Internet Draft
Expires: January 1995
File: draft-ietf-rsvp-spec-03.ps

L. Zhang
Xerox PARC
R. Braden
ISI
D. Estrin
ISI
S. Herzog
ISI
S. Jamin
USC

July 1994

Resource ReSerVation Protocol (RSVP) –

Version 1 Functional Specification

July 15, 1994

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 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.

What's Changed Since Seattle IETF

- Redesign generic RSVP API (section 3.6.2)
- Change encoding of style in *Resv* messages (section 3.1.2)
- Clarify filterspec functions (section 2.1)
- Simplify definition of DF style (sections 2.2, 2.4).
- Revise discussion of flowspec merging (section 2.3.3).
- Change format of variable-length filterspecs and flowspecs (section 3.1 and 3.6.1).
- Add a user authentication field in all RSVP messages (Section 3).
- Add short discussion of local repair (Section 3.3.3).
- Editorial nits.

1 Introduction

This memo describes RSVP, a resource reservation setup protocol designed for an integrated services Internet [RSVP93,ISInt93]. An application 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 *node* (i.e., router or host) 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 node reserves the resources and updates its *packet scheduler* and *classifier* control parameters to provide the requested QoS [ISInt93]. It is expected that RSVP implementations will execute in user space in a host, and in background in a router. On the other hand, the packet scheduler and classifier are expected to execute in the kernel of a host operating system, and in the high-speed packet forwarding path of a router.

RSVP messages are sent as IP datagrams; thus, RSVP occupies the place of a transport protocol in the protocol stack. However, like ICMP, IGMP, and routing protocols, 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.

RSVP is not itself a routing protocol, but rather it is designed to operate with existing and future unicast and multicast routing protocols. Thus, a host sends IGMP messages to join a multicast group, and then it sends RSVP messages to reserve resources along the deliver path(s) from that group. Unlike a routing protocol, RSVP is explicitly invoked by applications, to obtain a special QoS.

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 reserves resources for simplex data streams.
- RSVP is receiver-oriented, i.e., the receiver of a data flow is responsible for the initiation and maintenance of the resource reservation used for that flow.
- 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 models or "styles" (defined below) to fit a variety of applications.
- RSVP provides transparent operation through routers that do not support it.

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 largely 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).

There are two aspects to RSVP, its reservation model and its protocol mechanisms. 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 gives examples of both model and mechanism, and Section 2.5 summarizes the model of RSVP seen by a host. 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 forwards a copy of each data packet from a sender Si to every receiver Rj. Each sender Si and receiver Rj may correspond to a unique Internet host, or there may be multiple logical senders (e.g., multiple TV cameras) and/or receivers in a single host.

RSVP reserves resources for *simplex* data streams, i.e., it reserves resources in only one direction on a link, so that a sender is logically distinct from a receiver. However, the same application 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 given 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. 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 *reservation id* (ResvID). We use the term *session socket* for the (DestAddress, ResvID) 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.

Depending upon the reservation style and the session state already in place, a new or modified reservation request may or may not result in a call to admission control at each node [ISInt93]. If an admission control call fails, the reservation is rejected and an RSVP error message is sent 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 tolerable delay, the average throughput, the maximum burstiness, etc [Partridge92, ISInt93, IServ93]; it is used to set parameters to the packet scheduling mechanism in the node (router or host). The filterspec (plus the DestAddress) defines the set of data packets to receive this service; it is used to set parameters in the packet classifier component of the node. For all packets that are addressed to a particular session, only those that can match the filter spec(s) of that session will be forwarded according to the flowspec; the rest will be either dropped or sent as best-effort traffic.

More specifically, a filterspec may have two distinct functions.

- Sender Selection

  A filterspec may select packets that originate from a particular sender Si, from the entire stream of packets destined to a given DestAddress. The sender is selected using its IP source address and optionally a *generalized source port*, i.e., multiplexing field(s) at the transport layer (e.g., a UDP destination port) and/or the application layer (e.g., a particular subset of a hierarchically encoded video stream).

- Receiver Sub-selection

  A filterspec may distinguish different sessions with the same DestAddress by selecting a subset of the packets destined to that address. This subset is defined by a *generalized destination port*, which again may include transport-layer (e.g., UDP destination port) and/or application-layer demultiplexing information. An RSVP receiver Rj is defined by the pair (Hj, Pj), where Hj is the IP host address and Pj is the generalized destination port.

RSVP needs to distinguish different sessions. It is difficult to do this by matching generalized destination ports buried within the filterspecs, since the part of the filterspec that defines the generalized destination port should be opaque to an RSVP module in a router, which does not not know the structure of transport or application layer protocol headers. Therefore, RSVP identifies a session by the pair (DestAddress, ResvID), where the ResvID's form a simple space of identifiers that RSVP can use to distinguish different sessions with the same DestAddress. The ResvID's need not themselves be (generalized) ports, but the the ResvID values that are used must have a one-to-one correspondence with the generalized ports in use for the given DestAddress.

