Internet Draft
Expires: April 1994
File: draft-braden-rsvp-00.ps

Lixia Zhang
Xerox PARC
Bob Braden
USC-ISI
Deborah Estrin
USC/USC-ISI
Shai Herzog
USC-ISI
Sugih Jamin
USC
October 1993

Resource ReSerVation Protocol (RSVP) –

Version 1 Functional Specification

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. Internet-Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet-Drafts as reference material or to cite them other than as a "working draft" or "work in progress."

Abstract

This memo describes version 1 of RSVP, a resource reservation setup protocol designed for an integrated services Internet. RSVP provides receiver-initiated setup of resource reservations for multicast or unicast data flows, with good scaling and robustness properties.

1 Introduction

This memo describes RSVP, a resource reservation setup protocol designed for an integrated services Internet [RSVP93,ISInt93]. A host invokes RSVP to request a specific quality of service (*QoS*) for a data stream. Hosts and routers use RSVP to deliver these requests to the routers along the path(s) of the data stream and to maintain router and host state to provide the requested service. This generally requires reserving resources in those nodes.

At each router along the path, RSVP passes a new resource reservation request to an *admission control* routine, to determine whether there are sufficient resources available. If there are, the router reserve the resources and updates its *packet scheduler* and *classifier* control parameters to provide the requested QoS [ISInt93].

The objectives and general justification for RSVP design are presented in [RSVP93,ISInt93]. In summary, RSVP has the following attributes:

- RSVP supports multicast or unicast data delivery and adapts to changing group membership as well as changing routes.
- RSVP is not itself a routing protocol, but it is designed to use the existing unicast and multicast routing protocols to determine the data path(s).
- RSVP reserves resources for simplex data streams.
- RSVP is receiver-oriented, i.e., the receiver of the data flow is responsible for the initiation and maintenance of the resource reservation.
- RSVP maintains "soft-state" in the routers, enabling it to gracefully support dynamic membership changes and automatically adapt to routing changes.
- RSVP provides several "reservation styles" with different reservation models to fit a variety of applications.
- RSVP provides transparent operation through routers that do not support it.

There are two aspects to RSVP, its reservation model and its protocol mechanisms. The RSVP protocol mechanisms provide a general facility for creating and maintaining distributed reservation state across a mesh of multicast delivery paths. These mechanisms treat the reservation parameters as opaque data, except for certain well-defined operations, and simply pass them to the traffic control modules (admission control, packet scheduler, and classifier) for interpretation. Although the RSVP protocol mechanisms are independent of the encoding of these parameters, the encodings must be defined in the reservation model that is presented to an application (see section 3.6.1).

In order to efficiently accommodate heterogeneous receivers and dynamic group membership, RSVP makes the receivers responsible for requesting resource reservations [RSVP93]. Each receiver can request a reservation that is tailored to its particular requirement, and RSVP will deliver this request to the routers along the reverse path(s) to the sender(s).

Sections 2.1 and 2.2 of this memo summarize the RSVP reservation model, while Sections 2.3 describes the protocol mechanisms. Sections 2.4 presents the host model, and Section 2.5 gives examples of both model and mechanism. Section 3 presents the functional specification for RSVP.

2 RSVP Overview

2.1 RSVP Reservation Model

Figure 1 illustrates a single multicast distribution *session*. The arrows indicate data flowing from senders S1 and S2 to receivers R1, R2, and R3, and the cloud represents the distribution mesh created by the multicast routing protocol. Multicast distribution replicates each data packet from a sender Si and delivers a copy to every receiver Rj (whether a packet actually arrives at Rj depends on the specified QoS and perhaps upon congestion encountered along the path). Each sender Si and receiver Rj may correspond to a unique Internet host, or there may be multiple senders (e.g., multiple TV cameras) and/or receivers in a single host.

The RSVP model for data distribution is *simplex*, i.e., it reserves resources in only one direction on a link, so that senders are logically distinct from receivers. However, the same host may act as both sender and receiver.

```
     Senders                                Receivers
                    _____________________
                   (                     ) ===> R1
        S1 ===>    (    Multicast        )
                   (                     ) ===> R2
                   (    distribution     )
        S2 ===>    (                     )
                   (    by Internet      ) ===> R3
                   (_____________________)
```

Figure 1: Multicast Distribution Session

All data packets in a session are addressed to the same IP destination address DestAddress. For multicast delivery, DestAddress is the multicast group address to which the data is addressed, and the receivers will have all "joined" this multicast group. For unicast delivery, DestAddress is simply the unicast address of the single receiver. RSVP identifies a session by DestAddress plus a 32-bit stream identifier called the *destination stream id* (DestSID). We use the term *session socket* for the (DestAddress, DestSID) pair that defines a session. RSVP treats each session independently. In the rest of this document, a particular session (hence, session socket) is always implied even if not stated.

If a particular sender's flow arriving at a router has no corresponding reservation in place, the router will either drop the packets from the flow or send them using a best-effort QoS. This choice is controlled by each sender.

Depending upon the reservation style and the state already in place for the session, a new or modified reservation request may result in a call to admission control. If an admission control call fails, the reservation is rejected and an RSVP error message is sent downstream to the receiver(s) responsible for it.

A single RSVP resource reservation request is defined by a *flowspec* together with a *filterspec*; this pair is called a *Flow Descriptor*. The flowspec specifies the desired QoS in a quantitative manner, e.g., the allowable delay, the average bandwidth, the maximum burstiness, etc [ISInt93,IServ93]. It is used to parametrize the packet scheduling mechanism in the router or host. The filterspec defines the set of data packets to receive this service.

A flowspec is opaque to RSVP. However, given a pair of flowspecs Fls1 and Fls2, RSVP needs to be able to: (1) determine whether they are identical, (2) determine whether Fls1 “ >″ Fls2, or that they cannot be compared, and (3) find a third flowspec Fls3 that dominates both (Fls3 “ >″ Fls1 and Fls3 “ >″ Fls2), even if Fls1 and Fls2 cannot be compared. A flowspec may be a complex multi-dimensional vector; the relationship Fls1 “ >″ Fls2, when it is defined, indicates that Fls1 represents at least as strict a request (and hence represents at least as large a resource commitment) as Fls2.

The data packets selected by a particular filterspec will presumably be all be addressed to DestAddress. The filterspec may also select data packets only from particular senders Si; the set of senders selected by a particular filter is referred to as the *scope*. A filter may further reduce the data packet subset based on flow demultiplexing fields such as a UDP port, or perhaps on some hierarchical encoding bits within the application layer.

2.2 Reservation Styles

RSVP has a number of distinct *reservation styles*, which determine the precise reservation model for applications. The following reservation styles have been defined so far; others may be introduced in the future.

