Internet Draft
Expires September 26, 1997
File: draft-zappala-multicast-routing-mech-00.ps

Daniel Zappala
USC Information Sciences Institute
March 1997

A Route Setup Mechanism For Multicast Routing

March 26, 1997

Status of Memo

Abstract

This document describes a multicast route setup protocol that can be used to install alternate paths and pinned routes when they are requested by receivers. We describe the protocol, derive some of its properties, and address its applicability to unicast route setup.

1 Introduction

This document describes a multicast route setup protocol that can be used to install alternate paths and pinned routes when they are requested by receivers. This protocol is designed as part of the interdomain multicast routing architecture described in [7]. In general, this protocol is useful when multicast routers wish to install explicit routes in a multicast tree without coordinating the routing of the entire tree according to a globally defined metric. Thus, in addition to being used as prescribed in [7], this protocol may also be used to install a QoS route for a receiver. We have focused on designing a multicast route setup protocol; a later section describes the relevance of our work to unicast route setup.

For the purposes of this document, we assume that receivers use a reservation protocol such as RSVP [8, 2] to reserve resources for unicast and multicast flows. By default, these reservations are obtained over an opportunistic, shortest-path multicast tree computed and installed by a multicast routing protocol, likely either DVMRP [6], MOSPF [5], PIM [4] or CBT [1]. Each sender may have its own tree, or all senders may use a shared tree. Throughout this document we assume sender trees, although the mechanism is equally applicable to shared trees.

We also assume that a receiver, or some entity acting on behalf of a receiver, may request several services in place of its current opportunistic route:

- *Alternate Path*: A route that is an alternative to the currently installed route. A receiver may wish to use an alternate path when it is unable to reserve resources along its current path.
- *Pinned Route*: A route that remains unchanged unless a node along the route fails. A receiver may wish to know that once it has secured a reservation, the route will not change unless it fails, and hence the reservation will likely remain in place. When an application does not use a pinned route (the route is opportunistic), the reservation protocol must adapt the reservation whenever the route adapts to a shorter path, even if the original path is still working.

Using these basic services, a receiver may ask for an alternate path that is opportunistic, an alternate path that is pinned, or it may ask to pin its current route. Note that an opportunistic alternate path has some pinned hops while the remaining hops are opportunistic; see [7] for more details.

As part of the architecture described in [7], we assume that a receiver asks its first-hop router for an alternate path or a pinned route. This router in turn contacts a local route construction agent to construct a route and encodes the response as an explicit route. The setup protocol connects the receiver to the multicast tree along this new path. Along the way, the setup protocol pins any hop that is listed in the route; thus if the receiver wants a pinned route, then every hop between the receiver and the sender must be listed.

Table 1: MORF Protocol Messages

Messages	**Parameters**
Setup(*Group*, *Target*, *Route*)	*Group* : multicast group
Failure(*Group*, *Target*, *SetupRt*, *TreeRt*)	*Target* : sender or core
Teardown(*Group*, *Target*)	*Route* : explicit route
	SetupRt : route from Setup
	TreeRt : route used by tree

