p + (n - 1) * t_q$$

Without route change notification, RSVP has to discover route changes by periodically sending *Path* messages. The speed with which it finds route changes depends directly on its refresh period t_r. Thus, without route change notification, the worst-case time for RSVP to recover its state after a route change is

$$t' = 2n * t_p + n * t_q + t_r$$

Therefore route change notification speeds up RSVP state recovery by at most

$$\Delta_t = t' - t = t_q + t_r \approx t_r$$

Experimentation with a prototype implementation of route change notification has confirmed its effectiveness.

6.6 Evaluation

Table 6.1 summarizes the costs of each of the services discussed in this chapter. Of these services, route change notification has the lowest cost since it only uses local messages exchanged within a single router between the routing protocol and RSVP. We have implemented this service in the ISI RSVP prototype and in the DVMRP multicast routing protocol used in the MBone.

Smooth switching is the most costly of all of the mechanisms, as it requires MORF to send a *Merge* message down every re-routed subtree. This means that the route setup protocol may form temporary loops during re-routing, decreasing the effectiveness of Match

Table 6.1: Mechanism Costs

Service	**Costs**
Smooth Switching	MORF must send *Merge* message on re-routed subtrees, add additional *Standby* and *Switch* messages.
Advance Advertising	RSVP must add *Collect* message.
Sender Deactivation	Multicast routing protocol must add *De-Activate* and *Re-Activate* messages, keep sender-specific routing entries. Hosts and routers must add IGMPv3 extensions.
Route Change Notification	Routing protocol must accept local Route Queries, keep an extra bit for routing entries.

or Fail in preventing loops. The route setup protocol must also add *Standby* and *Switch* messages to coordinate the re-routing. The cost of this mechanism likely outweighs the utility of being able to switch to shortest paths. Many of the same benefits of smooth switching could be realized by instead limiting the path length of alternate routes as well as the use of alternate path routing during high load. These techniques have been shown to improve network utilization in telecommunications networks [KZ89, Aki84] and could likely be applied to alternate path routing in the Internet as well.

For intradomain routing, however, smooth switching is a viable mechanism for installation of QoS routes. The mechanistic premise of smooth switching – that a receiver may re-configure the multicast tree used by other receivers – is identical to that of QoS routing. QoS routing assumes that a receiver with a "larger" QoS requirement than that of other receivers may re-route those receivers, assuming it can find a path that will satisfy all of them. When group membership is dynamic and receivers wish to independently compute routes with available resources, the Smooth Switching protocol discussed herein offers a scalable method for receiver-initiated installation of QoS routes, as contrasted with the sender-oriented mechanisms often proposed in the literature.

Advance advertising is a comparatively low cost service. It requires only an additional message in RSVP for collecting the advertising information, and it does not create state in routers. This is a small cost to pay for allowing routers to determine service characteristics of routes before installing them, provided the frequency with which routers issue the message is low. Ultimately the adoption of this service depends on whether alternate path routing is prevalent and whether service characteristics are needed to compute a route for a receiver.

Sender deactivation uses two relatively simple routing messages, but requires sender-specific routing entries, which adds extra protocol state for multicast routing protocols that use shared trees. In addition, the utility of the service must justify implementation of enhanced IGMP features. Whether sender deactivation is implemented depends largely on whether RSVP receivers need the dynamic filter reservation style. Similar service can be provided by sequentially installing and removing a set of *fixed filter* (or sender-specific) reservations, without the assurance that a reservation for a newly activated sender will be accepted by the network. Whether this alternative is feasible depends in turn on resource availability within the network.

Chapter 7

Contributions and Future Work

We have designed an architecture and multicast route setup protocol for alternate path routing and route pinning. These two services are fundamental in the sense that they provide an alternative to shortest path routing and opportunistic routing, respectively. As such they are useful for supporting both adaptive, best-effort real-time applications as well as those that use a reservation protocol.