1. Wildcard-Filter

   Using the Wildcard-Filter (WF) style, a receiver creates a single reservation, or resource "pipe", along each link, shared among all senders for the given session. The size of this "pipe" is the maximum of the resource requests for that link from all receivers, independent of the number of senders using it.

   The term 'wildcard' refers to a filter scope that implicitly selects all senders, inplicitly extending to new senders as soon as they start sending to the group.

2. Fixed-Filter

   Using the Fixed-Filter (FF) style, each receiver selects the particular sender whose data packets it wants to receive, and it specifies a corresponding flowspec for that sender.

Receivers that select the same sender will share the reservation for that sender (indeed, due to multicasting there is only one stream of data packets from any Si in a particular router, regardless of the number of receivers Rj downstream). The shared reservation for Si will be greater than (in the sense of “$>$” as discussed earlier) or equal to all of the individual flowspecs from receivers Rj that selected Si.

However, reservations for different senders are distinct; they do NOT share a common pipe. The total reservation on a link for a given session is the sum over the reservations for different senders.

A Fixed-Filter reservation request from a particular receiver Rj generally contains a list of Flow Descriptors, each consisting of a filterspec (specifying some sender Si) and a corresponding flowspec. If a receiver using the FF reservation style changes its sender selection during the session, this is treated as a new reservation that is subject to admission control and may fail. The receiver may also modify the flowspecs, again subject to admission control.

3. Dynamic-Filter

   Using the Dynamic-Filter (DF) style, each receiver creates N distinct reservations to carry flows from up to N different senders. A DF style reservation also specifies a list of K filterspecs $(0 <= K <= N)$, defining particular senders to use these reservations, as well as a common flowspec. Like the FF style, the DF style causes distinct reservations for different senders.

   A later DF reservation from the same receiver may specify the same value of N and the same common flowspec but a different selection of particular senders, without a new admission control check. This is known as *channel switching*, in analogy with a television set. Each DF style reservation may be said to be “owned” by the receiver that established it and is permitted to switch channels (change senders) used by that reservation.

   If a receiver using the DF reservation style changes the number of distinct reservations N or the common flowspec, this is treated as a new reservation that is subject to admission control and may fail. Those data packets for the same session socket but from senders that are not currently selected may either be dropped or simply sent best-effort.

The essential difference between the FF and DF styles is that the latter allows dynamic channel switching without admission control. Once a DF-style reservation has been made, the receiver may switch channels without danger of an admission control failure due to limited resources (unless the route changes to a lower-capacity path or new senders appear).

In order to provide admission-free channel switching, the DF style must cause the routers must reserve the requested bandwidth for *all* the possible senders Si, even though some of this bandwidth may be unused at any one time, or always. DF reservations may therefore have a significant cost to the Internet in under-utilized reservations.

Wildcard-Filter reservations are appropriate when the total bandwidth required is independent of the number of senders. This is true for audio conferences, where a limited number of people talk at once. Thus, each receiver might issue a Wildcard-Filter reservation for twice one audio channel (to allow some over-speaking). On the other hand, the Fixed-Filter and Dynamic-Filter styles create independent reservations for the flows from different senders; this is required for video signals, whose 'silence' periods are typically uncoordinated among different senders.

2.3 RSVP Protocol Mechanisms

2.3.1 RSVP Messages

RSVP messages are sent as IP datagrams; thus, RSVP occupies the place of a transport protocol in the protocol stack. However, like ICMP and IGMP, RSVP is really an Internet control protocol; it does not carry any application data, and its messages are processed by the routers in the path.

Each receiver host sends RSVP reservation, or *Resv* messages into the Internet, carrying Flow Descriptors requesting the desired reservation; see Figure 2. These reservation messages must follow the reverse of the routes the data packets will use, all the way upstream to all the send that lie within the scope of active reservations. These *Resv* messages are finally delivered to the sender hosts, so that the senders can set up appropriate Traffic Control parameters for the first hop.

In order to route the *Resv* messages in the reverse direction from each receiver to all selected senders for a given session socket, each sender must transmit RSVP *Path* messages forward along the uni-/multicast routes provided by the routing protocol(s). These *Path* messages store *path state* in all the intermediate routers, effectively combining all the routing trees given by the routing protocol for the same DestAddress.

```
Sender                                             Receiver

                 ---------------------
   Path -->     (                     )
 Si =======>    (    Multicast        ) Path -->
   <-- Resv     (                     ) =========> Rj
                (    distribution     ) <-- Resv
                (_____________________)
```

Figure 2: RSVP Messages

The minimum information content of a *Path* message is a list of sender IP addresses, since this is required for routing *Resv* messages. However, *Path* messages may carry the following additional information:

- A template describing the format of data packets that the sender will originate.

  This template takes the form of two bitstrings forming a (value, mask) pair. Zero mask bits represent "don't care" (variable) bits in data packets. If present, this template is used by RSVP to validate the filters in a *Resv* message. Without such a template in the path state, there will be no feedback (except poor service) to the receiver that sets an impossible filter by mistake.

- A flowspec defining an upper bound on the traffic that will be generated.

  This flowspec can be used by RSVP to prevent over-reservation on the non-shared links starting at the sender [RSVP93].

A (template, flowspec) pair in a *Path* message is called a *Sender Descriptor*.

2.3.2 Soft State

To maintain reservation state, RSVP uses "soft state" in the router and host nodes. RSVP soft state is created and maintained in two directions by *Path* and *Resv* messages. It is periodically refreshed by messages that are identical to the message that established the state, and it is removed at each node by a timer-driven cleanup procedure if no refresh message is received within a *cleanup timeout* interval. If a route changes, the next copy of the message will initialize the state on the new route, while the state on the now-unused segment of the route will time out. Thus, whether a message is "new" or a "refresh" is determined separately for each message at each node, depending upon the state at that node. (This document will use the term "refresh message" in this effective sense, to indicate an RSVP message that does not modify the existing state at the node in question.)

RSVP sends its messages as IP datagrams, without reliable delivery. If a reservation request fails, an RSVP error message is returned to the receiver; however, RSVP sends no positive acknowledgment messages to indicate success. Periodic transmission of refresh messages by hosts and routers should replace any lost RSVP messages.

If the set of senders Si or receivers Rj changes, or if any of the reservation requests change, the RSVP state is adjusted accordingly. To modify a reservation, a receiver simply starts sending the new values; it is not necessary to "close" the old reservation first. RSVP believes the latest *Path* and *Resv* messages (ignoring the possibility of reordering).

