Internet Draft
Expires December 30, 1995
File: draft-ietf-rsvp-routing-00.ps

Deborah Estrin
USC/ISI
Scott Shenker
PARC
Daniel Zappala
USC/ISI
June 1995

Routing Support For RSVP

June 30, 1995

Status of Memo

Abstract

This memo describes an interface between RSVP and routing protocols. The interface allows RSVP to request information and services from routing in the form of an asynchronous query-reply protocol.

1 Introduction

This document describes an interface between RSVP [4] [3] and routing protocols. The interface allows RSVP to request information and services from routing in the form of an asynchronous query-reply protocol. This document describes the design of the interface and details a specification of a portion of the interface based on our experience using RSVP with DVMRP [2] as a supporting multicast routing protocol. The routing services we discuss represent only a subset of the functionality that RSVP may need; we are continuing to investigate other services. This document assumes familiarity with RSVP.

First, Section 2 presents a brief overview of relevant RSVP functionality and some routing problems RSVP encounters. These problems lead to a set of RSVP's requirements for routing, presented in Section 3. Section 4 details a general routing model that RSVP can use to interact with a variety of different routing protocols. Section 5 outlines the particular queries and responses that comprise the interface between RSVP and routing, focusing on how RSVP uses this interface. Section 6 discusses the costs and limitations placed on routing if it implements the services requested by RSVP. Finally, Section 7 details the specification of the interface messages.

2 RSVP Overview

Using RSVP, sources send *Path* messages hop-by-hop to the destination. Each router uses the *Path* message to create reverse-path routing state and forwards the message toward the destination. Receivers send *Resv* messages toward sources, following the reverse-path routing state. Each router uses the *Resv* message to query admission control and accept or reject the embedded reservation request. All of the routing and reservation state is *soft-state*; RSVP refreshes the soft-state periodically and removes it after a period of inactivity.

Note that after RSVP processes a *Path* message at a router, it re-sends it as if it originated from the original sender. For multicast destinations, the forwarding entry depends on both the source and the destination; thus RSVP can only resume forwarding of the *Path* message if it knows the necessary routing entry. RSVP may also need to send a different *Path* message out each outgoing interface on multicast routers. It needs this functionality to establish reverse-path routing information, to advertise service characteristics [1] over different branches of a multicast tree, to send multicast Path messages over non-RSVP clouds, and to pack multicast Path messages. Thus, for this functionality, RSVP may also need to know the necessary routing entry.

RSVP has no knowledge of route changes, other than by capturing them with periodic *Path* messages. In addition, once a route changes, RSVP has no guarantee that the new path can be reserved at a level equal to the old one. Thus, route changes may lead to a long-term service disruption. Figure 1 shows an example of how this may happen. Receiver

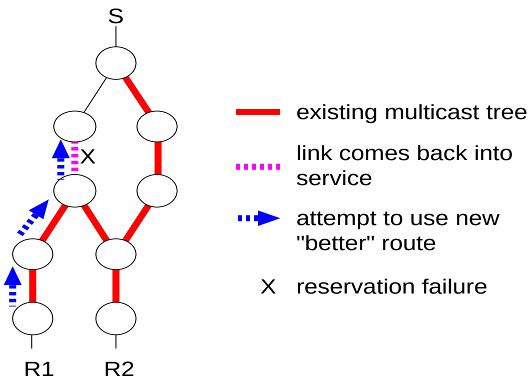

Figure 1: Service Disruption Caused by Route Change

R1 has reserved a branch of the existing multicast tree. A link comes back into service, causing routing to find a new shortest path. Multicast routing adapts to this change and the reservation protocol sends a reservation up the new route. The reservation fails, however, and R1 is left without a reserved route.

RSVP is also limited to establishing reservations over whichever path routing chooses. Most routing protocols maintain a single, shortest path between a source and destination. We refer to this shortest path as the *default route*. If RSVP is unable to attain a reservation over the default route, then it has no recourse.

