Internet Draft
Expires January 1, 1999
File: draft-ietf-rsvp-routing-02.ps

Daniel Zappala
University of Oregon
Jeff Kann
USC/ISI
June 1998

RSRR: A Routing Interface For RSVP

July 1, 1998

Status of Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

Abstract

This memo describes version 2 of RSRR, a routing interface for RSVP. By using this interface, RSVP may obtain forwarding information from routers and use it to place reservation state within the network. Version 1 of this interface was designed primarily for RSVP interaction with IPv4 multicast routing protocols. Version 2 adds support for IPv4 unicast as well as IPv6 unicast and multicast routing. A backwards compatibility mechanism is provided.

1 Introduction

This document describes Routing Support for Resource Reservations (RSRR), an abstract interface by which RSVP [5] may obtain routing information. RSVP is a resource reservation protocol used by hosts to request Quality of Service from the network. RSVP does not include a routing protocol; instead it may use any underlying routing protocol(s) to determine where it should carry resource reservations. The RSRR interface provides RSVP with forwarding table entries and notifies it when those entries change. This document elaborates on the routing support described in the RSVP spec [2].

This document describes version 2 of RSRR. In addition to IPv4 multicast routing protocols, which were covered extensively in the version 1 document, version 2 explicitly addresses IPv4 unicast routing and adds support for IPv6 unicast and multicast routing. Version 2, like version 1, also provides RSVP with information on the interfaces known to RSRR. Finally, version 2 includes a backwards-compatibility mechanism that allows RSVP to determine the version of RSRR that a routing daemon is using.

Section 2 gives an overview of RSVP functionality and its requirements for routing and kernel support. Section 3 describes RSRR as an abstract service interface. Section 4 describes how this interface might typically be implemented within RSVP. Appendix A gives a specification of RSRR used for communication between RSVP and routing daemons. Appendix B lists the RSRR version 1 specification, which version 2 implementations must also support. This is included for reference, since some implementations may support multiple versions. Appendix C lists the implementation status of RSRR.

2 RSVP Overview and RSRR Motivation

Using RSVP, sources send *Path* messages hop-by-hop to the destination (Figure 1a). 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 (Figure 1b). Each router uses the *Resv* message to query admission control to accept or reject the embedded reservation request. Reservation messages utilize various *styles* to allow sharing among different senders. For example, the shared-explicit style targets a reservation at a list of senders, and the wildcard style targets a reservation at all upstream senders (Figure 1c).

With proper operating system support, a router could pull an RSVP *Path* message out of the forwarding engine, do RSVP processing, and then resume forwarding the message. However, multicast *Path* messages have unique requirements. After RSVP performs its processing, it may need to send a different copy of the *Path* message out each outgoing interface. Normally, IP multicast replicates datagrams and sends identical copies out each of the outgoing interfaces. Thus, RSVP does not use regular IP multicast forwarding and

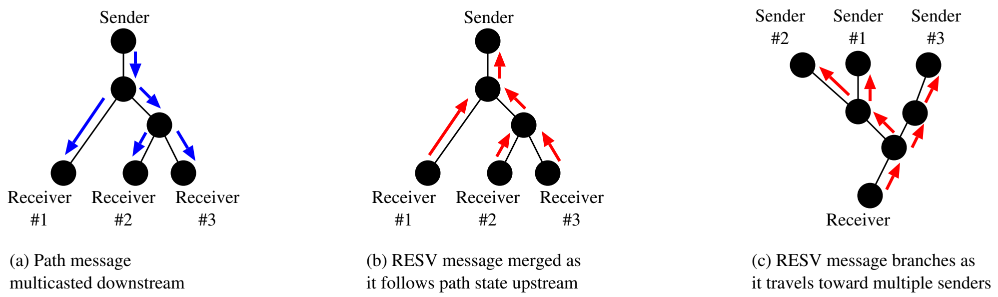

Figure 1: RSVP Overview

instead needs to replicate the *Path* messages on its own and simulate its own multicast forwarding.