When a *Resv* message is received at a router or sender host, the RSVP module checks whether the message is a new or modifedi reservation request or whether it simply refreshes an existing reservation. A new or modified request is passed to the admission control module for a decision. If the reservation is accepted, RSVP sets up (or modifies) the reservation and filter state. It also forwards the *Resv* message to the next reverse-hop router(s) or sender host(s), using the path state. If the request is rejected, RSVP discards the Resv message and returns a RSVP error message to the receiver host that originated it. If the modification replaces some previous state, RSVP may immediately remove the old state, or it may simply let the old state time out since it is no longer being refreshed.

```
 Previous      Incoming                      Outgoing
 Hops          Interfaces                    Interfaces


 ------            ---------------------
|      | Path -->  |                     |    Path -->
|      |-----------|                     |---------------
|______|  <-- Resv |                     |     <-- Resv
        data -->   |                     |    data -->
                   |        Router       |
 ------            |                     |
|      | Path -->  |                     |    Path -->
|      |-----------|                     |---------------
|______|  <-- Resv |                     |     <-- Resv
        data -->   |_____________________|    data -->
```

Figure 3: Router Using RSVP

Figure 3 illustrates the model of a router used by RSVP. The data arrives from a *previous hop* through a corresponding *incoming interface* and departs through an *outgoing interface*. The incoming interfaces shown in Figure 3 may be physical interfaces (e.g., to point-to-point links), or they may be logical interfaces, multiple paths using the same physical interface to a shared medium (e.g., an Ethernet). Output is designated by the outgoing interface rather than by a "next hop" address to be consistent with IP multicast routing. Since the same host may be both sender and receiver for a given session, the same physical interface may act in both the incoming and outgoing roles.

2.3.3 Merging RSVP Messages

To control its protocol overhead, RSVP supports *merging* of multiple *Path* and *Resv* messages for the same session socket. Those messages that cause a state change are forwarded without delay, while the rest may be "merged" into fewer messages, perhaps only one. Merging requires synchronization among the messages being merged. This is accomplished by saving the state of received messages, dropping the messages, and periodically generating and forwarding cumulative messages in their place. Thus, refresh messages are created hop-by-hop, at a rate determined by the *refresh period*.

Messages that modify the state in a node ("new" messages) must be forwarded without delay. Thus, the refresh period does not affect the rate at which new state propagates from end to end.

For *Path* messages, merging implies collecting together the Sender Descriptors from multiple incoming messages into a single outgoing *Path* message. For *Resv* messages, merging generally implies that only the essential (e.g., the largest) refresh reservation messages need be forwarded once per refresh period; redundant messages can be dropped. A successful reservation request will propagate as far as the closest point along the sink tree to the sender(s) where a reservation level equal or greater than that being requested has been made. At that point, the merging process will drop it in favor of a larger reservation.

As a result of merging, the number of RSVP control messages will increase less than linearly with the number of senders and receivers in a session. There is no merging across sessions, however, so the number of RSVP messages will increase linearly with the number of sessions.

The refresh period and the cleanup timeout must obey some general principles.

A. The cleanup timeout interval should be long enough to avoid reservation-flapping due to route-flapping.

If route flapping does occur, persistent state will allow duplicate reservations to be created along the alternate paths. This duplication is desirable to prevent interruptions of service quality due to route flapping.

B. The refresh period should be short enough in order to adapt quickly to route changes.

C. The refresh period must be long enough to control RSVP overhead.

Applications may differ in their sensitivity to outages, and should be able to have some control over the refresh period for their session state.

2.4 Host Model

Before a session can be created, the session socket (comprised of DestAddress and DestSID) must be assigned and communicated to all the senders and receivers by some out-of-band mechanism. In order to join an RSVP session, the end systems perform the following actions.

H1 A receiver joins the multicast group specified by DestAddress.

H2 A potential sender starts sending RSVP *Path* messages to the DestAddress.

H3 A receiver listens for *Path* messages.

H4 A receiver starts sending appropriate *Resv* messages, specifying the desired Flow Descriptors.

There are several synchronization issues.

- Suppose that a new sender starts sending data but there are no receivers. There will be no multicast routes beyond the host (or beyond the first RSVP-capable router) along the path;

the data will be dropped at the first hop until receivers(s) do appear (assuming a multicast routing protocol that "prunes off" or otherwise avoids unnecessary paths).

- Suppose that a new sender starts sending *Path* messages (H2) and immediately starts sending data, and there are receivers but no *Resv* messages have reached the sender yet (e.g., because its *Path* messages have not yet propagated to the receiver(s)). Then the initial data may arrive at receivers without the desired QoS.
- If the receiver starts sending *Resv* messages (H4) before any *Path* messages have reached it, RSVP will return *Err* messages.

  The receiver may simply choose to ignore such error messages, or it may avoid them in one of two ways. (1) It may synchronize in a higher-level protocol, e.g., a conference control protocol might ensures that *Resv* messages are not sent by any participant until all have started to send *Path* messages. (2) The receiver may synchronize using RSVP messages, by waiting for *Path* messages before sending *Resv* messages.

The interface (API) between RSVP and an application is not defined in this protocol spec, as it may be host-system dependent. However, Section 3.6.2 discusses the general requirements and presents a generic API.

2.5 Examples

Figure 4 shows schematically a router with two previous hops (labeled a and b) and two outgoing interfaces (labeled c and d). There are three upstream senders; packets from sender S1 (S2 and S3) arrive through previous hop a (b, respectively). There are also three downstream receivers; packets bound for R1 and R2 (R3) are routed via outgoing interface c (d, respectively).

```
                    ----------------
                  a |              | c
( S1 ) ---------->|              |----------> ( R1, R2)
                    |    Router    |
                  b |              | d
( S2,S3 ) ------->|              |----------> ( R3 )
                    |________________|
```

Figure 4: Router Configuration

We use the following notation for a *Resv* message sent by receiver Rj:

1. Wildcard-Filter

   WF(Rj; *{r})

   Here "*{r}" represents a Flow Descriptor with a "wildcard" filter (choosing all senders) and a flowspec of quantity r. For simplicity we imagine here that flowspecs are one-dimensional, defining for example the average bandwidth, and state them as a multiple of some unspecified base resource quantity B.

2. Fixed-Filter

   FF(Rj; S1{r1}, S2{r2}, ...)

   This message carries a list of sender, flowspec pairs, i.e., Flow Descriptors.

3. Dynamic-Filter

   DF(nB, Rj;) or DF(nB, Rj; S1, ...)

   This message carries the number n of channels to be reserved, each channel having bandwidth B. It also has an list, perhaps empty, of senders. Since each channel must carry the same bandwidth B, we omit flowspecs after the semicolon.

Figure 5 shows Wildcard-Filter reservations. The "Receive" column shows the *Resv* messages received over outgoing interfaces c and d, and the "Reserve" column shows the resulting reservation state for each interface. The "Send" column shows the *Resv* messages forwarded to previous hops a and b. In the "Reserve" column, each box represents one reservation "channel", with the corresponding filter.