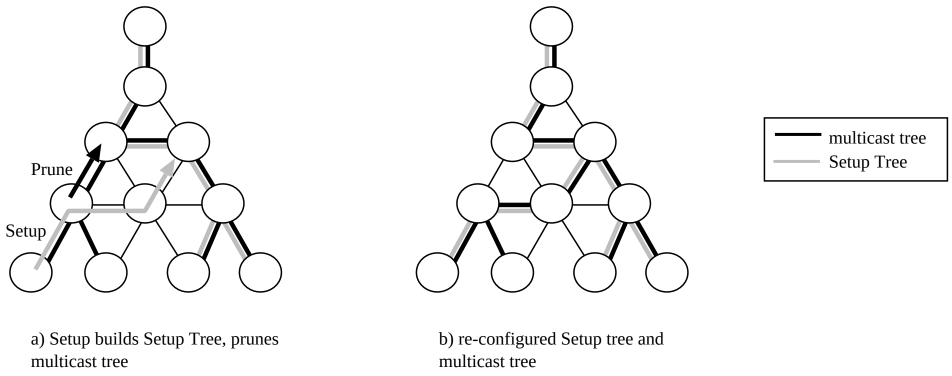

Figure 1: Using a Setup Message to Install a Route

2 The MORF Multicast Route Setup Protocol

We have designed the MORF multicast route setup protocol to install routes provided by local route construction agents. For each multicast tree built by the multicast routing protocol, MORF creates its own parallel multicast tree consisting of alternate paths and pinned routes. Each branch of this tree, called the Setup Tree, is constructed using a Setup message originated by a leaf router on behalf of local receivers. The Setup message contains an explicit route indicating the path the Setup should travel (Table 1). As the Setup travels upstream, MORF notifies the multicast routing protocol that it is overriding some local portions of the multicast tree with some branches in the Setup Tree. The multicast routing protocol adds these branches to the multicast tree and prunes any conflicting branches from the original tree (Figure 1a). The resulting multicast tree reflects the path installed by MORF (Figure 1b). The multicast tree may be for a single sender [4], or multiple senders may rendezvous via a core [4, 1]. In either case, the protocol is the same; in the following discussion we refer to sender-based trees for simplicity.

Since the Setup Tree overrides default opportunistic routing, each router in the Setup Tree must have a mechanism to detect failures of an alternate path or a pinned route. The setup protocol may rely on a unicast routing protocol to exchange query messages with its neighbors to determine whether they are alive, or it may use its own similar mechanism. Once a failure is detected, MORF sends a Teardown message both upstream and downstream of the failure to remove failed branches from the Setup Tree (Figure 2a). At each hop,

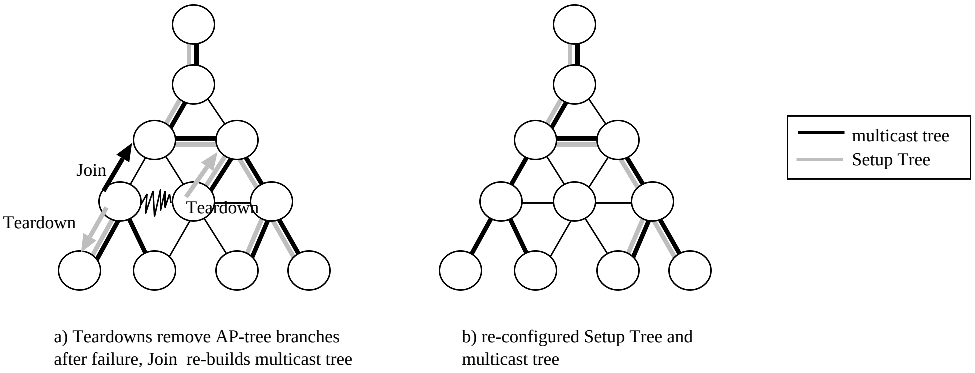

Figure 2: Using a Teardown to Remove a Failed Route

MORF notifies the multicast routing protocol of the branches it is removing. The multicast routing protocol re-builds the multicast tree to reflect its metric, often the shortest path to the sender (Figure 2b).

The above examples represent the simplified case when a Setup does not conflict with the rest of the Setup Tree. However, the setup protocol must also resolve Setup messages from different leaves that use conflicting routes, because leaf routers may use independent route construction agents. MORF resolves conflicts by choosing the first route that is installed for any given branch of the tree. Where subsequent routes meet this branch, they must conform to the route used from that point upward toward the source. If the setup protocol does not follow this restriction, then a number of looping scenarios may arise; Section 2.1 discusses these cases and the manner in which they are prevented.

