Alternate Path Routing for Multicast

Daniel Zappala, *Member, IEEE, ACM*

***Abstract*— Current network-layer multicast routing protocols build multicast trees based only on hop count and policy. If a tree cannot meet application requirements, the receivers have no alternative. In this paper, we propose a general and modular architecture that integrates alternate path routing with the network's multicast services. This enables individual multicast receivers to re-route a multicast tree according to their needs, subject to policy restrictions. Our design focuses on the two primary components of this architecture – a loop-free path installation protocol and a scalable, distributed path computation algorithm. Based on a simulation study, we demonstrate that using alternate path routing enables receivers to find acceptable paths nearly as well as a link-state protocol, with much lower overhead. We also show that our approach scales to large networks and that performance improves as a multicast group grows in size.**

***Index Terms*— multicast routing, alternate path routing, quality of service, performance evaluation**

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. This and other design choices were made in order to support applications with relatively elastic service requirements [3], such as data transfer (email, ftp, web browsing) and remote login.

Recently, there has been renewed interest in supporting a broader set of applications with real-time service requirements, such as conferencing and media broadcasts. In addition to scheduling, admission control, and resource reservation, new routing mechanisms are needed to support these types of applications. Current routing protocols compute paths based on hop count and policy, without regard to the capabilities of the path nor its current load.

The bulk of the work in this area has focused on designing and evaluating Quality of Service (QoS) routing protocols. 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. Computing a QoS-capable path in a scalable manner has proved to be very difficult since network conditions may change rapidly.

Our emphasis in this work is to serve all applications, not just those that need a QoS guarantee, by using the more general approach of alternate path routing. With alternate path routing, the network primarily uses shortest paths, and switches some load to longer paths only when the shortest paths are overloaded. In general, this model works well when the shortest paths are provisioned so that they handle most of the load in the network. Furthermore, a recent study by Savage et al. [4] indicates that alternate paths are available in the Internet, frequently with superior quality to the default path provided by current routing protocols. Alternate path routing will allow a wide range of applications to take advantage of these paths, including adaptive real-time applications and those using priority-based differentiated services.

The objective of this paper is to extend the Internet's current multicast services to include alternate path routing. The two primary components we discuss are path installation and path computation. Our main 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 meets these goals because shortest-path routing is used for most paths. In addition, both path installation and path computation use distributed, receiver-oriented mechanisms; we do not require global knowledge of the topology nor synchronization across group members.

This work has three major contributions:

- First, we create a general alternate path routing architecture that can be used for both real-time applications and Quality of Service routing. This architecture is the first to explicitly separate path installation and path computation into two components [5], [6]; QoSMIC [7] implicitly separates them, but was designed after this architecture was originally published and does not specify a path installation protocol. The modularity afforded by our routing architecture makes it easier to deploy and develop new routing services.
- Second, we design a general purpose path installation protocol called APM. APM allows receivers to re-route a multicast tree to avoid congested links by following a source route, and it avoids loops when multiple receivers re-route the tree at the same time. We identify conditions under which both QoSMIC and QMRP [8] are vulnerable to looping and show how APM detects and breaks these loops. This positions APM as a general-purpose protocol that can be used to install a QoS-capable route on behalf of any path computation protocol.
- Finally, we exhaustively evaluate a local path searching heuristic that uses only partial topology information to find feasible alternate paths. We demonstrate the effectiveness of this heuristic over a wide-range of topologies and loads. Local search can approximate the effectiveness of a link-state routing protocol, yet operate with much lower overhead.

D. Zappala is with the Department of Computer and Information Science, University of Oregon. This work was supported in part by the National Science Foundation under Grant ANI-9977524.

The rest of the paper is organized as follows. Section II examines related work in the areas of QoS routing and alternate path routing. Section III describes our routing architecture and how it interacts with the current multicast infrastructure. Section IV illustrates the design of our path installation protocol and Section V evaluates path computation heuristics based on limited searching. Finally, Section VI discusses our conclusions.

II. RELATED WORK

A. Alternate Path Routing

Alternate path routing has been used extensively in the telephone network to decrease the call blocking rate during overload. These protocols typically use a direct link if it is available, otherwise an alternate path consisting of one extra exchange is chosen [9], [10]. A common problem that must be avoided is that alternate path routing can decrease throughput at very high loads [11]. This occurs when calls using alternate paths block calls from being routed on a direct path. Many telephony algorithms solve this problem by using trunk reservation, in which a portion of each link is reserved for routing direct calls.

While this work can not be directly utilized for Internet routing (due to a more general topology and heterogeneous application requirements), the concept of using alternate paths has been applied toward the design of a variety of unicast routing protocols. One approach uses a link-state routing algorithm to rank paths according to preference [12]. Several distance-vector algorithms [13], [14] utilize alternate paths when a router detects congestion. Breslau [15] uses source routing to install alternate paths when the shortest-path route is congested. Most recently, Patek et al. [16] use alternate paths to achieve aggregate QoS for differentiated services.

B. Quality of Service Routing for Multicast

From the perspective of routing protocols, QoS research has concentrated primarily on extending unicast link-state protocols to include multicast routing. For unicast QoS routing, QoS characteristics for each link are distributed globally within link-state advertisements. A source is then able to compute a route that meets application requirements. A benefit of this approach is that it can be easily extended to multicast [17], [18]. Group membership is distributed along with the other link-state quantities, and a source computes a tree such that the QoS constraints of all receivers are met. However, link-state QoS routing protocols have the significant drawback of imposing very high overhead on the network. Reducing the overhead of link-state protocols limits the accuracy of the QoS computation and generally causes some degradation of performance. Some new techniques [19] may mitigate this effect.

One of the earliest distributed protocols for multicast QoS routing, developed by Sun and Langendorfer [20], uses path searching rather than a link-state protocol. In this work, a receiver contacts a router that is already on the multicast tree and has this router search for a QoS-capable path to the receiver. However, this work does not address how a router on the multicast tree is discovered, nor does it handle multiple receivers joining simultaneously.

A more promising approach is the QoSMIC [21], [7] protocol, which is based on YAM [22] and is designed specifically for multicast QoS routing. In QoSMIC, a new group member conducts a limited search by sending a multicast flood with a small hop count. If this search finds any routers already on the multicast tree, they respond with a "bid" that collects path characteristics for the shortest path between that point on the tree and the new member. Since the limited flood may not reach the multicast tree, the new member also has the option of asking a group manager to find routers on the multicast tree that are near it. Using either a centralized or distributed algorithm, the manager contacts several routers on the tree and asks them to send bids to the new member. The member then selects from all bids it receives and joins the multicast tree via this path.