To accomplish this objective, RSVP needs to obtain forwarding table entries from the routing protocol; this is what the RSRR interface is for. In addition, the operating system must support several basic functions for RSVP. First, RSVP needs to know which interface a *Path* message arrives on. This allows RSVP to suppress forwarding of multicast packets that routing has determined have arrived via the wrong incoming interface. It also needs to receive all multicast *Path* messages, even if they arrive on the wrong interface; this allows RSVP processing to occur for local receivers. Second, RSVP must be allowed to insert the original source address of the *Path* message in its IP header This allows the *Path* message to be processed by downstream routers as if it came from the original source. Finally, when RSVP sends a multicast packet, it expects that it can tell the operating system to forward the packet directly over a particular interface, without performing any of the usual multicast routing checks and without looping the packet back to other interfaces. These operating system functions, combined with RSRR, allow RSVP to forward distinct packets hop-by-hop over each link in a multicast path between a source and destination(s).

When obtaining forwarding table entries from RSRR, RSVP also needs to know whether the multicast routing protocol is using sender-rooted (e.g shortest-path) trees or shared trees (e.g. with the PIM [3] or CBT [1] multicast routing protocols). For shared trees, RSVP only needs to obtain and store one route for all senders to the group. In addition, RSVP must mimic the IP multicast forwarding model. This means that for bidirectional shared trees, a packet can be accepted over any interface and is forwarded over all other interfaces listed in the route.

When running over shared trees, RSVP can also significantly decrease the size of the *scope* object in wildcard filter *Resv* messages. RSVP uses a *scope* object, listing all upstream senders, to prevent looping of wildcard filter *Resv* messages [4] (Figure 2a). For shared trees, RSVP can use a *scope* object that includes a wildcard address, greatly reducing the size of the *Resv* message. A *Resv* message with a wildcard address would follow shared tree state but never sender tree state (Figure 2b).

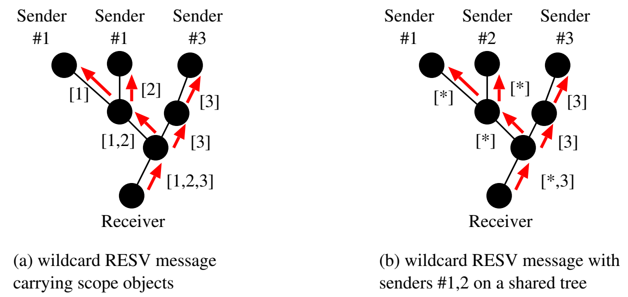

Figure 2: The *Scope* Object in Wildcard Reservations

3 Abstract Service Interface

We describe RSRR as an abstract service interface because it may be realized in several different ways. For example, some implementations may use system calls if the operating system can supply the functionality of RSRR. In a typical implementation, RSVP might use the RSRR abstraction as the basis of an interface to a routing support module (see Section 4), in addition to using the RSRR protocol (see Appendix A) to communicate with multicast routing protocol daemons.

To achieve this generality, the following interface description uses the term *RSRR client* to indicate the entity requesting routes and the term *RSRR server* to indicate the responding entity. The client is typically the RSVP daemon and the server is typically a routing protocol daemon or the operating system.

The first thing an RSRR client must do is determine the set of network interfaces the RSRR server is using. This allows the client to make sense of the routes it acquires from the server. RSRR represents network interfaces – which may be physical interfaces, tunnels, or any other operating system mechanism – as *virtual interfaces* or *vifs*. A vif has the following attributes:

id	a unique identifier,
threshold	a TTL threshold,
prefix	a CIDR prefix,
status	a bitmask status vector,
family	the address family,
length	the address length, and
local_addr	a local address.

This information represents what the RSRR client (e.g. RSVP) needs to simulate IP multicast forwarding. IP multicast uses the *TTL threshold* to control the scope of packets, checking it on a per-packet basis. A newer form of administrative scoping is enforced on a per-route basis, so it is reflected in the routes themselves and is thus transparent to the RSRR client. The server may also define a *status* vector for any other information it needs to pass to the client.