3 RSVP Requirements

To address these limitations, we have designed an interface between RSVP and routing. Using this interface, RSVP may acquire routing entries to allow it to send control messages hop-by-hop, as well as request routing services that provide it with extra functionality. We express the interface design as a set of requirements that RSVP places on routing. The initial set of requirements includes:

- supplying on demand the forwarding entry for a source-destination pair,
- providing notification of route changes for supplied routes,
- keeping a working route in place once a reservation has been secured, barring any failed links or routers along the route, and
- constructing on demand explicit routes when a reservation request is rejected along the default route.[1]

[1] We use the term explicit routes in place of source routes because they may be used by receivers as well as sources.

We expect this list of requirements to grow as we continue to investigate how routing can support RSVP.

RSVP needs to learn about routing entries for the reasons discussed in Section 2. Note that for unicast routing RSVP does not need to learn about routing entries; routing is based on destination only and not on the source-destination pair. However, in order to provide a common interface to all routing protocols, we do not distinguish between unicast and multicast destinations.

RSVP requires notification of route changes so that it can adapt its reservations to changes in the route between a source and destination. It can do this by re-sending *Path* or *Resv* messages where needed. For multicast destinations, a route change consists of any change in the multicast tree for a source-group pair, including prunes and grafts as well as routing changes due to failed or recovered links. If routing can not support route change notification, then RSVP must poll routing for route entries in order to adapt to route changes.

Once RSVP secures a reservation over a route, it may request that routing "pin" the route by not adapting to route changes other than those caused by link or node failures. By providing this service, routing allows RSVP to provide applications with a service commitment that is interrupted only by failure along the committed path. If routing can not support route pinning, then RSVP must adapt to routing changes and may notify applications that its service commitment is interruptable. As part of this service, routing may also provide "smooth switching" from a reserved non-default route to a reserved default route. This may allow routing to reduce its overhead of maintaining non-default routes while still satisfying RSVP's requirement.

Finally, if RSVP is unable to attain a reservation over the default route, it may request that routing provide an alternate route between the source and destination. By providing this service, routing and RSVP can cooperate to find a route that satisfies a receiver's service request. Routing can choose routes based on feedback from RSVP concerning previous reservation availability. Routing also has the option of rejecting an alternate path request if there are no available paths. Figure 2 shows an example of using an alternate route. Receiver R2 has already reserved a branch of the existing multicast tree. Receiver R1 joins the tree, but its reservation attempt fails. Routing supplies a new route and the second reservation attempt succeeds.

4 RSVP Routing Model

Because RSVP must learn about routing entries from a variety of different routing protocols, we have adopted portions of the DVMRP routing model [2] as an abstraction through which RSVP can communicate with all routing protocols. A routing entry for a source-destination pair consists of an incoming virtual interface and a set of outgoing virtual interfaces. A virtual interface is simply a number that routing creates to refer to each of the interfaces

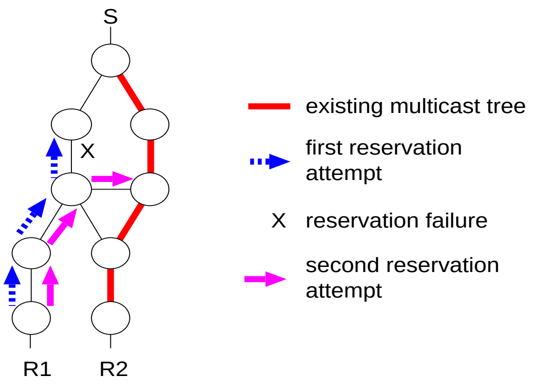

Figure 2: Using Alternate Routes

(or virtual interfaces) it uses to send and receive packets on the router. Note that the implementation of the virtual interface is hidden from RSVP; whether the interface is a real interface, a pseudo-device, or a tunnel is irrelevant.