Note that QoSMIC is not appropriate for unicast QoS routing, as finding a route depends on having a multicast tree to connect to. Similarly, when there are very few members, QoSMIC will have a hard time finding feasible paths. This makes bootstrapping the first few members rather difficult [23].

Another distributed protocol, QMRP [24], [8], uses a more targeted search than QoSMIC. In QMRP, a new member first tries to join the multicast group using the shortest path between itself and the root of the tree (either the core or the source). Along the way, the request checks available resources. If the application's requirements can't be met, the probe backtracks and *detours* copies of the request message to all adjacent nodes. These nodes then try to continue following their shortest-path route to the source of the multicast group, and continue as long as each link has available resources. Any time a reservation cannot be made, backtracking occurs again.

In its basic form, QMRP can explore an unlimited number of paths, which can lead to a high success rate but also very high overhead. Thus, the authors recommend limiting the search overhead by restricting the number of nodes that can detour a request message. This restricted form of QMRP is designated QMRP-n, meaning n nodes between the new member and source can be actively detouring a request message.

C. Limited Searching

The distributed QoS routing protocols described above take two different approaches toward limited searching. Both DCUR and QMRP tightly integrate path selection with path finding by checking resource availability as paths are explored. The advantage of this approach is that infeasible paths are quickly pruned and the overhead of searching is directed to paths that are likely to have the desired QoS characteristics. However, a significant disadvantage to this approach is that the path selection criteria must be universally standardized, making it difficult to upgrade and limited to only QoS-enabled applications.

A more flexible approach is to collect paths separately, then allow the host to apply its selection criteria. This is the approach we have taken in designing our limited search heuristics and is similar to the approach used by QoSMIC.

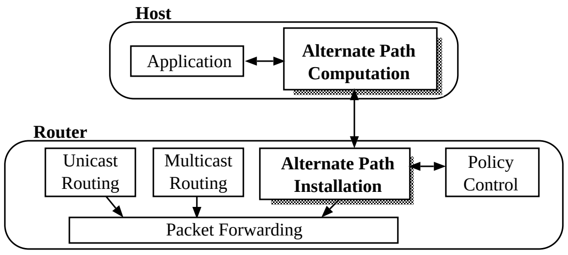

Fig. 1. Routing Architecture

This separation allows us to design general path finding heuristics that can be used for both alternate path routing as well as QoS routing. As new heuristics are developed they can be deployed on a host-by-host basis. Moreover, paths that are collected can be shared both spatially (among several members of the same multicast group) and temporally (for several multicast groups in succession), which reduces the overhead of path collection.

There are several differences between QoSMIC's limited search and the heuristics we propose (see Section V). The primary difference is that QoSMIC only conducts a search to find routers that are already on the multicast tree. Our heuristic uses the search to collect paths from nearby routers to the root of the multicast tree. This makes our search more efficient since we collect more information about the network at about the same cost. In addition to this difference, QoSMIC also requires configuration of a group manager to support joining the multicast tree. No such configuration is required in our architecture.

III. ROUTING ARCHITECTURE

Fig. 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 (DVMRP, MOSPF, PIM, SSM) or a shared tree with a single core (CBT, PIM, BGMP). The architecture supports multiple sources, as long as they each use a single core or their own shortest-path tree. 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 IV). 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 [11].

The two key features of our routing architecture that enable it to scale to large networks and large multicast groups are that (1) alternate path computation is distributed to group members and (2) the computation relies primarily on information that is local to the group member. Distributing the path computation means that each group member only needs to find a route between itself and the rest of the tree, rather than requiring the source to find routes to each member. In effect, the multicast routing problem is reduced to a set of unicast routing problems, allowing group members to use existing unicast routing protocols as the basis for a path finding algorithm. Furthermore, members use only local information, such as the status of previous reservation requests, local resource availability, and knowledge of the nearby topology. This enables scaling that would not be possible if group members needed to know information about the entire network.

Distributing the path computation also allows group members to incrementally refine their path finding algorithms. Group members do not need to globally agree on a metric or a particular algorithm, leaving them free to innovate as necessary.

One important part of our routing architecture that we do not explore within this paper is the policy control element. The job of this element is to screen source routes and prevent malicious, malfunctioning, or misguided group members 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 is part of our ongoing work and not within the scope of this paper.

Placing path computation under the control of hosts result in several significant advantages for our architecture. First, allows hosts to deploy new techniques for finding paths without requiring simultaneous upgrading of the routing infrastructure. Second, having hosts compute paths makes it easier to build a server infrastructure for sharing the path finding effort. This could be useful for the case when many hosts on the same subnet join the same group and are affected by the same congested link; a server could suppress many duplicate path computations. One potential drawback of this approach is that hosts may compute routes that do not obey routing policy. However, it is simple for the path installation protocol to check policy constraints as the route is installed. Such a policy check will likely be necessary anyway, particularly if the route crosses several domains.

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 [1] 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, at which point the application can restart its monitoring to determine whether the new path has improved its performance.

One strength of our approach is that alternate path routing as defined above is a general service that works with both unicast and multicast routing, regardless of whether QoS routing is supported. For QoS applications, RSVP [25] provides both service monitoring (e.g. compare the reservation made to the service received) and link identification (e.g. a reservation failure). Alternate path routing is then used by RSVP hosts to route around reservation failures. However, because RSVP is not widely deployed, it is important that alternate path routing can operate equally well without it. For multicast communication, both routers and hosts support *mtrace*, which can be used for link identification. Likewise, for unicast communication routers support *traceroute* for link identification and hosts may use a bandwidth-probing tool such as *b-probe* or *c-probe* to determine bandwidth availability along the path.

For the remainder of this paper, we focus on generic alternate path routing without QoS support, and assume some kind of link identification is available to hosts.

IV. 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 III. 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 (shortest-path) tree.

After alternate paths are installed, then data flows over a single multicast tree from the root to the group members. This tree is the combination of all branches of the alternate path tree, plus any branches from the default tree needed to reach members who are still using their shortest path. Fig. 2 illustrates this concept.

Receivers using alternate paths coexist safely with receivers who are using the default multicast tree. As can be seen from Fig. 2, receivers on the default 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 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

[1]*mtrace* is a tool that queries routers in a multicast tree for statistics such as packet rates and losses.

