Alternate Path Routing for Multicast

Daniel Zappala
Department of Computer and Information Science
University of Oregon, Eugene
zappala@cs.uoregon.edu

Abstract— **Alternate path routing has been well-explored in telecommunication networks as a means of decreasing the call blocking rate and increasing network utility. However, aside from some work applying these concepts to unicast flows, alternate path routing has received little attention in the Internet community. We describe and evaluate an architecture for alternate path routing for multicast flows. For path installation, we design a receiver-oriented alternate path protocol and prove that it reconfigures multicast trees without introducing loops. For path computation, we propose a scalable local search heuristic that allows receivers to find alternate paths using only partial network information. We use a simulation study to demonstrate the ability of local search to find alternate paths approximately as well as a link-state protocol, with much lower overhead.**

I. INTRODUCTION

The primary purpose of current Internet routing protocols is to maintain connectivity. Both unicast and multicast routing protocols identify a single, shortest-path route (or tree) from a source to a destination. This design evolved from earlier load-based ARPANET routing protocols [1], [2] as a means for providing stability in a rapidly-expanding and dynamic internetwork.

Recently, attention has shifted toward the development of routing protocols that find paths based on meeting application requirements and increasing network utilization. Research in this area has concentrated on designing and evaluating Quality of Service (QoS) routing protocols for unicast [3], [4] and multicast [5], [6]. In this work, an application is assumed to specify a QoS request to the network as some combination of delay, bandwidth, and loss characteristics. The QoS routing protocol then computes a path that has available resources and installs this path for the application. Many QoS routing protocols are based on a link-state framework that globally distributes topology, link load measurements, per-flow resource usage, and multicast group membership. Due to this high overhead, the link-state approach does not scale well to large networks or large numbers of highly-dynamic groups. Newer research has focused on distributed protocols that search a portion of the network for a feasible path, subject to delay and bandwidth constraints [7], [8], [9]. While this work is promising, it relies on message passing and a decision criteria that requires universal deployment in order to be effective. Our emphasis in this work is to serve all applications, not just those that need a QoS guarantee, and to use a more general approach, in which path collection, the selection criteria, and path installation are split into separate components.

We are investigating the use of alternate path routing to more generally and more scalably satisfy these same goals of meeting application requirements and increasing network utilization. With alternate path routing, the network primarily uses shortest-path routing, and switches some load to longer paths only when the shortest paths are overloaded. This model works well when the shortest paths are provisioned so that they handle most of the load in the network. While alternate path routing cannot take into account as wide a range of application requirements as QoS routing, it may be useful for adaptive real-time applications or those using priority-based differentiated services.

Alternate path routing has a long and successful history within the telephone network [10], [11], [12], [13], [14], [15], where the shortest path between exchanges (typically one hop across the backbone) is used until its capacity is reached, and then alternate paths (usually two hops) are tried. This work cannot be directly utilized for Internet routing, as the Internet has a much more general topology and heterogenous application requirements. Nevertheless, a recent study by Savage et al. [16] indicates that alternate paths are available in the Internet, frequently with superior quality to the default path provided by current routing protocols. Developing an alternate path routing architecture for the Internet would allow applications to take advantage of these paths. Toward this end, some work has been done applying the alternate path approach to unicast Internet routing [17], [18]. However, alternate path routing for multicast remains an open area of research.

In this paper we describe and evaluate two important components of alternate path multicast routing: (1) path computation and (2) path installation. Our primary concern in designing these components is the ability to scale to large networks and situations when group membership is highly dynamic. The alternate path approach naturally meets these goals because shortest-path routing is used for most paths. In addition, we distribute the work of path computation and installation to the group members. Furthermore, each group member uses only partial information about the network's topology to compute alternate paths. This information does not need to be consistent nor synchronized across group members. Likewise, our path installation component allows a group member to reconfigure the multicast independently from other receivers. Finally, both components operate independently of the unicast and multicast routing protocols, so they are not limited to networks that use link-state routing.

This paper represents the first step in assessing the feasibility of our alternate path approach. First, we develop an alternate path protocol that allows receivers to install alternate paths into multicast trees without introducing loops. It is not a trivial matter to define a loop-free multicast routing protocol that allows

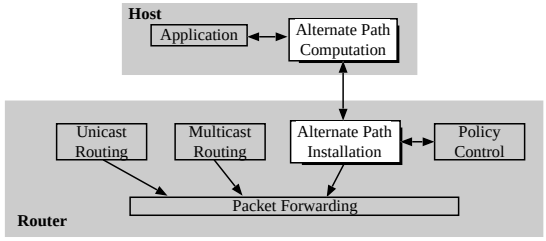


Fig. 1. Routing Architecture

receivers to use arbitrary routes[1]; we identify a general looping problem and prove that our protocol is loop-free. Second, we evaluate path computation heuristics from the perspective of meeting application requirements. Thus, our evaluation is in terms of success rate, latency, overhead, and path length. We reserve an assessment from the standpoint of network utilization for future work.

The rest of the paper is organized as follows. Section II describes our routing architecture in more detail. In Section III we illustrate our alternate path protocol and in Section IV we present our simulation study of path computation heuristics. Section V examines our plans for future work based on this approach.

II. ROUTING ARCHITECTURE

Figure 1 diagrams the components of our routing architecture. An application starts by joining a multicast group, and its first-hop router joins the *default multicast tree*. The default tree can be a shortest-path tree [19], [20] or a shared tree with a single core [21], [22], [23]. If the application determines that service on the default tree is unacceptable, it may request an alternate path by contacting the path computation component. If this component is able to find an alternate path, it expresses it as a source route and delivers it to the path installation protocol, which reconfigures the multicast tree with this route.

The path installation component operates independently of the multicast routing protocol (for details, see Section III). This independence is an important feature because it makes it simple to apply trunk reservation in our architecture; trunk reservation improves network utilization by limiting the use of alternate paths during high load [24].

Another key feature of our routing architecture is a policy control element that screens source routes and prevents malicious, malfunctioning, or misguided users from undermining the integrity or efficiency of a multicast tree. For example, the policy control element should be able to prevent denial-of-service attacks, in which a group member installs a source route that is intended to degrade the service given to another group member. While we speculate that limiting the hop-count of a source route can greatly limit these dangers, policy control design is beyond the scope of this work.