Figure 3 shows an example of how all Setup messages except the first one must match the route already used by the Setup Tree. When a Setup message adds a node to the Setup Tree, it caches the route it will use to travel from that node upward toward the sender. If a subsequent Setup message arrives at that node, it compares the remaining route it must travel to the route cached locally. If the routes do not match, the node stops processing the Setup and sends a Failure message downstream (Figure 3a). The Failure message contains the route used by the failed Setup and the route used by the tree from the rejecting node upward (Table 1). A router receiving a Failure message merges the two routes it contains to construct a new route that will match the tree, then sends a second Setup with this route (Figure 3b).

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

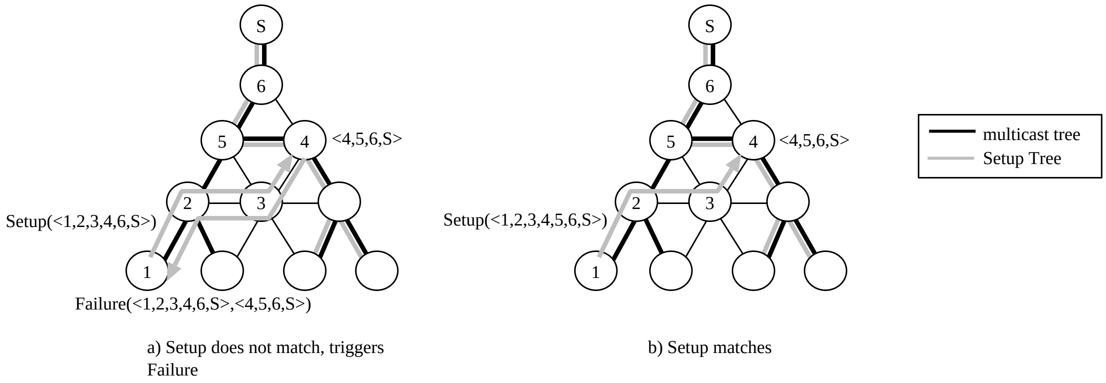

Figure 3: Matching Setup Messages

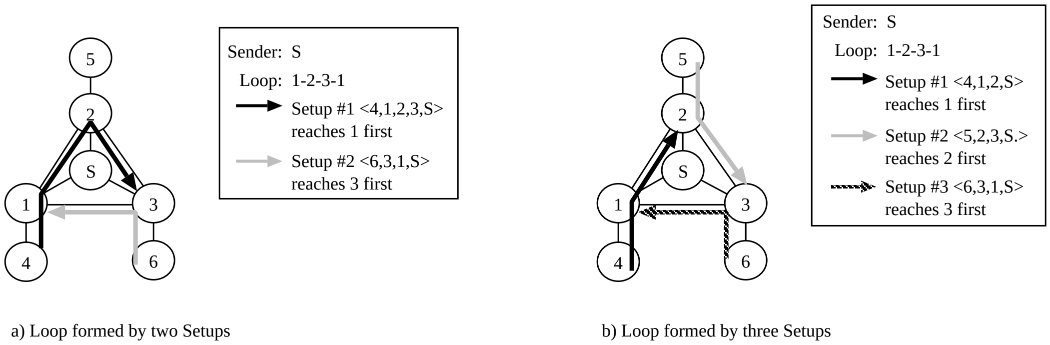

Figure 4: Loops Formed by Setup Messages

2.1 Looping Scenarios

When Setup messages are not restricted to matching the rest of the Setup Tree, a number of possible looping scenarios arise. Figure 4a shows two Setups, each using a strict explicit route. Based on their order of arrival, as shown, if the Setups merge they form a loop. This loop can be prevented if nodes A and C compare the routes and detect the loop will form. However, when three joins are involved, as in Figure 4b, a single node cannot prevent the loop from forming without having more information available.