As a result of merging, only the message with the largest flowspec is forwarded upstream to each previous hop. When the flowspecs are equal, the message whose receiver IP address is numerically larger is sent. Here we assume that IPaddr(R1) > IPaddr(R3) so the R1 reservation is sent upstream. Merging will therefore delete the messages enclosed in square brackets. In the outgoing interface c, there is a common reserved channel shared by R1 and R2.

Figure 6 shows Fixed-Filter style reservations. Merging takes place among the flow descriptors (i.e., filter spec, flowspec pairs); the unmerged messages are not shown. For example, the message forwarded to previous hop b, towards S2 and S3, contains flow descriptors received from outgoing interfaces c and d. Similarly, when FF(R2; S1{B}) and FF(R3; S1{3B}) are merged, the result is the single message FF(R3; S1{3B}) sent to previous hop a, towards S1.

For each outgoing interface, there is a private reservation for each source that has been requested, but this private reservation is shared among the receivers that made the request.

```
                        |
          Send          |      Reserve             Receive
                        |
                        |       --------
  WF(R1; *{3B}) <- (a)  |  (c) | *{3B}  |     (c) <- WF(R1; *{3B})
                        |      |________|
 [ WF(R2; *{B}) <- (a)] |                   [ (c) <- WF(R2; *{B}) ]
                        |
 [ WF(R3; *{B}) <- (a)] |
------------------------|----------------------------------------
                        |       -------
  WF(R1; *{3B}) <- (b)  |  (d) | *{B}  |     (d) <- WF(R3; *{B})
                        |      |_______|
 [ WF(R2; *{B}) <- (b)] |
                        |
 [ WF(R3; *{B}) <- (b)] |
```

Figure 5: Wildcard-Filter Reservation Example

```
                        |
          Send          |       Reserve             Receive
                        |
                        |       --------
  FF(R3; S1{3B}) <- (a) |  (c) | S1{B}  |   (c) <- FF(R2; S1{B}, S2{5B})
                        |      |________|
                        |      | S2{5B} |
                        |      |________|
------------------------|-------------------------------------------
                        |       --------
               <- (b)   |  (d) | S1{3B} |   (d) <- FF(R3; S1{3B}, S3{B})
  FF(R3; S2{5B}, S3{B}) |      |________|
                        |      | S3{B}  |
                        |      |________|
```

Figure 6: Fixed-Filter Reservation Example

Figure 7 shows an example of Dynamic-Filter reservations. Within the reservation boxes, the receiver that owns the reservation is shown followed by a colon. R2 and R3 have made reservations for which there is currently no filter, as indicated by a filter of '?'.

```
                        |
         Send           |      Reserve             Receive
                        |
                        |       -----------
  DF(2, R1; S1) <- (a)  |  (c) | R1: S1{B} |   (c) <- DF(2, R1; S1, S2)
                        |      |___________|
    DF(2, R2; ) <- (a)  |      | R1: S2{B} |   (c) <- DF(2, R2; S2)
                        |      |[R2: S2{B}]|
  DF(2, R3; S1) <- (a)  |      |___________|
                        |      | R2:  ?{B} |
                        |      |___________|
------------------------|-------------------------------------------
                        |       -----------
  DF(2, R1; S2) <- (b)  |  (d) | R3: S1{B} |    (d) <- DF(2, R3; S1)
                        |      |___________|
  DF(2, R2; S2) <- (b)  |      | R3:  ?{B} |
                        |      |___________|
    DF(2, R3; ) <- (b)  |
```

Figure 7: Dynamic-Filter Reservation Example

Both R1 and R2 have requested reservations for S2. Since there is only one stream of packets from S2, only R1's reservation is actually made. However, the router keeps a record of R2's request as a "hidden reservation", shown in square brackets. If R1 removes its reservation for S2, the node must check for a hidden reservation for the same source, so that it can reassign the channel to R2 without waiting for R2's next refresh. This use of hidden reservations for Dynamic-Filter reservations is necessary to avoid new admission control decisions (which might fail) and to ensure continuity of service.

A router should not reserve more Dynamic-Filter channels than the number of upstream sources. Thus, in Figure 7, both R1 and R2 request that admission control admit two channels worth of bandwidth. However, there are only 3 sources upstream from this point, so these requests are satisfied with a total of 3 channel reservations.

Since there is only one source upstream from previous hop b, the first parameter of the DF message (the count of channels to be reserved) could be decreased to 1 in the forwarded reservations. However, this is unnecessary because the routers upstream will reserve only one channel, regardless.

We note that Dynamic-Filter requests cannot be merged, since they tie each reservation to a particular owner. A node is permitted to include Flow Descriptors that are not relevant to the path (i.e., not upstream along that path). Thus, in Figure 7, the node could forward each of the three *Resv* messages exactly as received from c and d, to both previous hops a and b; the previous hop would ignore the irrelevant Flow Descriptors in each case.

As another example of these concepts, Figure 8 shows the previous hop router on incoming interface (a) in Figure 7. Here there are two hidden reservations.

```
       Send            |      Reserve          Receive
                       |       -----------
 DF(2, R1; S1) <- (a)  |  (c) | R1: S1{B} |  (c) <- DF(2, R1; S1)
                       |      |           |
   DF(2, R2; ) <- (a)  |      |[R2:  ?{B}]|  (c) <- DF(2, R2; )
                       |      |           |
 DF(2, R3; S1) <- (a)  |      |[R2: S1{B}]|  (c) <- DF(2, R3; S1)
                       |      |___________|
```

Figure 8: Previous Hop to Figure 7 interface (a).

3 Functional Specification

There are currently three types of RSVP messages: *Path, Resv*, and *Err*À fourth type, *Teardown* is under consideration.

3.1 Message Formats

3.1.1 Path Message

```
      0             1              2             3
+-------------+-------------+-------------+-------------+
| Vers | Type |    Flags    |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+  ---
|                      DestAddress                      | Session
+-------------+-------------+-------------+-------------+
|                        DestSID                        | Socket
+-------------+-------------+-------------+-------------+  ---
|                    Refresh Period                     |
+-------------+-------------+-------------+-------------+
|                    State TTL Time                     |
+-------------+-------------+-------------+-------------+
|                Previous Hop Address                   |
+-------------+-------------+-------------+-------------+
|       ///////////////     |          SD Count         |
+-------------+-------------+-------------+-------------+
|              Sender Descriptor List                   |
+-------------+-------------+-------------+-- ...
```

IP Fields:

Protocol

46

IP Source Address

The IP address of the host or router sending this message.

IP Destination Address

The IP address of the data destination (DestAddress).

RSVP Fields:

Vers

Version number. This is version 1.

Type

1 = Path Message

Flags

1 = No-Refresh

A *Path* message received with this flag bit on will be forwarded with no (further) merging, the path state TTL timer(s) will not be reset (as if no refresh message had been received), and a refresh message will not be generated during the current refresh period for the senders listed, However, if there is no path state a *Path* message containing this flag bit will be ignored.