The RSRR client obtains the set of network interfaces by issuing an Interface Query of the form:

$$Interface\_Query().$$

The server responds with an Interface Reply that includes the number of vifs and a list of the vifs with their attributes as defined above:

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

The RSRR server automatically notifies the client if the set of interfaces or their status changes. Notification is in the form of a spontaneous Interface Reply.

Once the RSRR client has received the Initial Reply, it can begin requesting routes by sending a Route Query for a source-destination pair:

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

The RSRR server responds by sending a Route Reply:

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

Multicast routes consist of an *incoming vif* and an *outgoing vif bitmask*. Unicast routes consist of a single bit set in the *outgoing vif bitmask* to indicate the next hop; the *incoming vif* is always zero and should be ignored by the client. In addition, for unicast queries and replies the *source* address is zero.

By setting the *notification* flag in the Route Query, the RSRR client may ask the server to notify it when a route changes. A multicast route change is a change in the *incoming vif* or *outgoing vif bitmask*, i.e. joins, prunes, link failures and link recoveries are all included. A unicast route change is a change in the *outgoing vif bitmask*. If the server is able to provide this service, it sets a corresponding *notification* flag in the Route Reply, otherwise it clears the flag. If the client receives a Route Reply with the *notification* flag set, it can assume that routing will notify it – via a spontaneous Route Reply – when the route for the source-destination pair changes. In the meantime, the RSRR client can assume the route has not changed. If the RSRR server can not support route change notification, then the client must poll routing for forwarding table entries in order to learn of route changes.

The *tree type* flag in the Route Reply is used only for multicast routes. It has one of the following values:

- sender
- unidirectional shared

- hybrid unidirectional shared
- bidirectional shared
- hybrid bidirectional shared

The tree type indicates how forwarding occurs on the tree and thus how to interpret the route contained in the Route Reply. The *sender* value indicates that a separate multicast tree is constructed for each sender. Typically this is a shortest-path tree, but it may also be built using QoS routing, alternate paths, or other methods. In this case, packets must arrive via the *incoming vif* and are then sent on all of the vifs listed in the *outgoing vif bitmask*. A *unidirectional shared* tree is constructed for the entire group, but data flows in only one direction – from the root of the tree down to all group members. The forwarding model for the *unidirectional shared* tree is the same as for a sender tree; the difference is that senders transmit data to the root of the tree along separate branches. At routers where these branches are located, a Route Reply with the *sender* tree type will be returned. A *bidirectional shared* tree allows data to be sent by any group member. An incoming packet may arrive on any interface, and it is sent on all remaining vifs listed in the *outgoing vif bitmask*. For this tree type, the *incoming vif* is set to zero and should be ignored.

Hybrid multicast trees are those where some senders use a shared tree and others use a sender tree. If the tree type is set to one of the hybrid types, this indicates that the sender listed in the Route Reply is using the shared tree given by the route. Senders that are using a sender tree will have a *sender* tree type. Note that for hybrid trees the RSRR client must issue queries for each sender in case that sender is using a separate tree. This is in contrast to the case where all senders use a shared tree, in which case the RSRR client does not need to issue separate Route Queries for each of them.

Hybrid trees have the further consequence that senders can change tree types over time. Changes of this sort are handled by route change notification. For example, if a sender on a hybrid shared tree later switches to a sender-based tree, the RSRR server will send an updated Route Reply with a new route and a new tree type. Likewise, if all senders are using a shared tree, but later one switches to a shortest-path tree, the RSRR server will send two Route Replies – one for the shared tree that changes the tree type to hybrid, and one for the sender that is now using a sender tree.