Our design of the MORF multicast route setup protocol is novel in that it is receiver-oriented and uses explicit routes to prevent looping. Other setup protocols have primarily used sender-oriented setup mechanisms or required nodes in the multicast tree to know their downstream neighbors. By avoiding storage or processing requirements that scale with the number of downstream nodes, MORF is more scalable than its predecessors. We have contrasted the MORF loop prevention scheme with other methods for detecting and breaking loops. The MORF scheme may result in a slightly longer setup delay, but this delay is of lesser importance when establishing an alternate path because the use of such a path implies a failure of some type has already occurred. We have also compared our route pinning design to the alternative of following hop-by-hop the currently-installed route. This latter method has been used recently for a unicast route pinning protocol [GKH97] and the authors have contemplated extending it to multicast. Our work has illustrated the pitfalls of using this design for receiver-oriented multicast.

Our simulation study has verified our approach to constructing alternate paths, namely that routers near the members of a multicast group can find paths without global distribution of the topology. We have identified several techniques with a high success rate for finding paths. In addition, all of the techniques we simulated find paths that are comparable in length to the shortest path. Combined with limits on path length imposed by the route construction agent, these heuristics allow routers to exploit the redundancy of large networks. We have also demonstrated that the topology model used for our simulations can have a large impact on the results obtained, further confirming the work of Zegura et al. on topology generation [ZCB96, ZCD].

We have made the first known attempt to design extended routing services for RSVP. While the idea of smooth switching has been discussed for several years, we have identified its mechanistic costs within the MORF framework. The overall complexity of the mechanism is high and we do not foresee it being implemented at the interdomain level. In contrast, we have found that advance advertising requires no additional state within routers and can be implemented within RSVP using a simple collection script. We have also identified the costs of supporting RSVP's dynamic filter reservation style. We have tested and implemented route change notification, and it is currently deployed in the MBone.

Should the Internet engineering community decide to adopt MORF, further work will be needed to completely specify it and test it. Our design includes extensions for robustness, multi-access subnets, and bundling. One of the key decisions to be made is whether fast teardown is needed or the base specification is enough. If adopted, any of these additional features need to be integrated into the basic setup mechanism, along with methods for refreshing the state, to form a complete specification. Other setup protocols that use refresh mechanisms (i.e. PIM and RSVP) use a refresh message to both keep state alive and capture routing changes. However, use of refreshes for MORF is limited only to keeping state alive; route changes are ignored and route failures result in the installed route being removed. Additional work is needed to interface MORF with the multicast routing protocol so that when a pinned route is removed, the multicast routing protocol quickly re-installs an opportunistic route.

The future use of MORF will also depend on the design of host services and an API from the host to the routers to request alternate path setup and route pinning. Our routing and reservation architecture includes a QoS manager to handle service issues for applications. The QoS manager needs methods for measuring the service supplied to a user, such as measuring packet delays, monitoring reservation requests, and perhaps receiving qualitative input from the user. Further work will also be needed to refine the heuristics the QoS manager uses to decide which actions it will take. For example, if packet loss is high, the QoS manager may choose from many different actions. These may include making a reservation or asking for an alternate path. For an application using hierarchically encoded video, the manager may also choose to leave a multicast group for a high level of encoding. All of these actions are adaptive responses to poor performance, and it is not clear yet which choice to make in a given situation and how the QoS manager would make that choice. Additional work is also needed to develop an API by which the QoS manager asks MORF to install an alternate path, as well as provide identify links that should be avoided (i.e. due to congestion or reservation failure).

Our investigation of route construction has focused on using several heuristics that do not require extensive modification to the current routing infrastructure. One item of future work is to design a prototype of a route construction agent and deploy it to find alternate paths. The agent can use the *traceroute* or *mtrace* utility to gather its partial map, pending routing protocol support of remote table lookup. A complete agent design will need to maintain its partial map of the network, periodically deleting old information and polling for new nodes. The agent will also need to use an API to communicate with the QoS manager regarding bottleneck links and reservation failures, as well as updated status reports when links become uncongested or reservations succeed.