For further discussion of the use of this bit, see Section 3.3 below.

8 = Drop

If this flag bit is on then data packets will be dropped when they are destined to this session but their sender is not currently selected by any filter. If this flag bit is off, such data packets will still be forwarded but without a reservation, i.e., using a best-effort class.

RSVP Checksum

A standard TCP/UDP checksum, over the contents of the RSVP message with the checksum field replaced by zero.

DestAddress, DestSID

The IP address and stream Id identifying the session, i.e., the session socket.

Previous Hop Address

The IP address of the interface through which the host or router last forwarded this message.

The Previous Hop Address is used to support reverse-path forwarding of *Resv* messages. This field is initialized by a sender to its IP address (see IP Source Address above) and must be updated at each router hop as the *Path* message is forwarded.

Refresh Period

This field specifies the refresh timeout period in milliseconds. See Section 3.3 below.

State TTL Time

This field specifies the time-to-live for soft state, in milliseconds. It determines the cleanup timeout period; see Section 3.3 below.

See Section 3.2 below.

SD Count

Count of Sender Descriptors that follow.

Sender Descriptor List

A list of Sender Descriptors (see below). The order of entries in this list is irrelevant.

Each sender must periodically send a *Path* message containing a single Sender Descriptor describing its own data stream. These messages are addressed to the uni-/multicast destination address for the session, and they are forwarded to all receivers, following the same paths as a data packet from the same sender. *Path* messages are received and processed locally to create path state at each intermediate router along the path.

If an error is encountered while processing a *Path* message, an RSVP *Err* message is sent to all the sender hosts listed in the Sender Descriptor List.

Path messages are distributed from senders to receivers along the exact paths that the data will traverse, using uni-/multicast routing. This distribution actually takes place hop-by-hop, allowing RSVP in each router along the path to observe and modify the message. Routing of *Path* messages is based on the sender address(es) from the Sender Descriptor(s), not the IP source address. This is necessary to prevent loops; see Section 3.2.

Each Sender Descriptor consists of a sender template that defines the format of data packets and a corresponding Flowspec that describes the traffic characteristics. The template is composed of an explicit IP address plus a variable-length mask-and-value pair VF, in the following format:

```
+-----------+-----------+-----------+-----------+
|                 Sender IP Address             |
+-----------+-----------+-----------+-----------+
|      VFlength         |      VFoffset         |
+-----------+-----------+-----------+-----------+  ---
|                 V: VF Value Part              | 4*VFlength
/                                               / octets
/                                               /
+-----------+-----------+-----------+-----------+  ---
|                 M: VF Mask Part               | 4*VFlength
/                                               / octets
/                                               /
+-----------+-----------+-----------+-----------+  ---
```

The value M and the mask V each have length 4∗VFlength octets. M and V define a filter using a simple mask-and-match algorithm applied to the packet at 4∗VFoffset octets from the beginning with the internet layer header. The VF part should not include the sender IP address or DestAddress; RSVP will effectively embed these explicit addresses into the template before using it to validate a filterspec.

3.1.2 Resv Message

Resv messages are sent from receivers to senders along reverse paths established by *Path* messages.

```
       0              1              2              3
+-------------+-------------+-------------+-------------+
| Vers | Type |    Flags    |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+  ---
|                      DestAddress                      | Session
+-------------+-------------+-------------+-------------+
|                        DestSID                        | Socket
+-------------+-------------+-------------+-------------+  ---
|                    Refresh Period                     |
+-------------+-------------+-------------+-------------+
|                    State TTL Time                     |
+-------------+-------------+-------------+-------------+
|                     RecvAddress                       |
+-------------+-------------+-------------+-------------+
| Dynamic Reservation Count |         FD Count          |
+-------------+-------------+-------------+-------------+
|                Flow Descriptor List                   |
+-------------+-------------+-------------+-- ...
```

The fields are the same as defined earlier for a *Path* message, except for the following:

IP Fields:

IP Source Address

The IP address of the host or router sending this message.

IP Destination Address

The IP address of the next-hop router or host to which this message is being sent.

RSVP Fields:

Type

2 = Resv Message

Flags

1 = No-Refresh

A *Resv* message received with this flag bit on will be forwarded without (further) merging, the TTL timer for the reservation state to which it refers will not be reset (as if no refresh message had been received), and no hop-by-hop refresh message will be generated during the current refresh period for the receiver specified by

RecvAddress. However, if there is no reservation state for this receiver or no path state for this session, a *Resv* message containing this flag bit will be ignored.

For further discussion of the use of this bit, see Section 3.3 section below.

The following flag bits indicate the reservation style.

2 = Fixed-Filter

4 = Dynamic-Filter

RecvAddress

The IP address of the receiver that originated this message.

Dynamic Reservation Count

The number of channels to be reserved, for a Dynamic-Filter style reservation.

If the ResvStyle is Dynamic-Filter, this integer value must be constant and equal or greater than (FD Count). For other ResvStyles, this field must be zero.

FD Count

Count of Filter Specs in the Flow Descriptor List.

Flow Descriptor List

A list of Flow Descriptors, i.e., (Filterspec, flowspec) pairs, to define individual reservations for Fixed-Filter and Dynamic-Filter styles. The first entry in the list may have special meaning (see below); the order of later entries is irrelevant.

Each Flow Descriptor has the following form:

```
+-------------+-------------+-------------+----------//-+
|  FiltSLen   |       Filter Spec ...                   |
+-------------+-------------+-------------+----------//-+
|  FlowSLen   |                 ... Flow Spec ...       |
+-------------+-------------+-------------+----------//-+
```

Here FiltSLen and FlowSLen are one-octet fields specifying the lengths in octets (including the length byte) of the Filterspec and flowspec, respectively.

A Wildcard-Filter style reservation is encoded as a special case of a Fixed-Filter reservation: a single Fixed-Filter channel in which the Filterspec specifies the wild-card filter, i.e., it selects packets from all senders Si to the given session socket.

The following specific rules hold for different reservation styles.

- Wildcard-Filter

  To obtain Wildcard-Filter service, set FD Count = 1 and include a single Flow Descriptor whose Filterspec part is a *wild card*, i.e., selects all senders. and whose flowspec part defines the desired flow parameters.

- Fixed-Filter

  Include a list of FD Count >= 1 Flow Descriptors, each defining a sender Filterspec and a corresponding flowspec.

- Dynamic-Filter

  Include max(1, FD Count) Flow Descriptors in the message. Here the FD Count specifies the number of sender Filterspecs that are included. If DC is the Dynamic Reservation Count, then DC >= FD Count >= 0.

  The Flowspec part of the first Flow Descriptor defines the desired size of all the DC channels that are reserved. The Flowspec parts of later Flow Descriptors (if any) are ignored.