4 Implementing RSRR within RSVP

A typical RSVP implementation could use the RSRR abstraction to build a routing support module that handles interface configuration and routing queries for IPv4 and IPv6 unicast and multicast routing information. Figure 3 shows how the routing support module would interact with RSVP and other router components. The module has three interfaces, all of which are modeled on the RSRR abstract service interface. First, RSVP's core

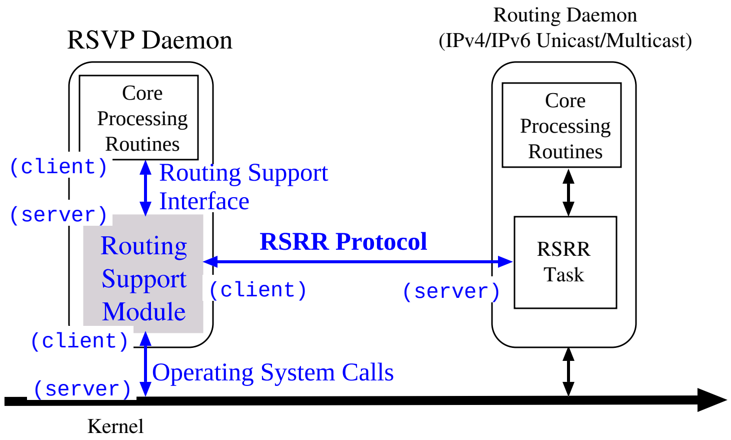

Figure 3: RSVP Routing Support Module

processing routines request interface configuration and forwarding table entries from the routing support module. The module then collects this information from several sources. It uses operating system calls when the kernel is able to provide the required information. Otherwise, it uses the RSRR Protocol to collect this information from a routing protocol daemon. Appendix A specifies the RSRR protocol for interprocess communication. For example, ISI's current implementation gets unicast interfaces and routes from the kernel, while it gets multicast interfaces and routes using the RSRR protocol.

5 Conclusion

Using RSRR version 2 and the routing support module has made it easier to upgrade RSVP to include IPv6 support by packaging interface configuration and route acquisition for IPv4 and IPv6 unicast and multicast routing into one module.

The impact of RSRR on a routing daemon is low. The daemon only has to handle RSRR protocol queries when an RSVP session is created and when a route for an RSVP session changes. The daemon incurs very little cost for providing route change notification; essentially it only has to tag the subset of its routes for which RSVP is interested in receiving notification. This amounts to keeping an extra bit for each routing entry. Furthermore, the RSRR services can be provided independently by each router, so its implementation is not subject to any interoperability constraints.

6 Acknowledgments

We would like to thank Bob Braden, Deborah Estrin, Bill Fenner, Scott Shenker, and Lixia Zhang for their help with 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, March 1997.

[3] 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.

[4] Daniel Zappala. "RSVP Loop Prevention for Wildcard Reservations". Work in Progress, February 1996.

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

Appendices

A RSRR Protocol Specification

This section details the RSRR version 2 messages format to be exchanged between RSVP and a routing protocol.

RSVP would like to use version 2 of RSRR to communicate with routing protocols, but fall back to version 1 if routing does not support version 2. One way to do this would be to issue an Interface Query for version 2 and wait to see if routing responds. If routing is using version 1 this message will be dropped and, after a timeout, RSVP could send a version 1 query.

However, to avoid this delay, we have included a mechanism in RSRR for backwards compatibility. RSVP issues an Interface Query using version 1, but sets the Num field to the maximum version of RSRR that RSVP supports. The routing protocol, if it supports only version 1, will ignore this field and respond with an Interface Reply for version 1. A routing protocol supporting version 2 and higher should examine this field and respond using a version equal to the minimum of Num and the maximum supported version of RSRR. This mechanism allows RSVP and routing to communicate using the maximum version of RSRR that they both support. Note that routing protocols implementing version 2 or higher should also be able to support queries for lower versions of RSRR. For reference, the version 1 RSRR specification is listed in Appendix A.

A.1 RSRR message format

```
An RSRR message consists of:

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

Version
        Routing Support for Resource Reservations Version.  This
        document specifies version 2 of RSRR.

Type
        This document defines four message codes for RSRR:

                1 = Interface Query
```

```
        2 = Interface Reply
        3 = Route Query
        4 = Route Reply

The rest of the message is defined separately for each RSRR code.
```

A.2 Interface Query

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

Version
        1   For compatibility reasons.

Type
        1 = Interface Query

Flags
        A bit vector that specifies what RSVP requests, currently
        only Notification bit is defined.