Note that in previous work [25], we have placed path computation under the control of first-hop routers due to policy concerns. However, it is likely that policy control will be needed at other routers along a given source route anyway, particularly if the route crosses administrative domains. Placing path computation within hosts makes it possible to deploy new techniques for finding paths without requiring simultaneous upgrading of the routing infrastructure. Having hosts compute paths likewise makes it easier to build a caching infrastructure for sharing the path finding effort.

Viewed another way, deploying a protocol to install alternate paths is a critical step as it then allows the community to work on host architecture, policy control, and path computation. While the first two of these areas are not within the scope of this paper, we must make several assumptions about the host architecture for our later discussion of path computation:

- **Service Monitoring:** First, applications need some mechanism for determining that the service they are currently receiving is inadequate. A simple example is a user of a real-time audio or video application (or the application itself) monitoring the loss rate and triggering an alternate path request if it exceeds a threshold for a period of time.
- **Link Identification:** Second, applications need to identify which links are causing the service degradation. Continuing the example above, the user or application may use *mtrace* [2] or a similar tool to determine that a bottleneck link may be congested.
- **Application Interface:** Finally, the application must give the list of problem links to the path construction component and ask it to find an alternate path. The interface should allow for an asynchronous response indicating that installation is complete, indicating the application can restart its monitoring to determine whether the new path has improved its performance.

Note that service monitoring and link identification may be simplified if the network supports Quality of Service using a resource reservation protocol (such as RSVP [26]). Service monitoring can compare the reservation made to the service received, and a reservation denial can indicate which link should be avoided. Section V discusses extensions to this work when resource reservation is available; the rest of this paper assumes it is not.

III. PATH INSTALLATION

We have developed an alternate path multicast (APM) protocol that allows a receiver to reconfigure a multicast tree with an arbitrary source route. This source route can be any loop-free sequence of nodes in the network, subject of course to policy restrictions as discussed in Section II. APM takes the source route and installs it into an *alternate path tree*, consisting of the collection of all the source routes used by any of the group members. The alternate path tree is a separate tree that takes precedence over the default tree.

Receivers using alternate paths coexist safely with receivers who are using the default multicast tree. Receivers on the de-

[1] By arbitrary we mean traversing any loop-free sequence of nodes in the network, subject of course to policy restrictions.

[2] *mtrace* is a tool that queries routers in a multicast tree for statistics such as packet rates and losses. Current multicast routers implement this functionality so that users may discover problems in a multicast tree they are using.

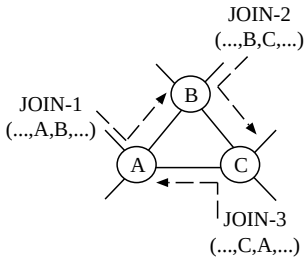
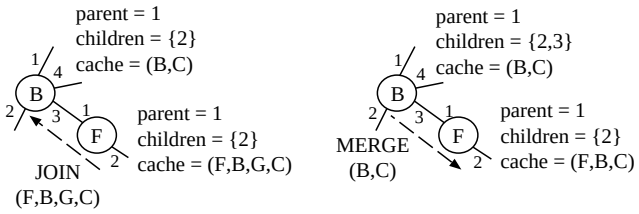


Fig. 2. Alternate path loop



(a) JOIN from F to B (b) MERGE from B to F

Fig. 3. APM JOIN and MERGE messages

fault tree use a branch that extends upward to the point where it meets the alternate path tree, then use the alternate path tree from that point forward. To prevent duplicate packets from being sent to these receivers, routers on the alternate path tree prune conflicting branches from the default tree.

A. Looping Dangers

Before describing the APM protocol in detail, we first illustrate how a naive path installation protocol can form loops if it allows receivers to join a multicast tree using an arbitrary source route. Note that at no time can these loops lead to looping of data.

We consider a scalable, receiver-oriented multicast routing protocol such as PIM [22]. A minor extension would allow a JOIN message to carry a source route. Figure 2 shows three such JOIN messages traveling upstream simultaneously from three separate receivers. Each JOIN message contains a source route; as a router processes the JOIN it creates multicast tree state for the group in question (a parent interface and a child interface), then forwards the message to the next hop in the source route. If a JOIN message reaches another source-routed branch of the multicast tree, it merges with the tree; that is, the router adds a child interface and does not forward the message any further.

Based on the order in which routers process the messages, it is possible that JOIN-1 merges with JOIN-2, JOIN-2 merges with JOIN-3, and JOIN-3 merges with JOIN-1. Hence, a loop forms, and none of the three receivers actually join the multicast tree.

Based on our reading of the QoSMIC protocol [27], we believe it may be vulnerable to join loops. At the very least, the reference above does not include enough details of the joining mechanism to indicate how it solves this problem. A related looping problem was discovered and solved during our work with the design of RSVP [28], so we would not be surprised to find it in other receiver-oriented multicast protocols.

B. The APM Protocol

Having illustrated the danger of forming loops with arbitrary source routes, we now describe the APM protocol and how it avoids loops.

B.1 Joining with an Alternate Path

When a receiver wishes to join a multicast tree with an alternate path, the APM protocol sends a JOIN message upstream containing the source route the receiver wants to use. The source route must be a strict source route, listing all nodes in order from the receiver to the root [3]. At each router listed in the source route, APM creates state for the *alternate path tree*, which is a separate multicast tree from the default tree. The state APM creates consists of the parent interface, the set of child interfaces, and a cache of the route sent upstream from that router. This process is illustrated in Figure 3a.

The goal of the APM protocol is to maintain consistent caches at each router in the alternate path tree. We define consistency to mean that the cache at node X on the alternate path tree must exactly match the route from X to the root of the alternate path tree. Likewise, if node X has cached the route (X, Y, Z), then node Y must have a cache containing (Y, Z).

To maintain this notion of consistency, APM uses a MERGE message to update caches as follows. When the JOIN message reaches the root of the tree or some other router on the alternate path tree, it stops and triggers a MERGE message. The MERGE message contains the cached route from the parent; the child who receives it updates its cache so that it is consistent with this route. The child repeats this process by sending a MERGE message to each of its children, so that eventually all caches are updated in the subtree. Figure 3b gives an example of how a MERGE message operates.