In addition, our study of route construction heuristics has focused on their applicability to a single sender and receiver. Further study is needed to determine the benefits gained when the costs of route construction are amortized in two different ways. First, the local topology exploration done by a router will help that router find alternate paths for a number of senders. Second, when one router finds an alternate path around a bottleneck, installation of that path will help all group members that are downstream of the bottleneck. Additional simulations can quantify the amortization gains for these cases, and deployment of an agent prototype can verify those results in the field. The results will be highly dependent on group distribution, so a simulation study will need to include diverse characterizations of sender and receiver locations.

While the costs of the smooth switching protocol are likely to be too high for interdomain routing, the mechanisms needed are identical to those for installing a QoS route of higher priority (or larger QoS) than those already in place. Similarly, the advance advertising mechanism may be of limited usefulness if it reports only static link characteristics, but if it instead is used to collect current resource availability it may also be useful for QoS routing. One area for future work is to use these two mechanisms to construct a QoS routing mechanism that does not rely on global distribution of network information. To determine its benefits, one could conduct an exhaustive study of a spectrum of route computation scenarios generated by combining varying amounts of knowledge about resource availability:

- none,
- local resource availability (i.e. on links within n hops),

- partial resource availability (i.e. polled with advance advertising), and
- full resource availability,

with both full maps of the topology and partial maps using the topology exploration heuristics of this dissertation. A study of these scenarios could examine their relative tradeoffs of overhead versus route quality and setup delay.

Our study of the sender deactivation mechanism indicates that it has a low overhead and thus merits more study to determine its utility. One method for doing this would be to compare the use of the dynamic filters RSVP style with sender deactivation versus installing and removing a succession of *fixed filter* (or sender-specific) reservations. To get similar functionality as the dynamic filter style, a receiver would need to make rapid changes of fixed filter reservations with a low blocking rate. The success of this approach will clearly depend on the load of the network. It will also require future work studying the ability of multicast routing protocols to perform fast joining and leaving of a sender's multicast tree and likewise the ability of RSVP to provide fast installation and removal of reservations. Work is already underway within the Internet engineering community to design a sender-specific version of IGMP. Of particular importance are the suppression of duplicate reports and the tradeoff between having hosts report on those senders they wish to receive data from versus those they would specifically like to exclude.

All of the protocol design in this dissertation has assumed that routing and reservation protocols manage state on behalf of individual multicast trees. Many in the Internet community have suggested designing multicast routing and reservation protocols that do not require per-group state but instead use aggregates of such state, at least within the backbone of the network. Some researchers have recently suggested a new network design called differentiated services (as opposed to integrated services). With this scheme, applications mark their packets based on delay preference and/or drop preference, and the network schedules the packets using class-based queueing [FJ95]. As the applications detect congestion, they adjust the number of packets they assign to each category. Another possible design is to modify the reservation protocol so that it carries contracts between institutional entities rather than flow-specific reservation requests. The contract could then be used to govern the aggregated traffic flow between the entities. Both of these potential designs need further study, and it is likely that future routers will handle both best-effort and real-time traffic, as well as aggregated traffic and traffic with individual service commitments.

Finally, future work is needed to improve the tools available for large-scale simulations. For this work, we pushed the limits of currently available network simulators by running tens of thousands of short simulations on topologies of thousands of nodes. Completing these simulations required extensive memory and performance profiling to optimize the simulation, often necessitating movement of code from the Tcl interface to the C++ engine. Developing larger-scale simulations often means making a tradeoff between specifying less information versus scaling to a larger problem solving environment. To develop such tools, for both programming and visualization, the network research community may need to incorporate ideas from the artifical intelligence and parallel processing communities. For example, artificial life researchers can study the effects of genetic mutations on large populations or examine group behavior within large communities of animals. Similarly, the network community needs a simulation environment that allows the user to specify simple, abstract behaviors and study those behaviors in a large-scale setting.

Reference List

[AH93] Gerald R. Ash and BaoSheng D. Huang. "An Analytical Model for Adaptive Routing Networks". *IEEE Transactions on Communications*, 41(11), November 1993.

[Aki84] J. M. Akinpelu. "The Overload Performance of Engineered Networks With Nonhierarchical and Hierarchical Routing". *AT&T Bell Laboratories Technical Journal*, 63(7), September 1984.