When RSVP receives a packet, it expects the forwarding engine to tell it which virtual interface the packet arrived on. This allows RSVP to suppress forwarding of packets that routing has determined have arrived via the wrong incoming virtual interface, while still allowing local processing. When RSVP sends a packet, it expects that it can tell the forwarding engine to forward the packet directly over a particular vif, without performing any of the forwarding engine's usual routing checks or lookups. Together, these functions allow RSVP to forward distinct packets hop-by-hop over each link in a unicast or multicast path between a source and destination(s).

The RSVP routing model defines a virtual interface (or vif) using the following attributes:

id	a unique identifier,
threshold	a TTL threshold,
status	a bitmask status vector, and
local_addr	a local address.

RSVP uses the TTL threshold to control the scope of a control message. The status vector currently only defines whether the virtual interface has been administratively disabled.

5 Routing Support

The following sections describe the routing support messages exchanged by RSVP and routing.

5.1 Obtaining Routing Entries

Before RSVP can obtain routing entries, it must first discover which virtual interfaces (vifs) routing is using. It does this by issuing an Initial Query:

$$Initial\_Query().$$

Routing responds with an Initial Reply that includes the number of vifs and a list of vifs as defined by the RSVP routing model:

$$Initial\_Reply(num, vif\_list).$$

Once RSVP has received the Initial Reply, it can begin requesting routing entries by sending a Route Query for a source-destination pair:

$$Route\_Query(source, destination).$$

Routing responds by sending a Route Reply that includes the incoming vif and an outgoing vif bitmask:

$$Route\_Reply(source, destination, incoming\_vif, outgoing\_vif\_bitmask).$$

5.2 Route Change Notification

RSVP can request notification of route changes by sending a Route Query with an additional notification flag:

$$Route\_Query(source, destination, notification).$$

By setting the notification flag in the query, RSVP requests that routing provide route change notification. If routing is able to provide this service, it sets a corresponding notification flag in the Route Reply, otherwise it clears the flag. If RSVP receives a Route Reply with the notification bit set, it can assume that routing will notify it when the routing entry for the source-destination pair changes. In the meantime, RSVP can use the routing entry indefinitely.

5.3 Route Pinning

Once RSVP has reserved a route, it may request route pinning by sending a Service Query to the last-hop router using a service bitmask to indicate the service requested:

$$Service\_Query(source, destination, service\_bitmask, vif).$$

The Service Query may optionally include the vif between the last-hop router and the host to indicate where the service begins.

The service bitmask codes two types of service: route pinning and smooth switching. If routing can provide route pinning, then RSVP can assume that the current route from the source to the destination will remain unchanged until a link or node along the route fails. If routing can provide smooth switching, then RSVP can assume that routing will not switch from the pinned route to a default route until (a) the default route is reserved or (b) the pinned route fails. Smooth switching requires considerable coordination between RSVP and routing; whether this is a viable service is related to whether the network has enough information to know how to respond, and whether another route reserved for the same per-hop service will provide the same end-to-end QoS.

If RSVP requests route pinning, then routing attempts to pin the route and returns a Service Reply in the same format with the service bitmask set to indicate whether it could perform the desired service. Routing may take a relatively longer time to return a Service Reply than other replies because it may have to contact other routers to determine whether it can provide the requested service. In addition, if routing returns a Service Reply indicating the route is pinned, it may later send an unsolicited Service Reply indicating that the route is no longer pinned. This may occur if a link or node fails, or if other constraints prevent routing from continuing to pin the route.

5.4 Alternate Routes

If RSVP fails to obtain a reservation over the default route, it may request an alternate route by sending a Service Query to the last-hop router as defined above. The service bitmask indicates RSVP's request for an alternate route. Routing returns a Service Reply indicating in the service bitmask whether it has provided the requested service.

To provide routing with feedback on the reservation status of routes, RSVP sends routing status reports to routing:

$$Status\_Report(source, destination, status, vif, status\_info).$$

At last-hop routers, RSVP tells routing whether a reservation over the current route was successful, using a status of "success" or "failure." The status info may contain failure information or the level of reservation attained; routing can use this as input to its route selection process.