Our design of the MERGE message uses a first-come, first-served approach to re-routing the default tree with an alternate path. Malicious source routes are screened by policy control, but otherwise the first receiver to reconfigure a branch of the multicast tree to its liking wins over later receivers. Lacking any metrics that allow the network to quantify some paths as better than others, this is the best we can do. In [25] we use a simple extension to APM to allow re-routing of the alternate path tree when QoS metrics are available.

Using MERGE messages to maintain consistent caches allows APM to safely detect and remove any loops that form during the joining process. Section III-C proves this assertion, but for now we illustrate how loops are broken. Following the example of Figure 2, router A will send a MERGE message with

[3] Requiring the source route to be strict does not prevent a receiver from using a loose route; a receiver can easily convert a loose route to a strict route by recording the path traversed by the loose route.

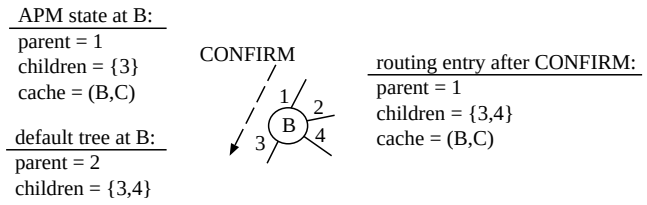


Fig. 4. APM CONFIRM message

the route $(..., A, B, ...)$ to router C, which will in turn send a MERGE message with the route $(..., C, A, B, ...)$ to router B. When router B sees that its updated cache entry would contain itself, it detects that a loop has formed and removes the state it has created with a FAILURE message (see Section III-B.3).

We have chosen to detect and break loops in this manner with APM because the join latency is small in the common case when there is no loop formed. In this case, the latency is only the time it takes to traverse a source route from the receiver to the multicast tree. In prior work [29], [25] we designed a setup protocol with the opposite tradeoff, namely loop prevention at the expense of larger latency in the common case. Since formation of a loop requires several receivers trying to use an alternate path at the same time, with routes that overlap to form a loop, and with processing of the JOIN messages interleaved at precisely the right times, we now believe they will be rare enough that the extra overhead is not worth the benefits of loop prevention.

B.2 Confirming the Join

While the MERGE message ensures that JOIN messages do not loop, APM also needs a mechanism to signal that the entire alternate path has been installed. Toward this end, each router on the alternate path tree also keeps one bit of state, called the *confirm bit* indicating whether the router has successfully joined the alternate path tree. When a router first processes a JOIN message, this bit is cleared. Once the JOIN reaches the root of the tree or reaches another router with the *confirm bit* set, then a CONFIRM message is sent downstream to all children in the tree. Each router processing the CONFIRM message sets the *confirm bit*. For simplicity, the CONFIRM message may be combined with the MERGE message when both need to be sent downstream at the same time.

When a router processes the CONFIRM message, it also locally overrides its routing entry for the default tree with the routing entry for the alternate path tree. It does this by using the parent interface specified in the routing entry for the alternate path tree and then taking the union of the child interface sets for both the alternate path tree and the default tree. Figure 4 illustrates this process. By operating at the level of routing entries, APM remains independent of the particular multicast routing protocol being used to build the default tree. Thus, APM should be compatible with DVMRP [19], MOSPF [20], and PIM, the most commonly-used multicast routing protocols.

At this time, the router also checks if the incoming interface for the alternate path tree is different from the incoming interface specified by the default tree. If it is, then it signals the multicast routing protocol to prune the conflicting branch of the default tree.

Note that using the CONFIRM message for pruning the default tree, rather than the MERGE message, ensures that routers do not reconfigure the default tree until the alternate path tree is ready. Otherwise, data would be lost for a short time when multiple merges occur and data would be lost for a long time when a temporary JOIN loop forms.

When the CONFIRM message reaches a receiver, it knows its alternate path has been installed and can signal the host that its request is complete. Note that a MERGE message may arrive at a receiver prior to a CONFIRM message, but this indicates only that the JOIN merged with another JOIN, and has not yet reached the confirmed portion of the alternate path tree.

B.3 Link and Policy Failures

To handle link failures, each router in the alternate path tree uses a Hello protocol to be sure its parent is still operational. The child router periodically sends a HELLO message to its parent and waits for an acknowledgment from the parent. If neither side hears from the other for some period of time, the link is considered down. The HELLO message can be aggregated across all trees, so that a router runs only a single instantiation of the protocol for each of its neighbors.

If a router detects a link failure, it tears down all alternate path branches using the link, allowing those routers to revert to the default tree. A router does this by sending a FAILURE message to the child (or parent) of the affected branch. The router then waits for an acknowledgment from the child (or parent) before deleting its state. Subsequent routers then repeat this process until the affected branch is removed. FAILURE messages can also be aggregated by sending one to each neighbor and listing the affected groups.

Note that policy failures at joining time also result in teardown via a FAILURE message.

B.4 State Maintenance

Many recent protocols, such as PIM [22] and RSVP [26], use soft state. With soft state, a router expects its neighbor to refresh state periodically, otherwise the state times out. With multicast trees, this is typically done on a per-tree basis.

APM uses hard state, in which routers expect to receive an acknowledgment for each message sent and resend the message if no acknowledgment is received after some time. APM uses hard state rather than soft state for a number of reasons. First, APM does not adapt alternate paths to link and node failures because it leaves the computation and assessment of alternate paths to the receiver. This makes it more efficient to use an aggregated HELLO message to detect link failures (and trigger teardown), rather than a soft-state refresh for each alternate path tree. Moreover, using hard-state allows APM to quickly detect and break JOIN loops; a hard-state acknowledgment timer can be set to a much shorter period than a soft-state refresh timer since it is used only at setup time.

C. Loop-Freedom

Having described the APM protocol, we now prove that it continually maintains a loop-free multicast tree, even while the tree is reconfigured with alternate paths. Here we are considering the multicast tree used to send data from the root to the group members; this tree is the combination of confirmed branches of the alternate path tree and any remaining default tree branches that have not been pruned.

First, it is clear that data transmitted by a source can never loop, since only one parent interface is allowed at any router. We must then prove that each receiver has a loop-free path to the root of the tree. We start by considering just the alternate path tree, and we assume that no link or node failures have occurred:

Theorem 1: The APM protocol maintains a loop-free, confirmed, alternate path tree, such that any receiver on the tree has a loop-free path to the root.