[AKK81] G. H. Ash, A. H. Kafker, and K. R. Krishnan. "Servicing and Real-Time Control of Networks with Dynamic Routing". *Bell System Technical Journal*, 60(8), October 1981.

[Ala94] Cengiz Alaettinoglu. *"Scalable Inter-Domain Routing with ToS, Policy and Topology Resolution"*. PhD thesis, University of Maryland, 1994.

[Alm92] P. Almquist. "Type of Service in the Internet Protocol Suite". RFC 1349, July 1992.

[Att81] R. Attar. "A Distributed Adaptive Multi-Path Routing - Consistent and Conflicting Decision Making". In *Fifth Annual Berkeley Workshop on Distributed Data Management and Computer Networks*, February 1981.

[Bal97] A. J. Ballardie. "Core Based Trees (CBT) Multicast Routing Architecture". work in progress, May 1997.

[BCFG$^+$97] Tom Billhartz, J. Bibb Cain, Ellen Farrey-Goudreau, Doug Fieg, and Stephen Gordon Batsell. "Performance and Resource Cost Comparisons for the CBT and PIM Multicast Routing Protocols". *IEEE Journal on Selected Areas in Communications*, 15(3), April 1997.

[BCS94] R. Braden, D. Clark, and S. Shenker. "Integrated Services in the Internet Architecture: an Overview". RFC 1633, June 1994.

[BDG91] Rick Bubenik, John DeHart, and Mike Gaddis. "Multipoint Connection Management in High Speed Networks". In *IEEE INFOCOM*, 1991.

[BFC93] A. J. Ballardie, P.F. Francis, and J. Crowcroft. "Core Based Trees". In *ACM SIGCOMM*, August 1993.

[BFG$^+$95] R. Bettati, D. Ferrari, A. Gupta, W. Heffner, W. Howe, M. Moran, Q. Nguyen, and R. Yavatkar. "Connection Establishment for Multi-Party Real-Time Communication". In *Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, April 1995.

[BJR97] A. J. Ballardie, N. Jain, and S. Reeve. "Core Based Trees (CBT version 2) Multicast Routing: Protocol Specification". work in progress, July 1997.

[BKJ83] Kadaba Barath-Kumar and Jeffrey M. Jaffe. "Routing to Multiple Destinations in Computer Networks". *IEEE Transactions on Communications*, 31(3), March 1983.

[Bre95] Lee Breslau. *"Adaptive Source Routing of Real-Time Traffic in Integrated Services Networks"*. PhD thesis, University of Southern California, December 1995.

[BZB$^+$97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification". work in progress, March 1997.

[CD92] S. Casner and S. Deering. "First IETF Internet Audiocast". *ACM SIGCOMM Computer Communication Review*, 22(3), July 1992.

[CDT96] Brad Cain, Steve Deering, and Ajit Thyagarajan. "Internet Group Management Protocol Version 3 (IGMP V.3)". Work in progress, 1996.

[CGS93] Israel Cidon, Inder S. Gopal, and Adrian Segall. "Connection Establishment in High-Speed Networks". *IEEE/ACM Transactions on Networking*, 1(4), August 1993.

[Cho91] C-H. Chow. "On Multicast Path Finding Algorithms". In *IEEE INFOCOM*, 1991.

[Cla88] David D. Clark. "The Design Philosophy of the DARPA Internet Protocols". In *ACM SIGCOMM*, August 1988.

[CZ] Ken Calvert and Ellen Zegura. "Georgia Tech Internetwork Topology Models". http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html.

[DB95] L. Delgrossi and L. Berger. "Internet Stream Protocol Version 2 (ST2): Protocol Specification Version ST2+". RFC 1819, August 1995.

[Dee88a] Stephen Deering. "Host Extensions for IP Multicasting". RFC 1054, May 1988.

[Dee88b] Stephen Deering. "Multicast Routing in Internetworks and Extended LANs". In *ACM SIGCOMM*, August 1988.

[Dee91] Stephen Deering. *"Multicast Routing in a Datagram Internetwork"*. PhD thesis, Stanford University, 1991.

[DEF$^+$94] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. "An Architecture for Wide-Area Multicast Routing". In *ACM SIGCOMM*, August 1994.