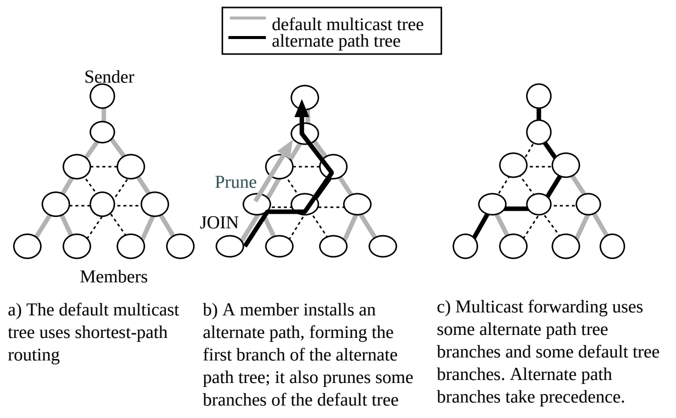
default multicast tree
alternate path tree
Sender
Members
Prune
JOIN
a) The default multicast tree uses shortest-path routing
b) A member installs an alternate path, forming the first branch of the alternate path tree; it also prunes some branches of the default tree
c) Multicast forwarding uses some alternate path tree branches and some default tree branches. Alternate path branches take precedence.

Fig. 2. Overview of alternate path installation

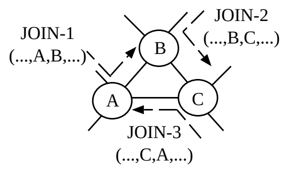

Fig. 3. Alternate path loop

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 [26]. A minor extension would allow a JOIN message to carry a source route. Fig. 3 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.

Depending 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.

As currently specified, QoSMIC [7] is vulnerable to this type of join loop. QoSMIC does not consider the possibility of multiple receivers joining a tree at the same time, and would need an additional mechanism to solve this problem. One possibility is that QoSMIC could use APM to install alternate paths. We note that a related looping problem was discovered and solved during our work with the design of RSVP [27], so it is not surprising to find it in other receiver-oriented multicast protocols.

The QMRP protocol [8] is vulnerable to a related ack loop, but the fix for this is simple to implement. In QMRP, when a member sends a REQUEST message upstream, an ACK is returned if the request is successful. When requests from several receivers are sent simultaneously, it is quite possible for them to overlap in the manner illustrated in Fig. 3. During the request phase, QMRP keeps separate state for each receiver, so these requests will not merge. However, when the requests are acknowledged, QMRP merges the multicast forwarding state and can, if the timing is right, perpetually send ACK messages around the loop. To prevent this problem from occurring, QMRP must be careful to (a) prune conflicting branches when it merges multicast forwarding state and (b) reject later ACKs that arrive on a different interface. By following these rules, a node that merges state will have only one incoming interface – the one on which the first ACK was received – and a second ACK from a different REQUEST message will be dropped. By pruning the interface established by the second REQUEST message, the node will also avoid receiving duplicate packets.

B. The APM Protocol

These design problems further underscore the importance of designing a path installation protocol that is provably loop-free. We now describe the APM protocol and illustrate how it detects and breaks loops. APM uses a novel caching mechanism to ensure that the source routes used by various members remain consistent.

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[2]. 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 Fig. 4(a).

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. Thus, 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 this router sends a MERGE message to its child. 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. Fig. 4(b) 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

[2]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.

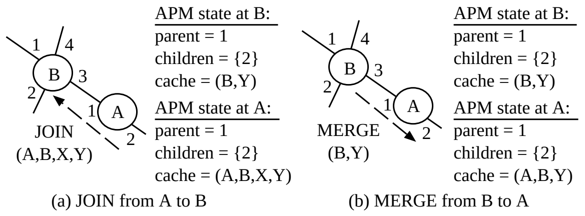

Fig. 4. APM JOIN and MERGE messages

any metrics that allow the network to quantify some paths as better than others, this is the best we can do. In [5] 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 IV-C proves this assertion, but for now we illustrate how loops are broken. Following the example of Fig. 3, router A will send a MERGE message with 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 IV-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 [5], we designed a setup protocol that prevents loops at the expense of increased latency; we abandoned this design when we determined that loop formation is a rare event.

2) Confirming the Join: While the MERGE message detects and breaks any JOIN message loops, 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. Fig. 5 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, MOSPF, and PIM, the most commonly-used multicast routing protocols.

At this time, the router also checks if the incoming interface

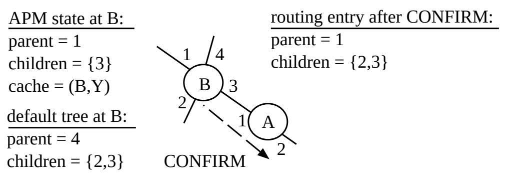

Fig. 5. APM CONFIRM message

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.

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 groups, 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.

To protect against transient failures, each router maintains a list of active neighbors for each of its interfaces. When a router is initialized, it sets each list to null and broadcasts a RESET message on each of its interfaces. Any neighbors that receive this message send an ACK and then treat the RESET as a FAILURE message, removing all branches that use the router as a parent.

A router that sends the RESET message collects ACKs and adds any such neighbor to its list of active neighbors for the given interface. Protocol messages are only processed for active neighbors; messages for inactive neighbors are always replied with a NACK. In this way, only active neighbors can create alternate routes through a given router, and state is kept consistent.

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

4) State Maintenance: Many recent protocols, such as PIM [26] and RSVP [25], 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 re-send the message if no acknowledgment is received after some time. Hard state is a natural and simple fit for APM given its features: fixed routes, simple failure detection, and loop-free path installation.

First, APM uses hard state because it does not adapt alternate paths to link and node failures. One of the primary uses of soft state in a protocol such as RSVP is to adapt to route changes automatically. However, if a given path is providing a user with good service, or if it has been reserved, then the user typically does not want that path to change unless it fails. When it does fail, the user or the application controls the next route that is installed. In fact, the original motivation for APM was to provide “route pinning” for RSVP [5].

Given this design, APM can avoid the overhead of periodic refresh messages that are sent by soft-state protocols. Instead, it is more efficient to use a HELLO message to detect link failures, as described above. This type of functionality is difficult to add to a soft-state protocol because the refresh message is used to build or refresh state for a particular group, whereas the HELLO message does not contain any group identifiers. State compression has been proposed for RSVP [28], but it adds significant complexity to the protocol and eliminates the ability to adapt to route changes automatically.

Finally, using hard state also allows APM to quickly detect and break JOIN loops, since each JOIN message must be acknowledged within a short period of time. This feature could also be added to a soft-state protocol, using a setup phase that is separate from the state refresh phase. However, this again adds complexity to a soft-state protocol, whereas hard state is a simpler and more natural fit.