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

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

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

We discuss each of these in turn. Due to the dynamic nature of multicast trees, a node may

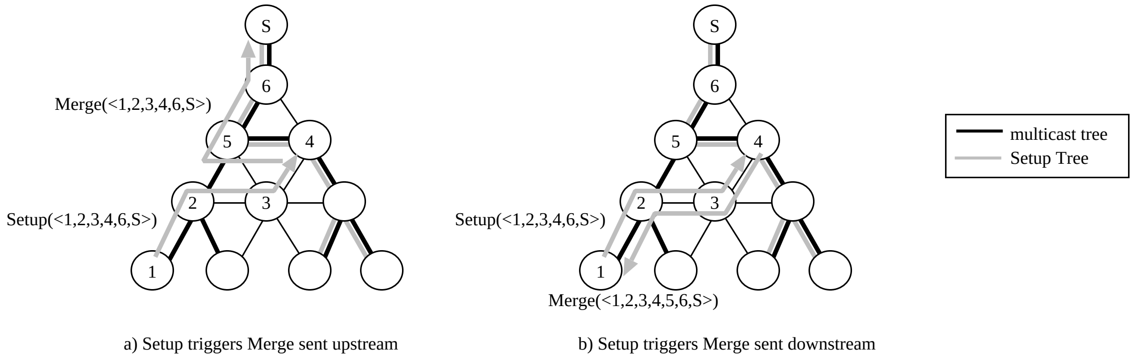

Figure 5: Merging Setup Messages Instead of Matching

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

One approach to keep nodes updated concerning upstream and downstream merges is to distribute information after each merge. Following solution (1) above, each Setup that merges can send a Merge message upstream containing its route (Figure 5a). Every node can then know all its descendants and thereby detect any loops. Alternatively, in keeping with solution (2) above, each Setup that merges can send a Merge message downstream containing the upstream portion of the route it merged with (Figure 5b). This allows every node to detect loops by knowing all its ancestors. We denote these two mechanisms as *Merge Up* and *Merge Down*, respectively. In both of these approaches, information distributed by the Merge messages may be stale, so loops such as that shown in Figure 4 may still form temporarily before being broken.

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

2.2 Analysis of Setup Mechanisms

Table 2 compares the setup mechanisms discussed above when building a single multicast tree, assuming there is no packet loss and that one receiver joins the tree at a time. The columns listing message and storage overhead consider the behavior of each mechanism at

Table 2: Comparison of Setup Mechanisms

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

a single node. Overhead in these cases is expressed in terms of a, the number of ancestors of a node, or d, the number of descendants of a node. The setup latency column lists time in terms of the number of trips taken between a receiver and the multicast tree.

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

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

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

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

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

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

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

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

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

2.3 Unicast Route Setup

Previous work has studied the efficacy of using source routing to support unicast real-time applications [3]. One way to use source routes to provide alternate paths or pinned routes is to embed the source route in an application's packets. Assuming the route will be inserted by a filter at a sender's nearest router, no modifications to applications will be needed. However, because many routers currently delay processing of source routed packets, this mechanism may not be applicable to applications with strict delay requirements.

An alternative is for the sender's nearest router to insert a label in the application's packets rather than a source route. This label can reference an alternate path or pinned route that is installed using MORF. Because unicast applications involve only one receiver, the setup latency will only be 1 trip. Either the sender or receiver can initiate the route setup, although initiating at the sender will require trivial modifications to the protocol.

3 Acknowledgments

Bob Braden, Deborah Estrin, and Scott Shenker provided valuable feedback on this work.

References

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

[2] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification". work in progress, November 1996.

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

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

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

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

[7] Daniel Zappala, Bob Braden, Deborah Estrin, and Scott Shenker. "Interdomain Multicast Routing Support for Integrated Services Networks". work in progress, March 1997.

[8] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. "RSVP: A New Resource ReSerVation Protocol". *IEEE Network*, September 1993.

Security Considerations

Security considerations are not discussed in this memo.

Author's Address

Daniel Zappala
USC Information Sciences Institute
4676 Admiralty Way, Floor 10
Marina del Rey, CA 90292
EMail: daniel@isi.edu