3.1.3 Err Message

```
       0             1              2             3
+-------------+-------------+-------------+-------------+
| Vers | Type |    Flags    |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+  ---
|                      DestAddress                      | Session
+-------------+-------------+-------------+-------------+
|                        DestSID                        | Socket
+-------------+-------------+-------------+-------------+  ---
|  Error Code | Error Value |        List Count         |
+-------------+-------------+-------------+-------------+
|              Sender|Flow Descriptor List              |
+-------------+-------------+-------------+-- ...
```

The fields are the same as in a *Path* message, defined earlier, except for the following:

RSVP Fields:

RSVPType

4 = Err message

Flags

0xxxxxxx = Towards receiver(s)

1xxxxxxx = Towards sender(s)

Error Code

A one-octet error description.

01 = No path information for this Resv (R)

02 = No Sender information for this Resv (R)
There is path information, but it does not include the sender specified in one or more of the Filterspecs listed in the Resv messager.
03 = Insufficient memory (S,R)
04 = Incorrect Dynamic Reservation Count (R)
Dynamic Reservation Count is zero or less than FD Count.
05 = Reservation problem (R)
The Error Value octet an error code specific to a lower-level routine. Possible reasons include: not enough resource available, flowspec syntax error, flowspec value error (internal inconsistencies of values), or Flowspec feature not supported.
07 = Unknown RSVPType field.
08 = Unknown RSVP version.
09 = FD Count Wrong
FD Count does not match length of message.

Error Value

Specific cause of the error described by the Error Code. Meanings are generally defined outside RSVP.

Sender—Flow Descriptor List

Optional list of Sender Descriptors (towards sender) or Flow Descriptors (towards receiver) indicating which message triggered the error.

The message may include a list of one or more descriptors to which the same error applies. An acceptable implementation may send a single descriptor per *Err* message, and may therefore send multiple *Err* messages as the result of one *Path* or *Resv* message.

If an error is encountered while processing a *Resv* message, an RSVP *Err* message must be sent to all receivers responsible for the reservation. However, in the case of shared Fixed-Filter reservations, the node that detects the error does not have a record of all the responsible receivers; the merging process downstream will have dropped all but one of the receiver identities. Therefore, the *Err* message is addressed to the session DestAddress and uni-/multicast to all receivers downstream from the node detecting the error.

This may deliver the *Err* message to irrelevant receivers as well as all relevant ones. The API in the receiver hosts is therefore asked to filter such *Err* messages, delivering them only to those applications that have made a reservation for the sender specified in the message.

3.1.4 Tear Message

The *Teardown* message is intended to force state to be deleted immediately, without waiting for the cleanup timeout period. It is an optimization, to allow resources to be freed more quickly. However, like the other RSVP messages, it is not reliably delivered.

State can only be removed by *Teardown* if it is not shared. In particular, Wildcard-Filter and Fixed-Filter reservations may be shared among multiple receivers, and due to merging RSVP does not keep track of all the responsible receivers; in these cases, teardown is not possible.

A more complete definition of *Teardown* messages is future work.

3.2 Avoiding Message Loops

RSVP routes its control messages, and every routing procedure must avoid looping packets. The merging of RSVP messages delays forwarding at each node for up to one refresh period. This may avoid high-speed loop, but there can still be "slow" loops, clocked by the refresh period; the effect of such slow loops is to keep state active forever, even if the end nodes have ceased refreshing it. RSVP uses the following rules to prevent looping messages.

1. When an RSVP message is received on through particular incoming Interface F, the message must not be forwarded out F as an outgoing interface. This implies that RSVP must keep track of the interface through which each message is received, to avoid forwarding it out that interface. Note that, although RSVP distinguishes incoming from outgoing interfaces, in many cases the same physical interface will play both roles.

2. When a *Path* message is received, a route must be computed for each of its sender Flow Descriptors. These routes, obtained from the uni/multicast routing table, generally depend upon the (sender host address, destination address) pairs. Each route consists of a list of outgoing interfaces; these lists (with the incoming interfaces deleted by rule (1)) are used to create merged *Path* messages to be forwarded through the outgoing interfaces.

   Assuming that multicast routing is free of loops, *Path* messages cannot loop even in a topology with cycles.

   Since *Path* messages don't loop, they create path state defining a loop-free path to each sender. As a result, *Resv* messages directed to particular senders cannot loop. However, this cannot protect against looping *Resv* messages that are directed towards all senders ("wildcard" sender). The following three rules are needed for this purpose.

3. A *Resv* message whose RecvAddress matches one of the IP addresses of the local node must be discarded without processing.

```
                    ----------------
                 a |                | c
( R1, S1 ) <----->|     Router     |<-----> ( R2, S2 )
                   |________________|

    Send & Receive on (a)    |      Send & Receive on (c)
                             |
  WF(R2; *{3B}) <-- (a)      |   (c) <-- WF(R2; *{3B})
                             |
  WF(R1; *{4B}) --> (a)      |   (c) --> WF(R1; *{4B})
                             |
                             |
     Reserve on (a)          |        Reserve on (c)
        ----------           |       ----------
       |  * {4B}  |          |      |   * {3B} |
       |__________|          |      |__________|
                             |
```

Figure 9: Example: Rule (1) for Preventing Loops.

4. Each *Resv* message carries a receiver address. When the choice of address to place in a merged *Resv* message is otherwise arbitrary, RSVP must use the IP address that is numerically largest.
5. When a *Resv* message is received, the Reverse Path Forwarding rule is applied to the receiver address in the message: the message is discarded unless it arrived on the interface that is the preferred route to the receiver.

Figure 9 illustrates the effect of the rule (1) applied to *Resv* messages. It shows a transit router, with one sender and one receiver on each side; interfaces a and c therefore are both outgoing interfaces and physical previous hops. Both receivers are making a Wildcard-Filter style reservation, in which the *Resv* message is to be forwarded to all previous hops for senders in the group, with the exception of the interface through which it arrived.

3.3 Soft State Management

The RSVP state associated with a session in a particular node is divided into atomic elements that are created, refreshed, and timed out independently. The atomicity is determined by the requirement that any sender or receiver may enter or leave the session at any time, and its state should be created and timed out independently.

Management of RSVP state is complex because there is not generally a one-to-one correspondence between state carried in RSVP control messages and the resulting state in nodes. Due to merging, a single message contain state referring to multiple stored elements. Conversely, due to reservation sharing, a single stored state element may depend upon (typically, the maximum of) state values received in multiple control messages.

For each element, there are two time parameters controlling the maintenance of soft state: the refresh period R and the TTL (time-to-live) value T. R specifies the period between successive refresh messages over the same link. T controls how long state will be retained after refreshes stop appearing.