RSVP also sends routing status reports to help it when coordinating alternate path requests. At every router, RSVP tells routing whether it has reserved the incoming link for a particular route, using a status of "unreserved" or "reserved." Routing assumes that all links are unreserved until told otherwise. When a receiver joins a multicast tree, it may re-route unreserved branches of the tree if it carries an explicit route for a different branch. However, receivers may not re-route reserved branches; routing should maintain stability for receivers that have already obtained a reservation.

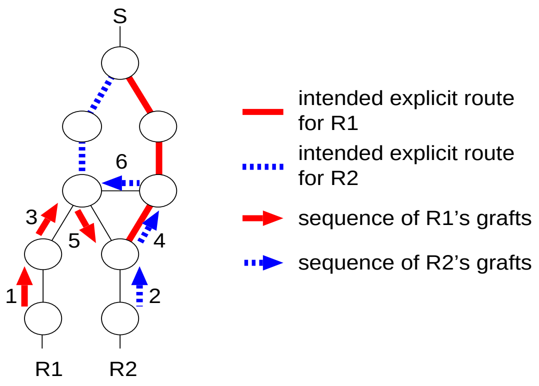

Figure 3: Black Hole With Explicit Route

6 Service Costs and Limitations

Each of the functions described above introduces routing costs and limitations.

Routing incurs very little cost for providing route change notification; essentially it only has to tag the subset of its routing entries for which RSVP is interested in receiving notification. This amounts to keeping an extra bit for each routing entry. Since this service can be provided independently by each router, its implementation is not subject to any interoperability constraints.

Providing route pinning introduces substantial costs for some routing protocols. If a routing protocol uses aggregation and maintains routes based on a group of sources (for example a subnet), then pinning a route requires extra source-specific state that would not otherwise be kept. Routing must continue to maintain the aggregated route for non-pinned sources.

Routers that implement route pinning will also encounter some basic operational limitations of the service. For example, pinning a portion of the multicast tree for one receiver has the side effect of pinning those links for all other downstream receivers. Some protocols may require that all routers between the source and destination must support route pinning in order to provide the service. In addition, if a router starts to pin a route during a route change, it may pin the wrong route. In some situations, routing may pin the route back onto itself, forming a black hole. The routing protocol needs added protocol mechanisms to keep pinning from disrupting the integrity of routing.

Routers that implement alternate path routing encounter the same costs and limitations as with route pinning. One significant difference is that the likelihood of forming black holes is much greater. Consider Figure 3, where receivers R1 and R2 both want to use an alternate route for a multicast tree of sender S. Routing uses an explicitly-routed graft for each receiver. Depending on the sequence of links that the two grafts travel, they may form a black hole in the multicast tree.

Because routing must manage to prevent black holes in order to provide both route pinning and alternate path routing, we present a solution here for routing protocols that implement these services using grafts. When a receiver requests a pinned route or an explicit alternate route, the last-hop router sends a graft containing the service request upstream along the intended route. Each router that receives the graft creates the appropriate pinning or alternate path state and attaches to this state a "connected" bit. The bit is initially cleared, signifying that the graft has not yet propagated to the sender or to another connected portion of the tree. Once the graft reaches the sender or another connected portion of the tree, the entire downstream branch becomes "connected" to a valid portion of the route. Routing sends a message downstream to set all of the appropriate connected bits. Any graft that reaches an "unconnected" portion of a route must be rejected and sent back downstream to erase any state.

7 Routing Support Specification

This section details the format of messages exchanged by RSVP and routing. We refer to this simple protocol as Routing Support for Resource Reservations (RSRR). We only include those messages that are currently being exchanged by the RSVP prototype and DVMRP. We do not include message formats for the Service Query, Service Reply, or Status Report because we do not yet have enough implementation experience with them.