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

        N = 1 if RSVP requests notification of interfaces changes

Num
        Maximum supported RSRR version
```

A.3 Interface Reply

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Version       | Type          | Flags         | Num           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vif ID-1      |Vif Threshold-1| Prefix        | Vif Status-1  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Address Family                | Address Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vif Local Address-1                                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
|...                                                            |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vif ID-N      |Vif Threshold-N| Prefix        | Vif Status-N  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Address Family                | Address Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vif Local Address-N                                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version
        2

Type
        2 = Interface Reply

Flags
        1 = Error occurred during initial query, 0 otherwise

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

Prefix
        The CIDR prefix for the address

Vif Status-N
        A bit vector representing the vif status:

```
+-+-+-+-+-+-+-+-+
|       |N|P|U|M|
+-+-+-+-+-+-+-+-+
```

N = 1 if notification will be made in case of vif changes.
P = 1 if vif is physical interface, 0 if it is virtual.
U = 1 if vif is unicast-disabled, 0 if it is enabled.
M = 1 if vif is multicast-disabled, 0 if it is enabled.

The rest of the field is transmitted as zeroes.

Address Family
        The address family of the following address.

Address Length
        The length of the following address in octets.

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

A.4 Route Query

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

Version
        2

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

Address Family
        The address family of the following address.

Address Length
        The length of the following address in octets.

Destination Address
        Destination address being queried (unicast or multicast)

Source Address
        Source address being queried (null for unicast)

Query Identifier
        Identifier used by reservation protocol

A.5 Route Reply

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

```
|                                                               |
+                                                               +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
Version
        2

Type
        4 = Route Reply

Flags
        The currently defined flags are:

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

        N is set if N is set in the corresponding route query and the
        router 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.

Address Family
        The address family of the following address.

Address Length
        The length of the following address in octets.

Destination Address
        destination address of query = destination address of reply

Source Address
        source address of query = source address of reply (null if unicast)

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

Tree Type
        the currently defined values are:

        0       sender tree
        1       unidirectional shared tree
        2       hybrid unidirectional shared tree
        3       bidirectional shared tree
        4       hybrid bidirectional shared tree

Reserved
        transmitted as 0

Outgoing Vif Bitmask
        bitmask of outgoing Vifs for (S,G) or default (S,*) if no
        group-specific state; invalid if E bit is set; RSVP
should handle entire bitmask size or otherwise warn
user that some routing information may be lost
```

B RSRR V1 Protocol Specification

B.1 RSRR V1 Message Format

```
An RSRR v1 message consists of:

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

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

Type
        This appendix defines four message codes for RSRR:

                1 = Initial Query
                2 = Initial Reply
```

```
                3 = Route Query
                4 = Route Reply
```

The rest of the message is defined separately for each RSRR code.

B.2 Initial Query

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

Version as defined above.

```
Type
        1 = Initial Query

Flags, Num
        0
```

B.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
```

B.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
```

B.5 Route Reply

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

Version as defined above.

```
Type
        4 = Route Reply

Flags
        The currently defined flags are:

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

        N is set if N is set in the corresponding route query and the
        router 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.

        S has the binary value 01 if the listed sender is using a shared
        tree, but some other senders for the same destination use sender
        trees.  S has the binary value 10 if all senders for the
        destination use shared trees.  Otherwise, S has the value 00.

        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

Reserved
        transmitted as 0

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

C Implementations

Code supporting RSRR that has been and is being developed includes:

- ISI's RSVP version 4.2 and Xerox PARC's distribution of DVMRP/mrouted version 3.8 support RSRR version 1, minus support for shared trees
- ISI plans to release a future version of RSVP with support for RSRR version 2,
- UO plans to add support for RSRR version 2 to Merit's GateD multicast version 5, specifically testing shared tree support with IPv4 PIM sparse mode. This code should be general enough to support other multicast protocols implemented within GateD.

Security Considerations

Security considerations are not discussed in this memo.

Authors' Addresses

Daniel Zappala
Department of Computer and Information Science
University of Oregon
Eugene, OR 97403
EMail: zappala@cs.uoregon.edu

Jeff Kann
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
EMail: kann@isi.edu