Path and *Resv* messages specify both R and T. When messages are merged and forwarded to the next hop, R should be the minimum R that has been received, and T should be the maximum T that has been received. Thus, the largest T determines how long state is retained, and the smallest R determines the responsiveness of RSVP to route changes. In the first hop, they are expected to be equal. The RSVP API should set a configurable default value, which can be overridden by an application for a particular session.

To avoid gaps in user service due to lost RSVP messages, RSVP should be forgiving about missing refresh messages. A node should not discard an RSVP state element until K * Tmax has elapsed without a refresh message, where Tmax is the maximum of the T values it has received. K is some small integer; K-1 successive messages may be lost before state is deleted. Currently K = 3 is suggested.

Let X indicate a particular message type (either "Path" or "Resv") and a particular session. Then each X message from node a to node b carries refresh period Rab and TTL time Tab.

- As X messages arrive at node b, the node computes and saves both the min over the Rab values (min(Rab)) and the max over the Tab values (max(Tab)) from these messages.
- The node uses K * max(Tab) as its cleanup timeout interval.
- The node uses min(Rab's) as the refresh period.
- Each refresh message forwarded by node b to node c has Tbc = max(Tab) and Rbc = min(Rab)
- A node may impose an upper bound Tmax and a lower bound Rmin, set by configuration information, and enforce: $Rmin <= R <= T <= Tmax$.

When a sender or receiver stops sending refreshes or a route change isolates a part of the path, the state in the nodes times out. However, we wish to avoid delaying the timeout of each hop by approximately one refresh period from the timeout of the preceding hop, i.e., avoid making the timeout time for the last node in the path increase linearly with the number of hops. This is avoided by the No-Refresh bit in *Path* and *Resv* messages.

- When a node has received no refresh message at the end of its current refresh period, it sends a (non-)refresh message with the No-Refresh bit on.

- A message with the No-Refresh bit is forwarded immediately hop-by-hop to the end of the path (unless it is lost).
- At each node, the receipt of a message with the No-Refresh bit on suppresses sending a refresh message at the end of the current refresh period.
- Receipt of a message with the No-Refresh bit on does not reset the T timer for the corresponding element(s).

In a perfect world these No-Refresh messages will cause all following nodes to time out at the same time. Due to small variations in timing, the actual behavior may be more complex, but all nodes should time out at approximately the same time.

The receiver should be conservative about reacting to certain error messages. For example, during a route change a receiver may get back "No Path" error messages until Path messages have propagated along the new route.

When *Resv* messages are merged, the message that is forwarded will carry the largest flowspec and the corresponding RecvAddress from the merged messages. If the same largest flowspec occurs in two or more merged messages, the resulting message should carry the numerically largest RecvAddress. This choice will ensure success of loop detection using the RecvAddress field.

3.4 Sending RSVP Messages

Under overload conditions, lost RSVP control messages could cause the loss of resource reservations. It recommended that routers be configured to give a preferred class of service to RSVP packets. RSVP should not use significant bandwidth, but its delay needs to be controlled.

An RSVP *Path* or *Resv* message consists of a small root segment (24 or 28 bytes) followed by a list of descriptors. The descriptors are bulky and there could be a large number of them, resulting in potentially very large messages. IP fragmentation is inadvisable, since it has bad error characteristics. Instead, RSVP-level fragmentation should be used. That is, a message with a long list of descriptors will be divided into segments that will fit into individual datagrams, each carrying the same root fields. Each of these messages will be processed at the receiving node, with a cumulative effect on the local state. No explicit reassembly is needed.

3.5 Automatic Tunneling

It is impractical to deploy RSVP (or any protocol) at the same moment throughout the Internet, and RSVP may never be deployed everywhere. RSVP must therefore provide correct protocol operation even when two RSVP-capable routers are joined by an arbitrary "cloud" of non-RSVP routers.

RSVP will automatically tunnel through such a non-RSVP cloud. Both RSVP and non-RSVP routers forward *Path* messages towards the destination address using their local uni-/multicast routing table. Therefore, the routing of Path messages will be unaffected by non-RSVP routers in the path. When a *Path* message traverses a non-RSVP cloud, the copies that emerge will carry as a Previous Hop address the IP address of the last RSVP-capable router before entering the cloud. This will cause effectively construct a tunnel through the cloud for *Resv* messages, which will be forwarded directly to the next RSVP-capable router on the path(s) back towards the source.

This automatic tunneling capability of RSVP has a cost: a *Path* message must carry the session DestAddress as its IP destination address; it cannot be addressed hop-by-hop. As a result, each RSVP router must have a small change in its multicast forwarding path to recognize RSVP messages (by the IP protocol number) and intercept them for local processing. See Section 3.6.5 below.

(There is a potential defect in tunneling. Merged *Path* messages can carry information for a list of senders, and since multicast routing depends in general upon the sender, it is not possible to ensure that all the non-RSVP routers along the tunnel will be able to route the packet properly. The effect turns out to be that tunnels may distribute path information to RSVP routers where it should not go, which may in turn lead to unused reservations at these routers. This is hoped to be an acceptable defect.)

Of course, if an intermediate cloud does not support RSVP, it is unable to perform resource reservation. In this case, firm end-to-end service guarantees cannot be made. However, if there is sufficient excess capacity through such a cloud, acceptable and useful realtime service will still be possible.

3.6 Interfaces

3.6.1 Reservation Parameters

The Flowspec format is currently specific to the CSZ packet scheduler [CSZ92]. The parameters are:

- QoS Type (Guaranteed, Predictive, ...)
- Max end-to-end delay
- Average data rate (bits/ms)
- Token bucket depth (bits)
- Global share id ¡— ?

Although a filterspec may be in part opaque, RSVP must be able to extract the sender IP address (or wildcard) from it, and to compare it with a sender template field in the path state. For compactness and simplicity of processing, this version of the RSVP specification defines an RSVP Filterspec to be composed of an explicit IP address plus a variable-length mask-and-value pair VF, in the following format:

```
+-----------+-----------+-----------+-----------+
|                Sender IP Address              |
+-----------+-----------+-----------+-----------+
|       VFlength        |       VFoffset        |
+-----------+-----------+-----------+-----------+  ---
|                V: VF Value Part               | 4*VFlength
/                                               / octets
/                                               /
+-----------+-----------+-----------+-----------+  ---
|                M: VF Mask Part                | 4*VFlength
/                                               / octets
/                                               /
+-----------+-----------+-----------+-----------+  ---
```

The value M and the mask V each have length 4∗VFlength octets. M and V define a filter using a simple mask-and-match algorithm applied to the packet at 4∗VFoffset octets from the beginning with the transport-layer header. VFlength can be zero, in which case $VFoffset$ is ignored. A "wildcard" Filterspec that will match any sender or receiver has the IP address and the VFlength both zero. The contents of M and V are opaque to RSVP. The VF part may or may not include the sender IP address or DestAddress; RSVP will embed these explicit addresses into the filterspec before handing it to traffic control. In general, RSVP cannot interpret the contents of the V and M fields, since they may depend upon protocols above the Internet layer.