7.1 RSRR message format

```
An RSRR message consists of:

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Version       | Type          | Flags         | Num           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|...                                                            |
|                                                               |

Version
        Routing Support for Resource Reservations Version.  This
        document specifies version 1 of RSRR.

Type
        This document defines four message codes for RSRR:

                1 = Initial Query
                2 = Initial Reply
                3 = Route Query
```

```
        4 = Route Reply
```

The Flags field, the Num field, and the rest of the message, are defined separately for each RSRR code.

7.2 Initial Query

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Version       | Type          | Flags         | Num           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version as defined above.

```
Type
        1 = Initial Query

Flags, Num
        0
```

7.3 Initial Reply

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Version       | Type          | Flags         | Num           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vif ID-1      |Vif Threshold-1| Vif Status-1                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vif Local Address-1                                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|...                                                            |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vif ID-N      |Vif Threshold-N| Vif Status-N                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vif Local Address-N                                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version as defined above.

```
Type
        2 = Initial Reply

Flags
```

```
        0

Num
        Number of vifs being reported

Vif ID-N
        ID for the Nth vif

Vif Threshold-N
        The threshold ttl for the vif; an outgoing message must have a
        ttl greater than the threshold to be sent

Vif Status-N
        A bit vector representing the vif status.  Currently only
        the Disabled bit is defined:

        +-+-+-+-+-+-+-+-+
        |             |D|
        +-+-+-+-+-+-+-+-+

        D = 1 if vif is administratively disabled, 0 otherwise.

        The rest of the field is transmitted as zeroes.

Vif Local Address-N
        The local address of the physical interface corresponding to the
        vif
```

7.4 Route Query

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Version       | Type          | Flags         | Num           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Destination Address                                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Source Address                                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Query Identifier                                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Version as defined above

Type
```

```
        3 = Route Query

Flags
        Currently only the Notification Bit is defined:

        +-+-+-+-+-+-+-+-+
        |             |N|
        +-+-+-+-+-+-+-+-+

        N = 1 if RSVP requests route change notification for this query,
        0 otherwise.

        The rest of the field is transmitted as zeroes.

Num
        0

Destination Address
        Group address being queried

Source Address
        Source address being queried

Query Identifier
        Identifier used by reservation protocol
```

7.5 Route Reply

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Version       | Type          | Flags         | Num           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Destination Address                                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Source Address                                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Query Identifier                                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Incoming Vif                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Outgoing Vif Bitmask                                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version as defined above.

```
Type
        2 = Route Reply

Flags
        Currently only the Error bit and the Notification bit
        are defined:

        +-+-+-+-+-+-+-+-+
        |           |E|N|
        +-+-+-+-+-+-+-+-+

        N is set if N is set in the corresponding route query and N can
        perform route change notification for the query.  Otherwise N is
        cleared.

        E is set if routing is unable to obtain routing information for
        the route query.  Otherwise E is cleared.

        The rest of the field is transmitted as zeroes.

Destination Address
        group address of query = group address of reply

Source Address
        source address of query = source address of reply

Query Identifier
        identifier used by reservation protocol, copied from query message

Incoming Vif
        incoming Vif for (S,G) or default (S,*) if no group-specific
                state; invalid if E bit is set

Outgoing Vif Bitmask
        bitmask of outgoing Vifs for (S,G) or default (S,*) if no
                group-specific state; invalid if E bit is set
```

8 Acknowledgments

We would like to thank Lixia Zhang for repeating the word "requirements" over and over again. We would also like to thank Bob Braden and Van Jacobson for their ideas, Bill Fenner for his help with the DVMRP implementation, and the IETF community for valuable

feedback.

References

[1] S. Shenker. “Network Element Service Specification Template”. Internet Draft, March 1995.

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

[3] L. Zhang, R. Braden, D. Estrin, S. Herzog, and S. Jamin. “Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification”. Internet Draft, March 1995.

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

Security Considerations

Security considerations are not discussed in this memo.

Authors' Addresses

Deborah Estrin
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0871
EMail: estrin@usc.edu

Scott Shenker
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
EMail: shenker@parc.xerox.com

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