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: Denote the root of the alternate path tree as router A_1, with a cached route of A_1 and the confirmed bit set. Clearly the root has a loop-free path to itself.

Now assume that router A_k also has a loop-free path to the root of the alternate path tree, designated as a cached route of $A_k, A_{k-1}, A_{k-2}, \ldots, A_1$ and the confirmed bit set. This router will send a CONFIRM message to its children containing the cached route. Thus, router A_{k+1} will set its cached route to $A_{k+1}, A_k, A_{k-1}, \ldots, A_1$ and set its confirm bit. We know that A_{k+1} is adjacent to A_k, and thus the path A_{k+1}, A_k is loop-free. By our previous assumption, we also know that $A_k, A_{k-1}, A_{k-2}, \ldots, A_1$ is loop free. Hence, the cached route at A_{k+1}, namely $A_{k+1}, A_k, A_{k-1}, A_{k-2}, \ldots, A_1$ is also loop-free.

Then, by induction every router A_i has a cached route and its confirmed bit set knows that its route represents a loop-free path to A_1. This indicates that APM installs and maintains a loop-free alternate path 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 $A_k, A_{k-1}, \ldots, A_1$ from the receiver to the root of the tree. If none of the routers in the path $A_k, A_{k-1}, \ldots, A_1$ 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, $A_k, A_{k-1}, \ldots, A_{j+1}$ and $A_j, A_{j-1}, \ldots, A_1$ The segment $A_k, \ldots, A_{j+1}$ is loop-free because it is a subset of the loop-free path $A_k, A_{k-1}, \ldots, A_1$. For the segment $A_j, A_{j-1}, \ldots, A_1$, 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 $A_k, A_{k-1}, \ldots, A_1$ is loop-free. ■

D. Using APM for QoS Routing

Because it is a general path installation protocol, APM can also be used to install paths for QoS routing. If all receivers use the same level of QoS, then the only additional mechanism required is to check for available resources at the same time the policy check is performed. If receivers can potentially use different levels of QoS, for example a tighter delay bound or a larger bandwidth reservation, then significant additions are needed. However, handling heterogeneous receivers is generally not considered an essential feature and significantly complicates the resource reservation protocol as well.

With this simple modification – checking for resource availability at each router during path installation – APM is suitable for use with any QoS path computation protocol. For example, QMRP or QoSMIC could be used to first find a feasible path, based on current resource availability, and then APM could be used to install this path. Likewise, the local search heuristic we discuss in the following section could be used to find feasible QoS routes.

The difference between using our local search versus QMRP or QoSMIC as the path searching protocol is that the former does not check the availability of QoS during searching while the latter do. This means our local search can potentially have a longer join delay, because it may take several installation attempts before a path with available resources is found. QMRP and QoSMIC, on the other hand, are more likely to find a feasible path with available resources, unless availability fluctuates rapidly. The advantage of our approach is that the path searching overhead can be much lower. Our ongoing work in this area confirms this benefit and also demonstrates that join delay with the local search option is only high at very high loads [23].

V. 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 III, 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 conduct a search of the network to find a path that avoids congested links. We assume that the network is large and therefore an exhaustive search is not possible.

In our approach, the receiver conducts a *limited search* by collecting paths from routers and forming a *partial map* of the network. The receiver uses the partial map to compute alternate paths in the same way that a link-state protocol uses a full map to compute shortest paths. With our limited search algorithms, all communication occurs between the receiver and individual routers. We explicitly avoid any message passing between routers so that path computation can be deployed relatively easily. In addition, since path computation is purely a local operation, new techniques can be deployed incrementally.

In the first part of this section, we explore two kinds of limited searches – local and randomized – and illustrate why we choose local search as the preferred path computation heuristic. In the second part, we demonstrate how local search is able to compete favorably with a link-state algorithm, even though it uses only partial information about the network. The advantages of local search are particularly evident in large networks. Finally, we examine the performance of local search with respect to multicast group size.

A. Limited Search Heuristic

Because path computation is distributed, each receiver only needs to find an alternate path from itself to the rest of the tree, without taking into account any other receivers. This means that path computation can leverage existing unicast routing protocols to find alternate paths. The only added functionality required by our limited search heuristic is the ability for a host to ask a router for the path from that router to some other node in the network.

The limited search heuristic consists of the following steps:

1) At startup, the path computation component (PCC) creates a partial map consisting of the paths from itself to a set of nodes called the $SearchSet$. For a local search, $SearchSet$ consists of all routers within n hops. For a randomized search, $SearchSet$ consists of m random routers.
2) As described in Section III, a service-monitoring component identifies congested links used by an application and notifies the PCC. The PCC will in turn add congested links to its map, if necessary, and mark them as congested.
3) When an alternate path is needed for some multicast tree, the PCC contacts each router in the $SearchSet$ and collects the unicast path from itself to a given router and from the router to the root of the tree. The PCC adds these paths to its partial map and runs Dijkstra's shortest path algorithm on the map to find the shortest path to the root that avoids links marked as congested. If such a path is found, then it is returned as the result of the heuristic.
4) If no such alternate path is found, the PCC asks the root of the tree to find an alternate path. The root of the tree runs the same limited search heuristic from its end and returns the results. If neither the receiver nor the root finds a path, then the limited search heuristic returns a null path.

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 [5]. 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 k minutes. During this time, the root can suppress excess queries by multicasting a command to receivers that tells them to back off.

The limited search heuristic can be tuned independently by each receiver (and the root) to favor either lower latency (search time) or lower communication overhead (number of nodes queried). To optimize for latency, each of the routers in the $SearchSet$ can be queried in parallel. To decrease communication overhead, the routers can be queried consecutively, starting with those that are nearest. This will decrease overhead if an alternate path is found before all routers in the $SearchSet$ have been queried. For local search, a useful compromise is to use an expanding-ring search [29] to first query the routers within 1 hop, then those within 2 hops, etc.

TABLE I
UPPER BOUNDS ON LATENCY AND COMMUNICATION OVERHEAD FOR LOCAL SEARCH

Type of Search	Latency	Overhead
Consecutive	μt	μ
Parallel	t	μ
Expanding-Ring	nt	μ

$$\mu = \begin{cases} c\frac{(c-1)^n-1}{c-2}, & c > 2 \\ 2n, & c = 2 \end{cases}$$

TABLE II
UPPER BOUNDS ON LATENCY AND COMMUNICATION OVERHEAD FOR RANDOM SEARCH

Type of Search	Latency	Overhead
Consecutive	mt	m
Parallel	t	m

Tables I and II list the upper bounds on latency and communication overhead for local and random search, respectively. The tables consider 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, n is the number of hops for local search, and m is the number of nodes queried for random search.