Proof: Consider a router A_i that is on a confirmed branch of the alternate path tree. By definition, this router has its confirm bit set, a cached route of $A_i, A_{i+1}, ...A_n$, and its parent set to A_{i+1}. Having this state implies that A_i must have received a CONFIRM message with route $A_{i+1}, A_{i+2}, ...A_n$ from A_{i+1}. This in turn implies that router A_{i+1} must also have its confirm bit set, a cached route of $A_{i+1}, A_{i+2}, ...A_n$, its parent set to A_{i+2} and a set of children including A_i. Now we can start with any receiver A_1 that is on the confirmed alternate path tree and has a loop-free, cached route $A_1, A_2, ...A_n$. Then by induction every router A_i has a cache that is consistent with this path, and APM has installed a loop-free path from A_1 to the root of the tree. Thus, all the confirmed branches maintained by APM form a tree. ■

Link and node failures cause any affected branches to be removed from the alternate path tree, clearing any confirm bits. By reasoning similar to the above proof, we can show that the FAILURE message removes these branches without introducing any loops. After the branches are removed, affected receivers must join again, and this case is covered by Theorem 1.

We can now consider the loop-freedom of the reconfigured multicast tree.

Theorem 2: A loop-free default multicast tree that is reconfigured by the APM protocol remains a loop-free multicast tree; that is, each receiver has a loop-free path from itself to the root of the tree.

Proof: A receiver on the reconfigured tree is either on a confirmed branch of the alternate path tree or a branch of the default tree. By Theorem 1, a receiver on the alternate path tree has a loop-free path to the root. A receiver that is not on the alternate path tree is on a branch of the default tree. By definition, the multicast routing protocol that created the default tree has created a loop-free path $A_1, A_2, ...A_n$ from the receiver to the root of the tree. If none of the routers in the path $A_1...A_n$ is on the alternate path tree, then we know this path is loop-free. If some router A_j is on the alternate path tree, then the receiver's path is split into two segments, $A_1...A_{j-1}$ and $A_j, B_1, ...B_m, A_n$ The segment $A_1...A_{j-1}$ is loop-free because it is a subset of the loop-free path $A_1...A_n$. For the segment $A_j, B_1, ...B_m, A_n$, we may consider A_j to be a receiver on the alternate path tree. By Theorem 1, we know this segment is a loop-free path to the root of the tree. Because each segment is loop-free, we know the entire path $A_1...A_n$ is loop-free. ■

IV. PATH COMPUTATION

Having defined an alternate path multicast protocol, we now turn to the issue of path computation. We assume that each receiver computes its own alternate paths, using the APM protocol to resolve any conflicts. As discussed in Section II, we further assume that each receiver is capable of service monitoring and link identification, such that a receiver is able to provide the path computation component with a list of congested links it would like to avoid. Finally, receivers start without any knowledge of the topology or any other group members.

The job of the path computation component, then, is to accumulate a *partial map* of the network and use this map to find paths that avoid congested links. It is important to note that in our alternate path routing architecture, path computation is purely a local decision. This means that as path computation techniques are improved, a new protocol can be deployed incrementally.

In this paper, we focus on one such protocol, a local search heuristic, that we believe is feasible today and requires relatively few changes to routers. We concentrate on using this heuristic to demonstrate the feasibility of our approach; discussion of several related heuristics can be found in [25].

We begin this section by describing our local search heuristic, then describe our simulation model and our results. For the majority of our simulations we use a single receiver, rather than a larger multicast group, because this represents the worst-case scenario. Multicast groups with more than one member can only improve the efficiency of path computation, as receivers will benefit from path computation and installation done by their neighbors. In a later part of this section we return to this issue and quantify these benefits for larger multicast groups.

A. Local Search Heuristic

Because path computation is distributed, each receiver only needs to find an alternate path to the root of the tree, without taking into account any other receivers. This means that path computation can build on the unicast routing protocol as a source of alternate paths.

With the local search heuristic, a receiver builds a partial map using the unicast paths of nearby routers. The algorithm run by the path computation component (PCC) consists of the following steps:

1. At startup, the PCC initializes the partial map with the paths from itself to all neighbors within n hops.
2. As each congested link is identified by an application, the PCC adds the link to its map and marks it.
3. When an alternate path is needed for some root, the PCC contacts all neighbors within n hops and collects the unicast path that each of them uses to reach the root. The PCC adds

TABLE I
LATENCY AND OVERHEAD BOUNDS FOR LOCAL SEARCH

Type of Search	Latency	Overhead
Consecutive	$c(c-1)^{n-1}t$	$c(c-1)^{n-1}$
Parallel	t	$c(c-1)^{n-1}$
Expanding-Ring	nt	$c(c-1)^{n-1}$

TABLE II
NETWORKS AND THEIR CHARACTERISTICS

Network	Nodes	Type	Degree	Diameter
F100	100	flat	4.26	8
T100	100	transit-stub	3.74	11
T1000	1000	transit-stub	4.35	13

these paths to its partial map and runs a Dijkstra computation on the map to find a path to the root that avoids links marked as congested.

4. If an alternate path is not found, the PCC asks the root of the tree to find an alternate path. The root of the tree runs the local search heuristic from its end and returns the results.

We have previously found that this last step is essential for hierarchical networks, since only the root is able to find paths that utilize the extra links that are nearer to it [25]. When the multicast group is large, however, the root should avoid doing a separate path computation for every group member. Likewise, receivers should avoid flooding the root with a large number of alternate path queries. To address these concerns, the root should only do one path computation during a period of m minutes. During this time, the root can suppress excess queries by multicasting a command to receivers that tells them to back off.

The local search heuristic can be tuned by each receiver to favor either lower latency or lower overhead. To optimize for latency, each of the neighbors within n hops can be queried in parallel. To decrease overhead, the neighbors can be queried consecutively, starting with those that are nearest. This will decrease overhead if an alternate path is found before all neighbors have been queried. A compromise is to use an expanding-ring search [30] to first query the neighbors within 1 hop, then those within 2 hops, etc.

Table I lists the latency and overhead bounds (in terms of nodes queried) for each of the above cases, where c is the maximum degree of connectivity for the network, t is the time to query a node for a unicast path, and n is the number of hops in the search. The latency bound for consecutive search is derived by noticing that a receiver will contact at most c neighbors, each of whom will contact at most $c-1$ neighbors because they do not backtrack to the receiver. Starting at each of the original c neighbors, $n-1$ more rounds of search will be conducted. The overhead bound is the same for all the cases of local search. Whereas the overhead for parallel search will always be at the maximum, the others will have a lower average-case overhead. We use the expanding-ring search option in our later simulations to illustrate this case.