[DEF$^+$97] Steven Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ahmed Helmy, and Liming Wei. "Protocol Independent Multicast Version 2, Dense Mode Specification". work in progress, May 1997.

[DFE$^+$95] S. Deering, B. Fenner, D. Estrin, A. Helmy, D. Farinacci, L. Wei, M. Handley, V. Jacobson, and D. Thaler. "Hierarchical PIM-SM Architecture for Inter-Domain Multicast Routing". work in progress, December 1995.

[DH95] S. Deering and R. Hinden. "Internet Protocol, Version 6 (IPv6) Specification". RFC 1883, December 1995.

[DHHS93] Luca Delgrossi, Ralf Guido Herrtwich, Frank Oliver Hoffmann, and Sibylle Schaller. "Receiver-Initiated Communication with ST-II". Technical Report 43.9314, IBM European Networking Center, 1993.

[DL93] Matthew Doar and Ian Leslie. "How Bad Is Naive Multicast Routing?". In *IEEE INFOCOM*, 1993.

[DT95] Stephen E. Deering and Ajit Thyagarajan. "Hierarchical Distance Vector Multicast Routing for the MBONE". In *ACM SIGCOMM*, August 1995.

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

[ERH92] Deborah Estrin, Yakov Rekhter, and Steve Hotz. "A Scalable Inter-Domain Routing Architecture". In *ACM SIGCOMM*, August 1992.

[Eri94] Hans Eriksson. "MBONE: The Multicast Backbone". *Communications of the ACM*, 37(8), August 1994.

[ES91] Deborah Estrin and Martha Steenstrup. "Inter Domain Policy Routing: Overview of Architecture and Protocols". *"Communications of the ACM"*, 21(1), January 1991.

[EZL$^+$96] Deborah Estrin, Daniel Zappala, Tony Li, Yakov Rekhter, and Kannan Varadhan. "Source Demand Routing: Packet Format and Forwarding Specification (Version 1)". RFC 1940, May 1996.

[FBZ94] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. "Network support for multimedia: A Discussion of the Tenet Approach". *Computer Networks and ISDN Systems*, 1994.

[FJ95] Sally Floyd and Van Jacobson. "Link-sharing and Resource Management Models for Packet Networks". *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[For79] J. Forgie. "ST - A Proposed Internet Stream Protocol". Internet Experimental Notes IEN-119, September 1979.

[FS94a] Inc. FORE Systems. "SPANS NNI: Simple Protocol for ATM Network Signalling (Network-to-Network Interface) Release 3.0". Technical report, FORE Systems, Inc., 1994.

[FS94b] Inc. FORE Systems. "SPANS UNI: Simple Protocol for ATM Network Signalling (User-to-Network Interface) Release 3.0". Technical report, FORE Systems, Inc., 1994.

[GKH97] R. Guerin, S. Kamat, and S. Herzog. "QoS Path Management with RSVP". work in progress, March 1997.

[GKK88] R. J. Gibbons, F. P. Kelley, and P. B. Key. "Dynamic Alternative Routing - Modelling and Behavior". In *Proceedings of the 12 International Teletraffic Congress*, June 1988.

[GKR97a] R. Guerin, S. Kamat, and E. Rosen. "Extended RSVP-Routing Interface". work in progress, July 1997.

[GKR97b] R. Guerin, S. Kamat, and E. Rosen. "Setting up Reservations on Explicit Paths using RSVP". work in progress, July 1997.

[Hed88] C. Hedrick. "Routing Information Protocol". RFC 1058, STD 0034, June 1988.

[Hed89] Charles L. Hedrick. "An Introduction to IGRP". Technical report, The State University of New Jersey, October 1989.

[HLFT94] S. Hanks, T. Li, D. Farinacci, and P. Traina. "Generic Routing Encapsulation (GRE)". RFC 1701, October 1994.

[Hot94] Steven Hotz. *"Routing Information Organization to Support Scalable Interdomain Routing with Heterogeneous Path Requirements"*. PhD thesis, University of Southern California, 1994.