These bounds are closely related to the size of the $SearchSet$. For random search this is simply m. The size of the $SearchSet$ for local search is denoted as μ; a loose upper bound on μ is simply c^n. A tighter bound can be derived by observing 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. Generalizing to n hops of search, we find that $\mu = c\sum_{i=0}^{n-1}(c-1)^i$. Based on the rule that $\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1}$, $r \neq 1$, we can reduce this to $\mu = c\frac{(c-1)^n-1}{c-2}, c > 2$. When $c = 2$, then $\mu = 2n$.

For local search, the communication overhead bound is the same for all types of searches. Whereas the communication overhead for parallel search will always be at the maximum, the others will have a lower average-case overhead. We use expanding-ring search in our later simulations to illustrate this case.

The limited search heuristic can be readily implemented by a host. A list of routers within n hops can be composed by using recursive SNMP queries. Routers can be chosen randomly by probing the IP address space or by choosing from a list of routers known to the routing protocol. Unicast paths can be collected by simply querying a router for its path to the root of the tree. If the router is using a path-vector protocol or a distance-vector protocol with source tracing [30],

TABLE III
NETWORKS AND THEIR CHARACTERISTICS

Networks	Nodes	Type	Degree	Diameter
F100 Set	100	flat	4.18-4.96	6-8
T100 Set	100	transit-stub	3.26-3.94	10-12
MBone	4178	map	-	-
T5100	5100	transit-stub	3.51	18

[31], then it already has the path stored in its routing tables. If a more simple distance-vector protocol such as RIP is being used, the router 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 evaluate the limited search heuristic under a wide range of topologies and across a full spectrum of load. The topology models we use include a flat random networks and transit-stub networks generated with the Georgia Tech ITM software [32]. Flat networks are created using the Doar-Leslie edge-connection method (a variation of the Waxman method) 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.

Table III lists the topologies we used and their characteristics. The bulk of our simulations use two sets of 100-node networks, one flat (F100) and one hierarchical (T100). Each set contains 10 networks, so that our results are not dependent on a particular topology. To test the scalability of limited search, we use two additional large networks. One is a map of the *MBone* topology [33] from 1999 collected by researchers at ISI. The other is a 5100-node transit-stub network, with some extra transit-stub and stub-stub links to provide some redundancy in the network. For these large networks our results use only a single topology.

We focus the bulk of our attention on the 100-node networks because they 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, these two models are quite useful in illustrating the extremes of local search behavior.

We also 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 the limited search heuristic is independent of whether the receiver or a router does the path computation.

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 may benefit from path computation and installation done by their neighbors.

Moreover, all of our experiments underestimate the success rate of our path computation heuristic. This is because a given receiver will attempt to find an alternate path for only a single sender. In practice, path computation servers will likely compute alternate paths for an entire site, so that path discovery may be shared among all the receivers at that site. This will increase the success rate since a given request may benefit from paths acquired for earlier requests. Likewise, computation servers can decrease load by allowing some requests to be fulfilled without any searching.

All of our simulations were conducted within a highly-optimized version of the original LBNL network simulator *ns* [34], allowing us to examine performance on much larger networks than are typically used in the networking community.

C. Evaluating Limited Search Options

Our first set of simulations demonstrates that (1) local search outperforms randomized search and (2) asking the sender (or the root of the multicast tree) is crucial in hierarchical networks. In all, we test the following six variations of limited search:

n-Hop
Use local search extending to n hops.

n-Hop+Sender
Same as above, but ask the sender for help if unable to find a route locally.

m-Random
Use randomized search with m routers.

m-Random+Sender
Same as above, but ask the sender for help if unable to find a route locally.

n-Hop+m-Random
Use both local and randomized search, with m routers chosen randomly from among those not within n hops.

n-Hop+m-Random+Sender
Same as above, but ask the sender for help if unable to find a route locally.

To examine the effectiveness of these different variations, we designed a workload that tests how well the limited search heuristic is able to find alternate paths in a lightly-congested network. For a given topology, a simulation begins by choosing one sender and one receiver. We then iterate through each of the links between the sender and receiver, designating each time a single congested link. The receiver then tries to find an alternate path around that link. For each sender-receiver pair, the simulation prints the number of links

TABLE IV
LIMITED SEARCH: ONE NETWORK OF SET F100

Variation	Average Success Rate
1-Random	0.69
1-Hop	0.77
1-Random+Sender	0.87
1-Hop+1-Random	0.91
1-Hop+Sender	0.93
2-Hop	0.97
1-Hop+1-Random+Sender	0.98
2-Hop+Sender	1.00

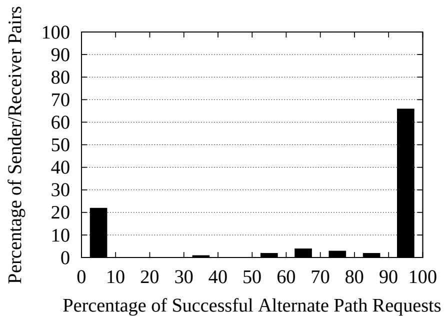

Fig. 6. 1-Hop Success Rate: histogram for F100

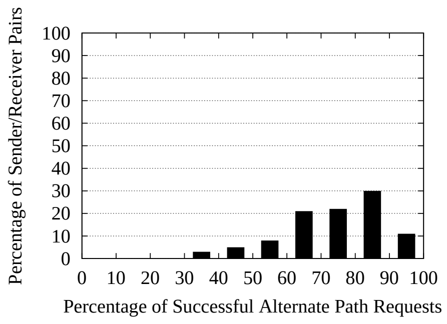

Fig. 7. 1-Random Success Rate: histogram for F100

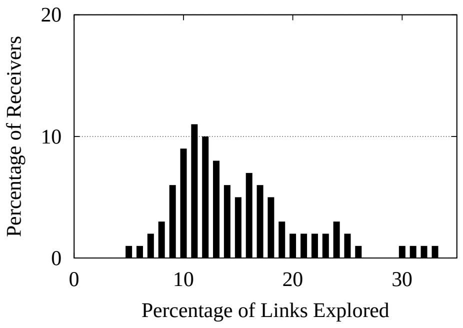

Fig. 8. 1-Hop+1-Random Overhead: histogram for F100

tried and the number of times an alternate path is found. For variations using randomized search, we repeated each sender-receiver pair 100 times to compute an average of these numbers.