The local search heuristic can be readily implemented by a host. A list of neighbors within n hops can be composed by using recursive SNMP [31] queries. Unicast paths can be collected by simply querying a node for its path to the sender. If the node is using a path-vector protocol or a distance-vector protocol with source tracing [32], [33], the node already has the path stored in its routing tables and can immediately respond to the query. If a more simple distance-vector protocol such as RIP [34] is being used, the node can instead use the *traceroute* program. The *traceroute* program uses a clever trick to elicit a response from each router along a path, but requires as many probes as there are hops in the path. Of course, a more efficient *traceroute* program could be developed that records the route at each hop, as has been done with the multicast traceroute program *mtrace*.

B. Simulation Model

We evaluated the expanding-ring local search heuristic within version 1 of the LBNL network simulator *ns* [35]. Because its performance depends heavily on the topology, we used two different network models: a flat random network and a transit-stub network. Flat networks are created using the Doar-Leslie edge-connection method [36] so that edges mostly connect nodes "near" each other. Transit-stub networks model a two-level hierarchy, with each stub network consisting of a flat network.

It is important to note that the networks we chose represent extremes and are thus good test cases for examining the performance of local search. In particular, we use a transit-stub network where each stub has a link to only one transit. This strict two-level hierarchy limits the effectiveness of local search because a receiver's neighbors are all within one stub network, which has a single link to the backbone. Thus, neighbors will not be able to supply paths that avoid bottlenecks in the backbone or in other stubs. Our simulations have shown that modeling more "realistic" networks with more connectivity between stubs and transits (and to other stubs) improves the performance of local search. Nevertheless, we present only these two models here because of their usefulness in illustrating the extremes of local search behavior.

Table II lists the topologies we used and their characteristics. The bulk of our simulations use two 100-node networks, one flat and one hierarchical. To test the scalability of local search, we used three 1000-node transit-stub networks, each using a different method of growth (increasing the size of the transit network, increasing number of transit networks, or increasing the size of the stub networks). To conserve space, we discuss only representative simulations from one of these networks as indicated in the table. All of these networks were generated with the Georgia Tech ITM software [37], [38].

Note that these topologies model only the router portion of the network. For simplicity, we do not model hosts, as they will generally connect to a single router and thus add only a single link to the number of links that need to be visited. Thus our evaluation of heuristics is independent of whether the receiver

or a router does the path computation.

C. One Receiver

We first evaluate path computation using one receiver and multiple congested links. We use only one receiver because this represents the worst-case scenario of the smallest multicast group. Larger multicast groups can only improve the performance of local search as a receiver may benefit from the path computation and installation of other receivers. The following section looks at larger group sizes.

For these simulations, we fix the percentage of links that will be marked as congested and randomly select those links. We then choose a sender and a single receiver, determine whether the receiver's shortest path contains any congested links, and if congestion is found attempt to compute an alternate path. We do this for 100 different senders and receivers, and run 100 of these simulations for each congestion percentage.

In each simulation, we compare the performance of local search to a link-state algorithm with full knowledge of the topology. We do not eliminate cases where the link-state algorithm can't find a path. We start with results for the 100-node flat network, then consider hierarchical networks.

C.1 Flat Network

In the 100-node flat network, it is fairly easy for the local search heuristic to find alternate paths because the graph is richly connected. In this section we examine the performance of the heuristic with respect to its ability to find paths, the length of these paths, and the overhead incurred by the search.

To examine the effectiveness of alternate path routing, we plot the percentage of non-congested receivers as a function of the number of congested links. We include receivers using either a shortest path or an alternate path as this gives us a more accurate picture of the results. Plotting only the alternate path success rate is misleading at low load when most receivers can get by with a shortest path. Likewise, during heavy load there are many receivers for whom no alternate path exists; a high success rate in this condition does not mean much when the vast majority of receivers can't be helped anyway. For comparison we also plot the result of using no alternate path routing.

Figure 5 shows the effectiveness of alternate path routing using local search compared to a link-state algorithm with full knowledge of the topology. This graph shows the improvement in the heuristic as the number of hops in in the expanding-ring search is increased. Using 2-Hop local search, a receiver is able to find alternate paths with success nearly equal to that of the link-state algorithm under light load (less than 15% congested links).

For comparison, we also include a link-state algorithm that limits the length of an alternate path to no more than 3 hops beyond the shortest path (designated as Link-State-3). The 3-Hop search does better than this at light loads, and nearly the same under heavy loads. This leads us to guess that under heavy loads the link-state algorithm finds very long alternate paths, whereas the local search heuristic does not. Figure 6 confirms

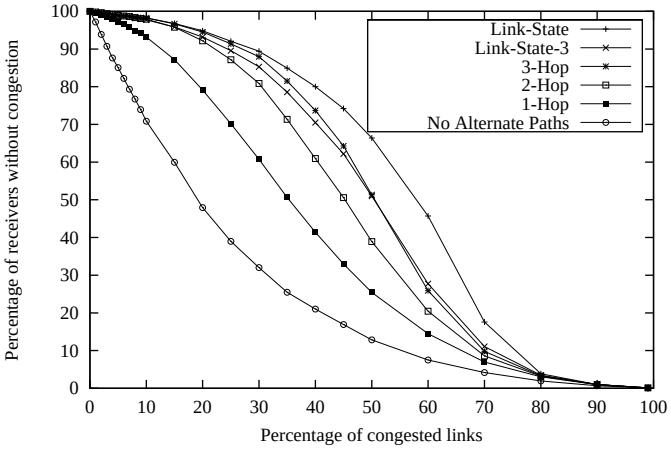


Fig. 5. Alternate path routing on F100

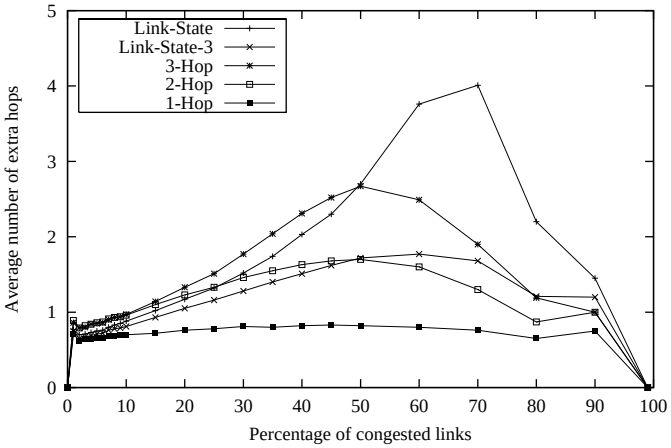


Fig. 6. Alternate path length on F100

this hypothesis. This figure shows the average path length in terms of the number of extra hops beyond the shortest path. The 3-Hop search has a path length near the link-state algorithm until 50% of the links are congested, after which path lengths are reduced as congestion increases further. This is exactly the situation we want for local search. Under high loads, long alternate paths should not be used since they decrease network utilization by taking away resources from the shortest path of other receivers [24].