[HSS91] B. R. Hurley, C. J. R. Seidl, and W. F. Sewell. "A Survey of Dynamic Routing Methods for Circuit-Switched Traffic". *IEEE Communications Magazine*, 25(9), September 1991.

[IDR93] International Standards Organization. *"Protocol for the Exchange of Inter-Domain Routing Information among Intermediate Systems to Support Forwarding of ISO 8473 PDUs"*, 1993.

[IT94a] ITU-T. "Draft Recommendation Q.2961". In *Draft Text for Q.2961, Negotiation/Renegotiation: Traffic and QoS Parameters*. Geneva, Switzerland, 1994.

[IT94b] ITU-T. "Draft Recommendation Q.2962". In *Draft Text for Q.2962, Negotiation/Renegotiation: Traffic and QoS Negotiation during Call Establishment.* Geneva, Switzerland, 1994.

[IT94c] ITU-T. "Draft Recommendation Q.2963". In *Draft Text for Q.2963, Negotiation/Renegotiation: Traffic and QoS Negotiation during Active Phase.* Geneva, Switzerland, 1994.

[IT94d] ITU-T. "Draft Recommendation Q.298x". In *Draft Text for Q.298x, Edinburgh TD 152 Rev 1 and 147, Broadband Integrated Services Digital Network (B-ISDN), Digital Subscriber Signalling System No. 2, User Network Interface Layer 3 Specification for Basic Call/Connection Control.* Geneva, Switzerland 13.-21., June 1994.

[Jac88] Van Jacobson. "Congestion Avoidance and Control". In *ACM SIGCOMM*, August 1988.

[KPP92] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. "Multicasting for Multimedia Applications". In *IEEE INFOCOM*, 1992.

[KZ89] Atul Khanna and John Zinky. "The Revised ARPANET Routing Metric". In *ACM SIGCOMM*, September 1989.

[MFF] Steve McCanne, Sally Floyd, and Kevin Fall. "LBNL Network Simulator". http://ee.lbl.gov/ns.

[MG90] Debasis Mitra and Richard J. Gibbens. "State-Dependent Routing on Symmetric Loss Networks with Trunk Reservations: Analysis, Asymptotics, Optimal Design". Technical Report 11212-900703-22, AT&T Bell Laboratories, July 1990.

[MGH91] Debasis Mitra, Richard J. Gibbens, and B. D. Huang. "Analysis and Optimal Design of Aggregated-Least-Busy-Alternative Routing on Symmetric Loss Networks with Trunk Reservation". In *Proceedings of the 13th International Teletraffic Congress*, June 1991.

[Mil84] D. L. Mills. "Exterior Gateway Protocol for Formal Specification". RFC 904, April 1984.

[Min91] Steve Minzer. "A Signalling Protocol for Complex Multimedia Services". *IEEE Journal On Selected Areas of Communications*, 9(9), December 1991.

[MJV96] Steven McCanne, Van Jacobson, and Martin Vetterli. "Receiver-driven Layered Multicast". In *ACM SIGCOMM*, August 1996.

[Moy94a] J. Moy. "Multicast Extensions to OSPF". RFC 1584, March 1994.

[Moy94b] J. Moy. "OSPF Version 2". RFC 1583, March 1994.

[Moy94c] John Moy. "Multicast Routing Extensions for OSPF". *Communications of the ACM*, 37(8), August 1994.

[MRR95] John M. McQuillan, Ira Richer, and Eric C. Rosen. "An Overview of the New Routing Algorithm for the ARPANET". In *ACM SIGCOMM*, January 1995. Originally published in: Proc. Sixth Data Communications Symposium, November, 1979.

[MS91] Debasis Mitra and Judith Seery. "Comparative Evaluations of Randomized and Dynamic Routing Strategies for Circuit-Switched Networks". *IEEE Transactions on Communications*, 39(1), January 1991.

[MS94] Danny J. Mitzel and Scott Shenker. "Asymptotic Resource Consumption in Multicast Reservation Styles". In *ACM SIGCOMM*, August 1994.

[MS95] Ibrahim Matta and A. Udaya Shankar. "Type-of-Service Routing in Datagram Delivery Systems". *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995.