Overall, for each of the generated topologies, we ran simulations for 100 sender-receiver pairs and computed the success rate. We did not count links for which there was no alternate path available, i.e. where even a link-state algorithm with full knowledge of the topology would not find an alternate path. Because of the exploratory nature of these results, we used only one of the 100-node flat networks and one of the 100-node transit-stub networks.

1) Flat Network: On the flat network, all the variations of limited search perform well and have limited overhead. Compared to randomized search, local search has the advantage of finding shorter alternate paths.

Table IV shows the average success rate for each of the limited search variations one network from the F100 set. By using a 2-Hop local search, or by combining local search with randomized searching or asking the sender, the limited search heuristic is able to have a success rate well over 90%.

An important difference between local and randomized search is seen when examining the distribution of the success rate among various sender-receiver pairs. Fig. 6 and 7 show histograms where we group the alternate path attempts of each sender-receiver pair and calculate the overall success rate of each pair. The 1-Hop heuristic has some pairs that always find alternate routes and others that never do. On the contrary, querying a single random router will always help to find some alternate paths in a flat random network.

The overhead of these heuristics is tied to the node degree of the network. The 1-Hop search typically explores between 5-15% of the links in the network, but can visit as many as 30% of them. Because the randomized search only explores one node, it's maximum overhead will be one third of the 1-Hop search; indeed, it always explores less than 10% of the links. These results are consistent with the degree of the flat network, which is about 4; a node will have on average 3 neighbors, resulting in 3 paths explored for the 1-Hop heuristic (as opposed to 1 path explored with 1-Random).

Finally, the 1-Random search finds longer alternate paths than the 1-Hop search. Nearly 90% of the paths found by 1-Hop are the same length as the shortest-path route or 1 hop longer, and the longest path is 2 hops longer. In contrast, the paths found by 1-Random can be as many as 7 hops longer than the shortest-path route, with most of them 1-3 hops longer.

By combining local and randomized search, we can realize some of the benefits of both approaches. In terms of success rate, nearly 70% of all sender-receiver pairs can always find an alternate path and all except for about 5% of the receivers are below 50%. The combined heuristic typically explores between 10-20% of the links in the network, as shown in Fig. 8. While this is a significant fraction of the 100-node network, it is much less than for a link-state algorithm and demonstrates the validity of our approach. As we show later, the overhead for larger networks is much smaller, generally less than 1%. Finally, while the combined heuristic can have alternate paths that are 6 hops longer than the shortest path, nearly 80% are at most 1 hop longer.

Our results also show that a pure random heuristic can perform nearly as well as the combined heuristic on the flat topologies. For example, 3-Random also allows about 70% of the sender-receiver pairs to always find an alternate path.

TABLE V
LIMITED SEARCH SUCCESS RATE: ONE NETWORK OF SET T100

Variation	Average Success Rate
1-Random	0.07
1-Random+Sender	0.14
1-Hop	0.35
1-Hop+1-Random	0.41
2-Hop	0.41
1-Hop+Sender	0.68
1-Hop+1-Random+Sender	0.74
2-Hop+Sender	0.77

However, the overhead for 3-Random is slightly higher; it visits 15-25% of the network links.

2) Transit-Stub Network: On the transit-stub network, the only successful variation is the one that combines local search with asking the sender (N-Hop+Sender). Table V shows the average success rates for each of the variations. Using N-Hop+Sender enables a receiver to find alternate paths around local bottlenecks, either within the vicinity of the receiver or near the sender. To further confirm this observation, 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 84% and 92%, respectively.

Randomized search has a particularly difficult time finding alternate paths in this topology because of its strict hierarchy. The 1-Random search finds an alternate path only 7% of the time, and even a 4-node random is successful less than 20% of the time. The reason for this poor performance is that a random search will generally not explore local links where congestion is likely to occur. Because they are chosen randomly, the sender and receiver will likely be in different stub networks, as will the router that is queried by the heuristic. Thus the map the receiver accumulates will not provide additional details of the sender or receiver's stub networks, but will more likely help to find alternate paths within the backbone.

For searching overhead and path length, our results for T100 are very similar to F100. The N-Hop+Sender variation explores fewer links in the transit-stub topology than plain N-Hop on the flat topology – most receivers using 1-Hop+Sender explore less than 10% of the links with 17% as the highest. Likewise, most receivers using 2-Hop+Sender explore less than 20% of the links. As with the flat network, alternate paths found with the N-Hop+Sender variations are shorter than those using randomized searching.

D. Examining Local Search in Depth

Our second set of simulations demonstrates that (1) local search is competitive with a link-state routing algorithm, while exploring only a fraction of the network topology, (2) local search naturally limits alternate path length during high load, and (3) local search finds most paths after only one hop during low load.

Due to its promise, as shown in our first set of simulations, we focus the rest of our evaluation on the performance of local search with the option of asking the sender for help. For brevity, we refer to this variation as simply local search. In addition to its superior performance in hierarchical networks, we prefer local search because randomized searching involves querying routers that may not belong to the same administrative authority and thus may not be well supported.

In this section, we evaluate local search as a function of load by varying the number of congested links in the network. We begin by randomly selecting a fixed percentage of links and mark them as congested, then randomly choosing a sender and a single receiver. The receiver measures the shortest path and determines whether a link on this path is congested. When congestion is found, the receiver tries to find an alternate path. When a feasible alternate path is found, the receiver installs it with APM, then measures this new path for congestion. This process is repeated until the receiver exhausts all of its choices for alternate paths.

For these simulations, we continue to 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 performed by other receivers. The following section looks at larger group sizes.

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 test the local search heuristic with expanding-ring search and describe the trade-offs involved in exploring more hops. Our metrics include the effectiveness of the search, the length of the alternate paths found, and the overhead incurred.

For success rate, we plot the percentage of receivers whose path from the source is not congested 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 most of our simulations, we test 300 receivers on each of 10 randomly-generated topologies, for a total of 3000 repetitions. When we use an MBone topology, we use a single network because the topology represents an actual network and is not randomly generated.

Because current practice indicates that links between stub and transit networks are most likely to be congested, we have run additional simulations (not reported here) using higher congestion probabilities for these types of links. For these simulations, our results are very similar to what we report here.