Figure 7 illustrates how 3-Hop local search finds alternate paths for these simulations. Under light loads, the heuristic generally finds paths within 1 or 2 hops, with 3 hops becoming increasingly more frequent at higher loads. The heuristic resorts to asking the sender less than 20% of the time, mostly under moderate loads. By comparing this graph to those for other heuristics (not shown), we find that expanding the local search to more hops decreases the likelihood that the sender is asked for a path. This may be an acceptable tradeoff for a single receiver, but for a large multicast group it will likely be better to ask the sender. This causes the sender to do one search near

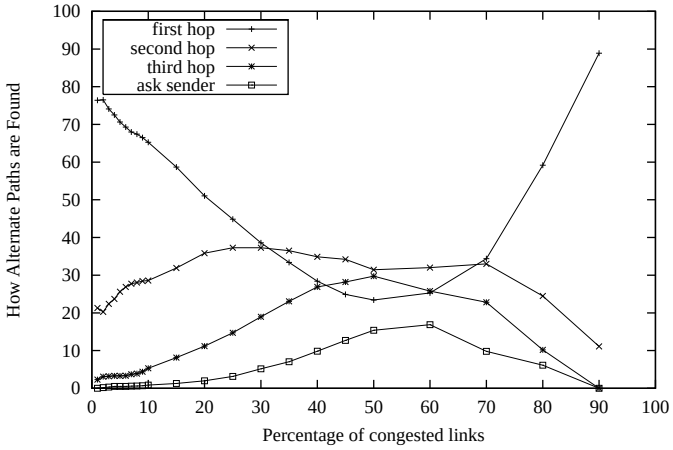


Fig. 7. How alternate paths are found: 3-Hop, F100

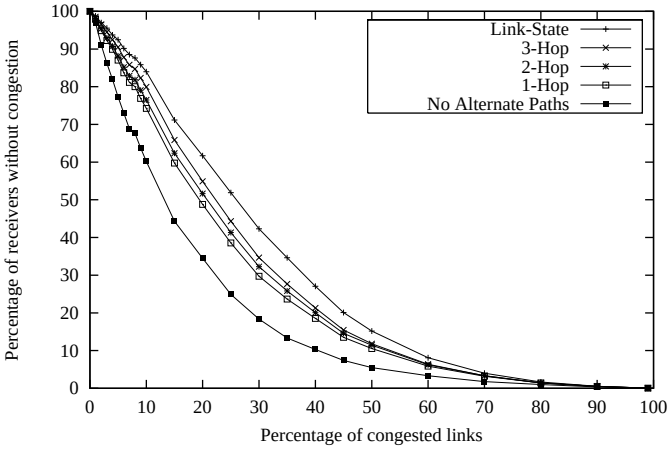


Fig. 9. Alternate path routing on T100

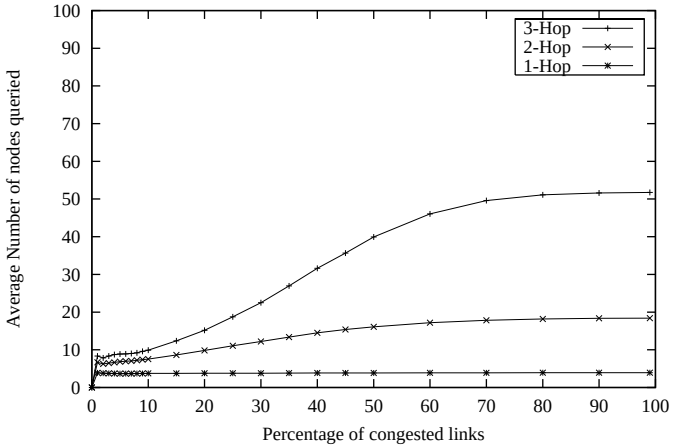


Fig. 8. Number of nodes queried on F100

itself (multiple requests are suppressed as discussed above), instead of having each receiver doing a large search.

Figure 8 shows the average number of nodes queried by each of the heuristics. When load is low, the number of queries stays below 10. Under higher loads, the overhead levels off to a value well below the bound for the heuristic (810 for 3-Hop). This is a simple illustration of how the overhead of the heuristic can be limited by limiting the depth of the search. It is important to note, however, that our simulation overestimates this overhead because we do not re-use the topology information collected by local search for later alternate path attempts. Section V discusses future work we will perform to assess this overhead in more dynamic situations. Nevertheless, it is important to limit overhead as congestion increases, as there can potentially be a large number of receivers exploring all n hops without any success. Thus, to further limit overhead, a node may refuse to respond to a local search query during high load.

C.2 Transit-Stub Network

In the 100-node transit-stub network, performance of the local search heuristic is decreased because most receivers are located in a stub that has only one link to its transit. However, overall there are many fewer alternate paths, so the heuristic still manages to perform well compared to a link-state algorithm. In addition, the overhead of local search is greatly reduced compared to the flat network.

Figure 9 shows the effectiveness of alternate path routing using local search compared to the link-state algorithm on the 100-node transit-stub network. In this network there is not much to gain by increasing the depth of the expanding-ring search. However, performance is at the most only about 10% worse than with the link-state algorithm. Path lengths for local search and for the link-state algorithm average about 1 extra hop.