There are many possible filters that cannot be expressed using a simple mask and value pair. A compact and general filter encoding is for further study.

3.6.2 Application/RSVP Interface

This section describes a generic API from an application to an RSVP control process. The details of a real interface may be operating-system dependent; the following can only suggest the basic functions to be performed. In particular, some of these calls cause information to be returned asynchronously.

An application could directly send and receive RSVP messages, just as an application can do file transfer using UDP. However, we envision that many applications will not want to know the details of RSVP operation, nor to provide the timing services necessary to keep the state refreshed, any more than an application wants to handle TCP retransmission timeouts. Therefore, a host using RSVP may be expected to have an RSVP control process to handle these functions. Using local IPC, applications will register or modify resource requests with this process and receive notifications of success or change of conditions.

1. Sender

   Call: SENDER(local socket, session socket, flowspec) $->$ sid

   This call is made by a sender host for each sender socket, to initiate RSVP processing. Wildcards may be inserted for the IP address and/or RSVP Id in the local socket, in which case the local system will choose appropriate values. The session socket consists of the uni-/multicast address of the receiving host(s) and the RSVP Id to use for this session.

   The SENDER call returns immediately with a local session identifier "sid", which may be used in subsequent calls. A local session control block is created and initialized, and the host begins sending periodic *Path* messages.

2. Receiver

   Call: RECVER(local socket) $->$ sid

   This call is made by a receiver host for each receiver socket, to initiate RSVP processing. The local socket consists of the uni-/multicast address for the desired session, and an RSVP Id.

   The RECVER call returns immediately with a local session identifier "sid", which may be used in subsequent calls. A local session control block is created and initialized, and the host begins listening for a *Path* message.

   Following this call, data may be returned asynchronously, containing the flowspecs and associated foreign sockets for sender hosts. Error message(s) may also be returned asynchronously.

3. Reserve

   Call: RESERVE(sid, style, Flowspec, Filterspec-list)

   A receiver uses this call to make a resource reservation. Style is an integer index indicating the reservation style.

   The first RESERVE call following a RECVER call will initiate the periodic transmission of *Resv* messages. A later RESV call may be given to modify the parameters of the earlier call; however, this may result in Admission Control failure.

   The RESERVE call returns immediately. Following this call, an error message or a data reply from the earlier RECVER call may be returned asynchronously.

4. Close

   Call: CLOSE(sid)

   This call may be made by either sender or receiver to terminate RSVP state for the given session id. It will delete local state and cease sending refreshes, allowing distributed state to time out.

5. Teardown

   Call: TEARDOWN(sid)

   This call sends *Teardown* messages to actively remove state from the routers, and then issues a CLOSE.

3.6.3 RSVP/Traffic Control Interface

In each router and host, enhanced QoS is achieved by a group of inter-related functions: a packet classifier, an admission control module, and a packet scheduler. We group these functions together under the heading *traffic control*. RSVP uses the interfaces in this section to invoke the traffic control functions.

((XXX Need some way to pass the Drop flag))

1. Make a Reservation

   Call: $Rhandle = TCAddFlow(Flowspec, Drop_flag, [Session\!-\!Filterspec[, Sender\!-\!Filterspec]])$

   Returns an internal handle Rhandle for subsequent references to this reservation.

   This call passes Flowspec to admission control and returns an error code if Flowspec is malformed or if the requested resources are unavailable. Otherwise, it establishes a new reservation channel corresponding to Rhandle, and if Filterspecs are supplied, installs a corresponding filter in the classifier.

   For Fixed-Filter reservation requests, RSVP knows about sharing and calls AddFlow only for distinct source pipes.

   For Dynamic-Filter reservation requests: suppose that the *Resv* message specifies a Dynamic Reservation Count = D, and F flow descriptors, where 0 ¡= F ¡= D. Then RSVP calls AddFlow D times, and D-F of those calls have null filterspecs.

2. Switch a Channel

   Call: $TCModFilter(Rhandle, [newFilterspec])$

   This call replaces the filter without calling admission control. It may replace an existing filter with no filter, modify an existing filter, or replace no filter by a filter.

3. Modify Flowspec

   Call: $TCModFlowspec(Rhandle, oldFlowspec, newFlowspec)$

   Here newFlowspec may be larger or smaller than oldFlowspec.

4. Delete Flow

   Call: TCDeleteFlow(Rhandle)

This call kills the reservation and reduces the reference count of, and deletes if the count is zero, any filter associated with this handle.

5. Initialize

   Call: TCInitialize( )

   This call is used when RSVP initializes its state, to clear out all existing classifier and/or packet scheduler state.

3.6.4 RSVP/Routing Interface

An RSVP implementation needs the following support from the packet forwarding and routing mechanism of the node.

- Promiscuous receive mode for RSVP messages

  Any datagram received for IP protocol 46 is to be diverted to the RSVP program for processing, without being forwarded.

- Route discovery

  RSVP must be able to discover the route(s) that the routing algorithm would have used for forwarding a specific datagram.

  $GetUCRoute(DestAddress) -> NextHop, Interface$

  $GetMCRoute(SrcAddress, DestAddress) -> Interface$

- Outgoing Link Specification

  RSVP must be able to force a (multicast) datagram to be sent on a specific outgoing virtual link, bypassing the normal routing mechanism. A virtual link may be a real outgoing link or a multicast tunnel.

  This is necessary because RSVP sends different copies of outgoing path messages on different links, even though all of them use the same source and destination addresses.

- Discover (Virtual) Interface List

  RSVP must be able to learn what virtual interfaces exist.

4 ACKNOWLEDGMENTS

Lixia Zhang, Scott Shenker, Deborah Estrin, Dave Clark, Sugih Jamin, Shai Herzog, Steve Deering, Bob Braden, and Daniel Zappala have all made contributions to the design of RSVP. We are grateful to Jamin and Herzog for prototype implementations. The original protocol concepts for RSVP arose out of discussions in meetings of the End-to-End Research Group.

References

[CSZ92] Clark, D., Shenker, S., and L. Zhang, *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms*, Proc. SIGCOMM '92, Baltimore, MD, August 1992.

[ISInt93] Braden, R., Clark, D., and S. Shenker, *Integrated Services in the Internet Architecture: an Overview*, working draft, October 1993.

[IServ93] Shenker, S., Clark, D., and L. Zhang, *A Service Model for an Integrated Services Internet*, working draft, October 1993.

[RSVP93] Zhang, L., Deering, S., Estrin, D., Shenker, S., and D. Zappala, *RSVP: A New Resource ReSerVation Protocol*, IEEE Network, September 1993.