[MT92a] Patrick A. Miller and Petre N. Turcu. "Generic Signaling Protocol: Architecture, Model, and Services". *IEEE Transactions on Communications*, 40(5), May 1992.

[MT92b] Patrick A. Miller and Petre N. Turcu. "Generic Signaling Protocol: Switching, Networking and Interworking". *IEEE Transactions on Communications*, 40(5), May 1992.

[NSC90] Don J. Nelson, Khalid Sayood, and Hao Chang. "An Extended Least-Hop Distributed Routing Algorithm". *IEEE Transactions on Communications*, April 1990.

[PNN96] The ATM Forum: Technical Committee. *"Private Network-Network Interface Specification Version 1.0 (PNNI 1.0)"*, March 1996.

[Pos81a] J. Postel. "Internet Protocol". RFC 791, September 1981.

[Pos81b] J. Postel. "Transmission Control Protocol". RFC 793, September 1981.

[RGW97] A. Orda R. Guerin and D. Williams. "QoS Routing Mechanisms and OSPF Extensions". work in progress, March 1997.

[RL94] Y. Rekhter and T. Li. "A Border Gateway Protocol 4 (BGP-4)". RFC 1654, July 1994.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. " RTP: A Transport Protocol for Real-Time Applications". RFC 1889, January 1996.

[Sha92] Nachum Shacham. "Multipoint Communication By Hierarchically Encoded Data". In *IEEE INFOCOM*, 1992.

[Shu94] Shirdhar B. Shukla. "Multicast Tree Construction in Network Topologies with Asymmetric Link Loads". Technical Report NPS-EC-94-012, Department of Electrical and Computer Engineering, Naval Postgraduate School, September 1994.

[SS95] J. Wroclawski S. Shenker. “Network Element Service Specification Template”. work in progress, November 1995.

[Sti95] Burkhard Stiller. ”A Survey of UNI Signalling Systems and Protocols for ATM Networks”. *ACM SIGCOMM Computer Communication Review*, 25(2), April 1995.

[TEM97] D. Thaler, D. Estrin, and D. Meyer. “Grand Unified Multicast (GUM): Protocol Specification”. work in progress, July 1997.

[TR96] David G. Thaler and Chinya V. Ravishankar. “Distributed Center-Location Algorithms: Proposals and Comparisons”. In *IEEE INFOCOM*, 1996. Also published in *IEEE Journal on Selected Areas in Communications*, April 1997.

[UNI94] The ATM Forum: Technical Committee. *“ATM User-Network Interface (UNI) Signalling Specification Version 3.1”*, September 1994.

[UNI96] The ATM Forum: Technical Committee. *“ATM User-Network Interface (UNI) Signalling Specification Version 4.0”*, July 1996.

[Wax88] Bernard M. Waxman. “Routing of Multipoint Connections”. *IEEE Journal on Selected Areas in Communications*, 6(9), December 1988.

[WC90] Zheng Wang and Jon Crowcroft. “Shortest Path First with Emergency Exits”. In *ACM SIGCOMM*, September 1990.

[WE94] Liming Wei and Deborah Estrin. “The Trade-offs of Multicast Trees and Algorithms”. In *1994 International Conference on Computer Communications Networks*, September 1994.

[WPD88] D. Waitzman, C. Partridge, and S. Deering. “Distance Vector Multicast Routing Protocol”. RFC 1075, November 1988.

[ZCB96] Ellen W. Zegura, Ken Calvert, and S. Bhattacharjee. “How to Model an Internetwork”. In *IEEE INFOCOM*, 1996.

[ZCD] Ellen W. Zegura, Kenneth Calvert, and M. Jeff Donahoo. “A Quantitative Comparison of Graph-based Models for Internet Topology”. In *IEEE/ACM Transactions on Networking.* accepted for publication.

[ZDE$^+$93] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. “RSVP: A New Resource ReSerVation Protocol”. *IEEE Network*, September 1993.

[ZSSC96] Z. Zhang, C. Sanchez, B. Salkewicz, and E. Crawley. “Quality of Service Extensions to OSPF”. work in progress, June 1996.