Figure 10 illustrates how 3-Hop local search finds alternate paths for the transit-stub network. Due to the symmetry of the sender and receiver stubs, a path is equally likely to be found locally or by asking the sender during low loads. This shows how important it is to ask the sender for a path in a hierarchical network. During heavy loads, most paths are found after 1 hop of local searching. Most likely, this is an artifact of the simulation: those receivers that are in the same stub are the only ones that stand a chance of finding an alternate path under heavy load. In this case, the path will be found with a local search and asking the sender won't be necessary.

Due to the transit-stub hierarchy, the overhead of searching is greatly reduced compared to the flat network. Even the 3-Hop search averages at most about 15 queries.

C.3 1000-Node Transit-Stub Networks

To test the scalability of the local search heuristic, we ran the same simulations on the three 1000-node transit-stub networks described above. For each of these networks, the relative performance of the heuristic to the link-state algorithm stayed about the same. On any of the networks, local search with 1-

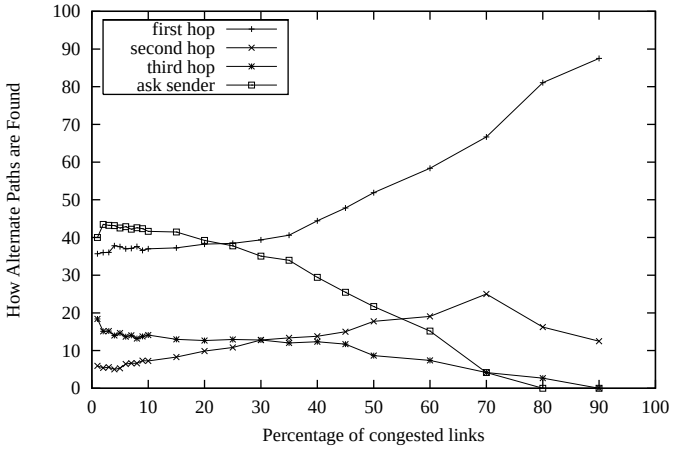


Fig. 10. How alternate paths are found: 3-Hop, T100

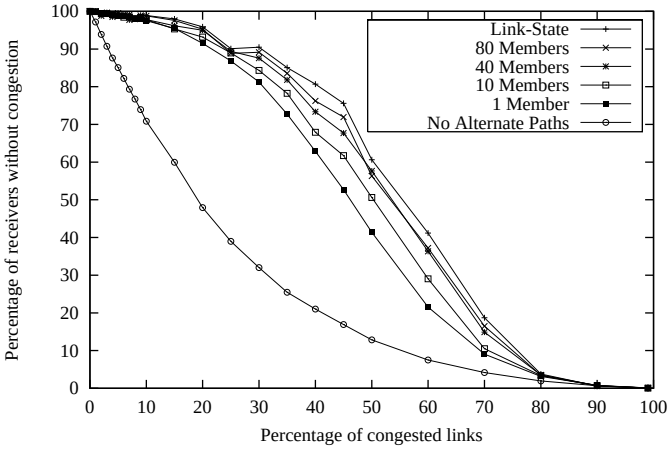


Fig. 12. Alternate path routing using 2-Hop versus group size on F100

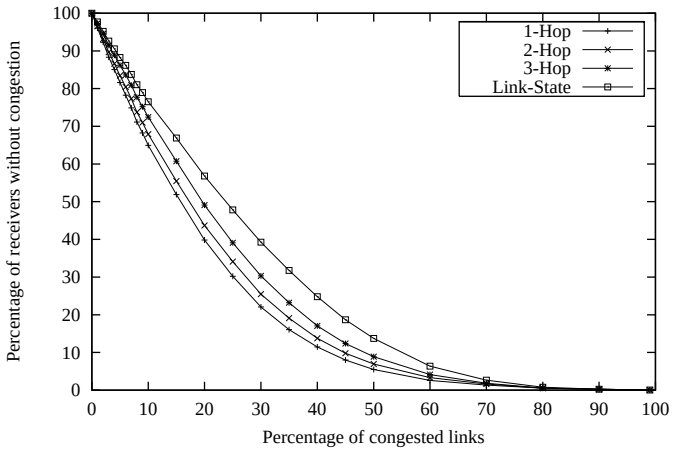


Fig. 11. Alternate path routing on T1000

3 hops was at most about 10%-20% worse than the link-state algorithm. Figure 11 shows a representative graph for network T1000.

Most importantly, this performance was achieved without increasing the overhead of the search; given a number of hops in the local search the overhead is constant, regardless of the size of the network. In fact, the percentage of links discovered by the heuristic ranges from 0.5% to 2.5%. This means that a receiver needs only a small percentage of the network map to compute alternate paths.

D. Multiple Receivers

Once a multicast group grows to include a large number of members, then receivers naturally benefit from the path computation and installation done by their neighbors. Thus, the larger a group, the easier it becomes to approximate link-state path computation with only a small local search.

Figure 12 shows a representative sample of our simulations: the effectiveness of alternate path routing using 2-Hop local search with varying numbers of group members on the flat network. This graph shows two important results. First, a single receiver gets the bulk of the performance gain over doing no alternate path routing. This means that alternate path routing for multicast can be effectively distributed to receivers and that path computation can be built on top of unicast routing. Second, for large groups, link-state path computation can be approximated by having each receiver obtain a small part of the network's map and run a local path computation. In this example, a 2-Hop local search is all that each receiver needs to run. We suspect this effect will be even more pronounced on "realistic" transit-stub networks.

V. CONCLUSIONS AND FUTURE WORK

This paper has described an architecture and mechanisms for computing and installing alternate paths for multicast. Our focus is on techniques that are scalable to large networks. Toward this end, we distribute path computation and installation to receivers. The receivers use a local search heuristic to explore a small portion of the network's topology and compute an alternate path. They then use an installation protocol that reconfigures a multicast tree with this new path. Our simulations demonstrate that the local search heuristic can find alternate paths nearly as well as a full link-state routing protocol, using only partial information about the network. This heuristic scales well to large networks because it inherently limits the amount of the network it will explore.

As discussed earlier, additional components are required to complete this architecture, namely service monitoring, link identification, and an application interface. In addition, we envision a number of areas for future work:

A. Dynamic Evaluation

Because our simulations are static, they do not capture the benefits of sharing path computation results over time. A dynamic evaluation will require a more complex model of network load and group membership, aging of topology information, and a mechanism for sharing topology information among

nearby receivers (i.e. designating one path computation component per site). This evaluation should show how the path computation overhead decreases with the frequency and distribution of alternate path requests. In addition, the APM protocol can be simulated to determine the average setup delay (i.e. how often loops are detected and broken).