1) Flat Networks: Under light load on the F100 set of flat networks, local search performs nearly as well as a link-state algorithm with full knowledge of the topology. As shown in Fig. 9 the effectiveness of local search improves as the number of hops in the expanding-ring search is increased. Using 2-Hop local search, a receiver is able to find alternate paths

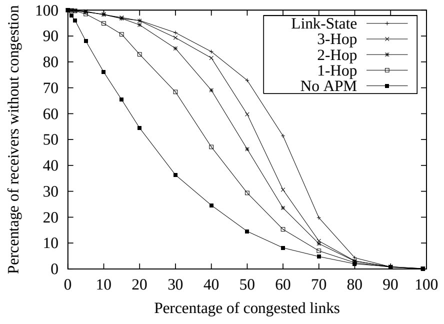

Fig. 9. Congestion Rate: F100

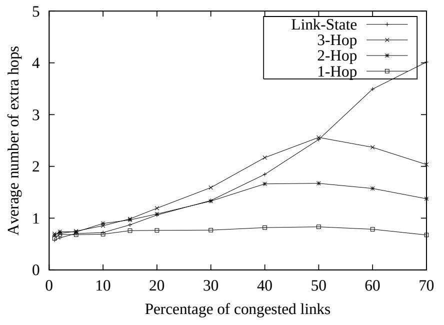

Fig. 10. Alternate Path Length: F100

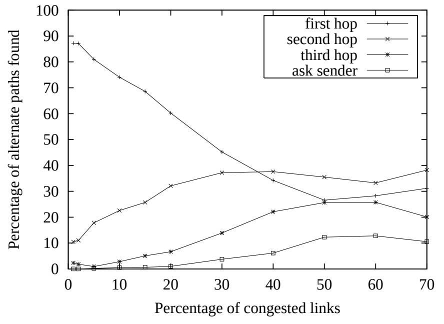

Fig. 11. Search Breakdown: 3-Hop, F100

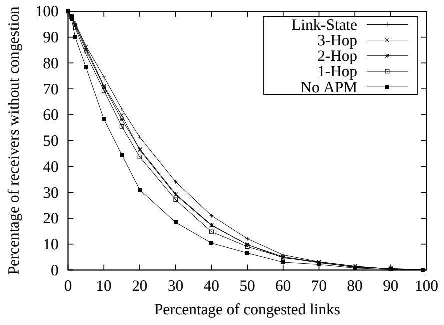

Fig. 12. Success Rate: T100

with success nearly equal to that of the link-state algorithm under light load (less than 15% congested links).

Under heavy load, the local search algorithm naturally avoids finding long paths. This is exactly the situation we want: 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 [11]. Fig. 10 illustrates this advantage by showing the average path length for the alternate paths found by each of the algorithms, 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. Local search naturally limits path length because it can only explore other nodes' shortest paths; at high load these paths are less likely to be congested if they are short. Note that the maximum path length for 3-Hop is 6 hops longer than the shortest path, so an average at or below 3 hops is expected.

Regardless of load, the number of links explored by local search is fairly constant, depending primarily on the number of hops that are searched and the network degree. For the 2-Hop search the average number of links explored is between 10 and 20, while for a 3-Hop search it averages between 30 and 40. While these numbers appear high for a 100-node network, they are small when compared to a link-state algorithm. Furthermore, this overhead is independent of the size of the network; as we show later in this section the percentage of links explored in larger networks is near 1%. As low as this overhead is, it is important to note that all of our simulations overestimate overhead because we do not re-use the topology information collected by local search for later alternate path attempts. Moreover, if a network manager wants to limit alternate path routing under high load, it is simple to do by simply refusing to respond to a local search query during high load.

Finally, our results also show that, under light loads, most alternate paths are found within one or two hops. Fig. 11 shows the percentage of alternate paths that are found during each step of the 3-Hop heuristic. We also note that the heuristic resorts to asking the sender less than 20% of the time, mostly under moderate loads. By comparing this graph to those where more hops are explored (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 trade-off 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 itself (multiple requests are suppressed as discussed above), instead of having each receiver doing a large search.

2) Transit-Stub Networks: On the T100 set of transit-stub networks, 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. We also find that the the overhead of local search is small and that asking the sender becomes very important in a hierarchical network.

Fig. 12 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 most only about 10% worse than with the link-state algorithm.

With regard to both overhead and path length, the performance of local search improves on the transit-stub networks.

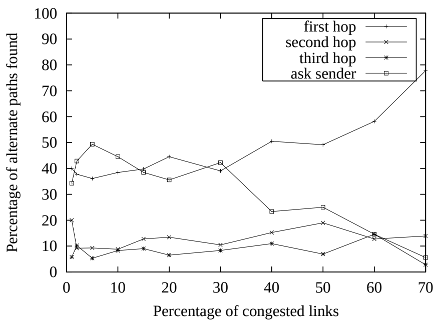

Fig. 13. Search Breakdown: 3-Hop, T100

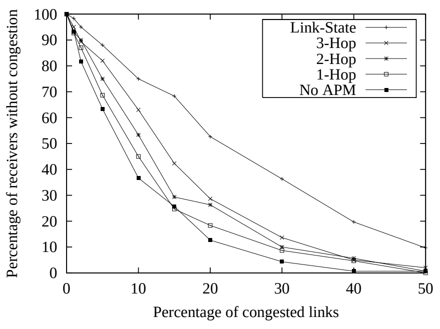

Fig. 14. Success Rate: MBone map

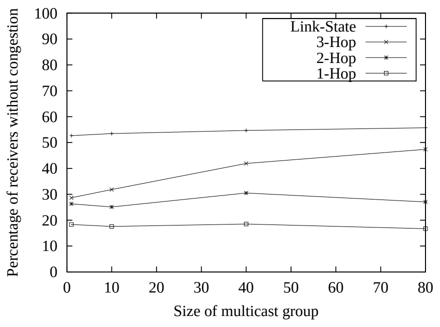

Fig. 15. Success Rate by Group Size: 20% congestion, MBone

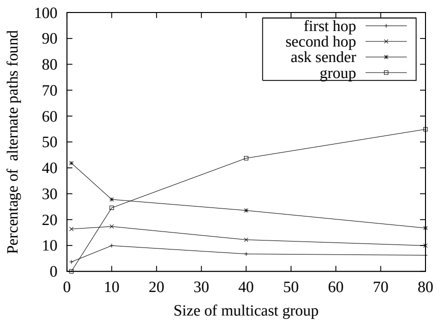

Fig. 16. Search Breakdown by Group Size: 3-Hop, 20% congestion, MBone

Due to the hierarchy of these topologies, the overhead of searching is greatly reduced compared to the flat network. The 1-Hop search explores an average of 6% of the links while the 3-Hop search explores about 11%. In addition, path lengths for local search average about 1 extra hop.