B. Network utilization

Having demonstrated the utility of alternate path routing from an application standpoint, it is also important to evaluate our architecture from the standpoint of network utilization. To guard against excessive alternate path routing, we can incorporate a form of trunk reservation into our architecture. Other important network-centric metrics include multicast tree cost, delay, and traffic concentration.

C. QoS routing

Finally, our alternate path architecture can easily be extended to a scalable QoS routing architecture. The local search heuristic can be modified to collect the current resource utilization on the paths it explores. Any QoS route computation algorithm can then be applied to the partial map to find a QoS-capable route. As the path is installed, the APM protocol can call RSVP locally at each router to create a reservation. In addition, a simple extension to APM can be employed to allow re-routing of the tree when its QoS is not sufficient [25]. While these modifications are straightforward, it is not clear how well this architecture will perform compared to existing QoS routing protocols. We intend to compare its performance to both the traditional link-state algorithms and the newer distributed approaches.

ACKNOWLEDGMENTS

The author would like to thank Deborah Estrin and Scott Shenker for collaborating on an earlier version of this work, Jiangbi Lin for helping to perform some of the experiments, and the anonymous reviewers for suggesting clarifications.

REFERENCES

[1] J. M. McQuillan, I. Richer, and E. C. Rosen, "An Overview of the New Routing Algorithm for the ARPANET," in *ACM SIGCOMM*, January 1995, Proc. Sixth Data Communications Symposium, November, 1979.

[2] A. Khanna and J. Zinky, "The Revised ARPANET Routing Metric," in *ACM SIGCOMM*, September 1989.

[3] Z. Wang and J. Crowcroft, "Quality of Service Routing for Supporting Multimedia Applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, 1996.

[4] G. Apostolopoulos, R. Guerin, S. Kamat, and S. K. Tripathi, "Quality of Service Routing: A Performance Perspective," in *ACM SIGCOMM*, August 1998, pp. 17–28.

[5] H. F. Salama, D. S. Reeves, and Y. Viniotis, "Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, April 1997.

[6] G. Rouskas and I. Baldine, "Multicast Routing with End-to-End Delay and Delay Variation Constraints," in *IEEE INFOCOM*, 1996.

[7] H. Salama, D. Reeves, and Y. Viniotis, "A Distributed Algorithm for Delay-Constrained Unicast Routing," in *IEEE INFOCOM*, 1997.

[8] S. Chen and K. Nahrstedt, "Distributed QoS Routing with Imprecise State Information," in *International Conference on Computer Communications and Networks*, 1998.

[9] Q. Sun and H. Langendorfer, "A Distributed Delay-Constrained Dynamic Multicast Routing Algorithm," in *European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'97)*, 1997.

[10] G. H. Ash, A. H. Kafker, and K. R. Krishnan, "Servicing and Real-Time Control of Networks with Dynamic Routing," *Bell System Technical Journal*, vol. 60, no. 8, October 1981.

[11] 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.

[12] D. Mitra and J. Seery, "Comparative Evaluations of Randomized and Dynamic Routing Strategies for Circuit-Switched Networks," *IEEE Transactions on Communications*, vol. 39, no. 1, January 1991.

[13] B. R. Hurley, C. J. R. Seidl, and W. F. Sewell, "A Survey of Dynamic Routing Methods for Circuit-Switched Traffic," *IEEE Communications Magazine*, vol. 25, no. 9, September 1991.

[14] D. Mitra, R. 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.

[15] G. R. Ash and B. D. Huang, "An Analytical Model for Adaptive Routing Networks," *IEEE Transactions on Communications*, vol. 41, no. 11, November 1993.

[16] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The End-to-End Effects of Internet Path Selection," in *ACM SIGCOMM*, August 1999.

[17] D. Estrin, Y. Rekhter, and S. Hotz, "A Scalable Inter-Domain Routing Architecture," in *ACM SIGCOMM*, August 1992.

[18] L. Breslau, *Adaptive Source Routing of Real-Time Traffic in Integrated Services Networks*, Ph.D. thesis, University of Southern California, December 1995.

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

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

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

[22] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, "An Architecture for Wide-Area Multicast Routing," in *ACM SIGCOMM*, August 1994.

[23] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D.Estrin, and M. Handley, "The MASC/BGMP Architecture for Inter-domain Multicast Routing," in *ACM SIGCOMM*, October 1998.

[24] J. M. Akinpelu, "The Overload Performance of Engineered Networks With Nonhierarchical and Hierarchical Routing," *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 7, September 1984.

[25] D. Zappala, *Multicast Routing Support for Real-Time Applications*, Ph.D. thesis, University of Southern California, 1997.

[26] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network*, September 1993.

[27] M. Faloutsos, A. Banerjea, and R. Pankaj, "QoSMIC: Quality of Service Multicast Internet protoCol," in *ACM SIGCOMM*, August 1998.

[28] D. Zappala, "RSVP Loop Prevention for Wildcard Reservations," ISI memo, February 1996.

[29] D. Zappala, D. Estrin, and S. Shenker, "Alternate Path Routing and Pinning for Interdomain Multicast Routing," Tech. Rep. USC-CS-97-655, University of Southern California, June 1997.

[30] S. Deering, "Multicast Routing in Internetworks and Extended LANs," in *ACM SIGCOMM*, August 1988.

[31] J.D. Case, M. Fedor, M.L. Schoffstall, and C. Davin, "Simple Network Management Protocol (SNMP)," RFC 1157, May 1990.

[32] C. Cheng, R. Riley, S.Kumar, and J.J. Garcia-Luna-Aceves, "A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect," in *ACM SIGCOMM*, September 1989.

[33] B. Rajagopalan and M. Faiman, "A New Responsive Distributed Shortest-Path Routing Algorithm," in *ACM SIGCOMM*, September 1989.

[34] C. Hedrick, "Routing Information Protocol," RFC 1058, June 1988.

[35] S. McCanne, S. Floyd, and K. Fall, "LBNL Network Simulator," http://ee.lbl.gov/ns.

[36] M. Doar and I. Leslie, "How Bad Is Naive Multicast Routing?," in *IEEE INFOCOM*, 1993.

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

[38] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *IEEE INFOCOM*, 1996.