Our results also show that asking the sender is very important on hierarchical networks. Fig. 13 shows the percentage of alternate paths that are found during each step of the 3-Hop heuristic. Because the sender and receiver are usually located in separate stub networks, a path is equally likely to be found locally or by asking the sender during low loads. 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 as the sender 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.

3) Large Networks: On both the MBone map and the T5100 transit-stub network, the local search heuristic continues to perform nearly as well as link-state routing during low load. Fig. 14 shows the success rate of local search on the MBone map. When 5% of the links are congested, using a link-state algorithm satisfies about 88% of the receivers, while a 3-Hop search satisfies about 83%. This is a very good result, considering using no alternate path routing results in a success rate of about 63%.

Most importantly, this performance was achieved by exploring only a small fraction of the network topology. For the two large networks, the percentage of links discovered by local search ranges from 0.1% for 1-Hop to about 1% for 3-Hop.

E. Exploring the Effect of 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. If two or more members share a congested link, then the computation and path installation done by one member may re-route the tree for some or all of the other affected members. Thus, the larger a group, the easier it becomes to approximate link-state path computation with only a small local search.

This effect is illustrated in Fig. 15, which plots the effectiveness of alternate path routing versus group size on the MBone map at the congestion level of 20%. At the group size grows, a 3-Hop local search becomes nearly as effective a full link-state computation. This validates a key component of our design: alternate path routing for multicast can be effectively distributed to receivers

One of the key benefits of distributing the alternate path computation is that a significant portion of group members get alternate paths for "free" due to some other group member installing a route that helps them. This is illustrated in Fig. 16, which shows how paths are found for the 3-Hop heuristic in the MBone map with 20% congestion. As the group size grows, an increasing percentage of alternate paths are found and installed by some other member of the group. This benefit decreases the overhead of local search as fewer nodes need to perform a computation.

The only negative effect seen when adding more receivers is that the average path from a sender to a receiver may increase as the group grows in size. However, this effect is worst during high levels of congestion, when alternate path routing is less likely to be used, due to trunk reservation [11].

VI. CONCLUSIONS

This paper proposes a general and modular architecture that integrates alternate path routing with the network's multicast services. This architecture can accommodate both our proposed alternate path routing protocols as well as QoS routing protocols.

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 evaluation of the local search heuristic demonstrates that it can find alternate paths nearly as well as a full link-state routing protocol, using only partial information about the network. Most impressively, the protocol scales very well to large networks, exploring only a small fraction of the links in the network.

We have also demonstrated that as a multicast group grows in size, less searching is needed to find alternate paths, as the computation done by one group member benefits others. This decreases the overhead of limited search while only slightly increasing alternate path length.

Finally, as the network becomes more congested, searching for alternate paths becomes less effective. This is acceptable since experience with the telephone network indicates alternate path routing should be disabled during periods of high load.

In summary, alternate path routing promises to be a useful method for scalably meeting multicast application requirements. We are interested in applying this approach to current research in the area of application-layer multicast.

ACKNOWLEDGMENTS

The author would like to thank Deborah Estrin and Scott Shenker for collaborating on an earlier version of this work and the 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 *Proc. Sixth Data Comm. Symp.*, November 1979, reprinted ACM SIGCOMM, January, 1995.

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

[3] S. Shenker, "Fundamental Design Issues for the Future Internet," *IEEE J. on Select. Areas Comm.*, vol. 13, no. 7, September 1995.

[4] 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.

[5] D. Zappala, "Multicast Routing Support for Real-Time Applications," Ph.D. dissertation, University of Southern California, 1997.

[6] ——, "Alternate Path Routing for Multicast," in *IEEE INFOCOM*, 2000.

[7] S. Yan, M. Faloutsos, and A. Banerjea, "QoS-Aware Multicast Routing for the Internet: The Design and Evaluation of QoSMIC," *IEEE/ACM Transactions on Networking*, vol. 10, no. 1, February 2002.

[8] S. Chen, K. Nahrstedt, and Y. Shavitt, "A QoS-Aware Multicast Routing Protocol," *IEEE J. on Select. Areas Comm.*, vol. 18, no. 12, Dec. 2000.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] D. J. Nelson, K. Sayood, and H. Chang, "An Extended Least-Hop Distributed Routing Algorithm," *IEEE Transactions on Communications*, April 1990.

[14] Z. Wang and J. Crowcroft, "Shortest Path First with Emergency Exits," in *ACM SIGCOMM*, September 1990.

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

[16] S. D. Patek, R. Venkateswaran, and J. Liebeherr, "Enhancing Aggregate QoS through Alternate Routing," in *IEEE Globecomm*, 2000.

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

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

[19] F. Hao and E. W. Zegura, "On Scalable QoS Routing: Performance Evaluation of Topology Aggregation," in *IEEE INFOCOM*, 2000.

[20] Q. Sun and H. Langendorfer, "A Distributed Delay-Constrained Dynamic Multicast Routing Algorithm," in *European Workshop on Interactive Dist. Multimedia Sys. and Telecomm. Services*, 1997.

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

[22] K. Carlberg and J. Crowcroft, "Building Shared Trees Using a One-To-Many Joining Mechanism," in *ACM Comp. Comm. Review*, Jan. 1997.

[23] D. Zappala and D. Zhou, "Performance Evaluation of Path Searching Heuristics for Multicast QoS Routing," in *IEEE Inter. Conf. on Comp. Comm. and Networks (ICCCN)*, October 2002.

[24] S. Chen, K. Nahrstedt, and Y. Shavitt, "A QoS-Aware Multicast Routing Protocol," in *IEEE INFOCOM*, 2000.

[25] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReserVation Protocol," *IEEE Communications Magazine, 50th Anniversary Issue*, May 2002, reprinted from IEEE Network, September 1993, Volume 7, Number 5.

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

[27] D. Zappala, "RSVP Loop Prevention for Wildcard Reservations," Internet Draft: work in progress, February 1996.

[28] L. Wang, A. Terzis, and L. Zhang, "RSVP Refresh Overhead Reduction by State Compression," Internet Draft: work in progress, March 2000.

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

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

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

[32] K. Calvert and E. Zegura, "Georgia Tech Internetwork Topology Models," Software at http://www.cc.gatech.edu.

[33] I. S. Project, "Mbone Map from 23 February 1999," http://www.isi.edu/scan/mbone.html.

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