All reservation requests for a given session must use filterspecs that specify the same DestAddress and the same generalized destination port (since receivers of the same substream, downstream of a given node, must share a common resource reservation in that node).

2.2 Reservation Styles

In addition to the Flow Descriptors, each RSVP reservation request specifies a *reservation style*. The following reservation styles have been defined so far.

1. Wildcard-Filter (WF) Style

   A Wildcard-Filter (WF) style reservation creates a single resource "pipe" along each link, shared by data packets from all senders for the given session. The "size" of this pipe is the largest of the resource requests for that link from all receivers, independent of the number of senders using it. (The concept of a "largest" flowspec is discussed later).

The term *wildcard* implies a filterspec that selects all senders. A WF reservation automatically extends to new senders to the session, as they appear.

2. Fixed-Filter (FF) Style

   A Fixed-Filter (FF) style reservation request creates reservation(s) for data packets from particular sender(s). A FF reservation request from a particular receiver Rj contains a list of one or more Flow Descriptors, each consisting of a filterspec, which specifies some sender Si, and a corresponding flowspec.

   FF reservations requested by different receivers Rj but selecting the same sender Si must necessarily share a single reservation in a given node. This is simply the result of multicast distribution, which creates a single stream of data packets in a particular router from any Si, regardless of the number of receivers downstream. The reservation for Si will be the maximum of the individual flowspecs from different downstream receivers Rj (see Section 2.3.3).

   FF reservations for different senders are distinct; they do NOT share a common pipe. The total reservation on a link for a given session is therefore the cumulative total of the reservations for each requested sender. A receiver that has established a FF style reservation may modify, add, or delete a flow descriptor at any time. However, any additional or modified reservations are subject to admission control and may fail.

3. Dynamic-Filter (DF) Style

   A Dynamic-Filter (DF) style reservation decouples reservations from filters. Each DF reservation request specifies a number D of distinct reservations to be made using the same specified flowspec. The number of reservations that are actually made in a particular node is $D' = min(D, Ns)$, where Ns is the total number of senders upstream of the node.

   In addition to D and the flowspec, a DF style reservation may also specify a list of K filterspecs, for some K in the range: $0 <= K <= D'$. These filterspecs define particular senders to use the D' reservations. Once a DF reservation has been established, the receiver may change the set of filterspecs to specify a different selection of senders, without a new admission control check (assuming D' and the common flowspec remain unchanged). This is known as *channel switching*, in analogy with a television set.

   In order to provide assured channel switching, each node along the path must reserve enough bandwidth for *all* D' channels, even though some of this bandwidth may be unused at any one time. If D' changes (because the receiver changed D or because the number Ns of upstream sources changed), or if the common flowspec changes, the refresh message is treated as a new reservation that is subject to admission control and may fail.

   Like a FF style request, a DF style request causes distinct reservations for different senders.

   As noted earlier, those data packets from senders that are not currently selected may either be dropped or sent best-effort.

WF reservations are appropriate for those multicast applications whose application-level constraints prohibit all data sources from transmitting simultaneously; one example is audio conferencing, where a limited number of people talk at once. Thus, each receiver might issue a WF reservation request for twice one audio channel (to allow some over-speaking). On the other hand, the FF and DF styles create independent reservations for the flows from different senders; this is required for video signals, whose 'silence' periods, if any, are uncoordinated among different senders.

The essential difference between the FF and DF styles is that the DF style allows a receiver to switch channels without danger of an admission denial due to limited resources (unless a topology change reroutes traffic along a lower-capacity path or new senders appear), once the initial reservations have been made.

Other reservation styles may be defined in the future.

2.3 RSVP Protocol Mechanisms

2.3.1 RSVP Messages

Each receiver host sends RSVP reservation (*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 senders. If a reservation request fails at any node, an RSVP error message is returned to the receiver; however, RSVP sends no positive acknowledgment messages to indicate success. *Resv* messages are finally delivered to the sender hosts, so that the hosts can set up appropriate traffic control parameters for the first hop.

```
  Sender                                           Receiver

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

Figure 2: RSVP Messages

Each sender transmits 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.

The path state is currently used by RSVP to route the *Resv* messages in the reverse direction from each receiver to all selected senders for a given session. In the future, this function may be assumed by routing protocols. *Path* messages have other functions; they carry the following additional information:

- A sender template, which describes the format of data packets that the sender will originate.

  The sender 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 match the filterspecs 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.

  ISSUE:

  > Should sender templates be defined to be precisely filterspecs, or should templates and filterspecs be allowed to use different syntax?

- 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.

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

2.3.2 Soft State

To maintain reservation state, RSVP keeps *soft state* in router and host nodes. RSVP soft state is created and periodically refreshed by *Path* and *Resv* messages, and it can be removed at each node by explicit *Teardown* messages. RSVP also has a timer-driven cleanup procedure if no message is received within a *cleanup timeout* interval.

When the route changes, the next *Path* message will initialize the path state on the new route, and future *Resv* messages will establish reservation state while the state on the now-unused segment of the route times out. Thus, whether a message is "new" or a "refresh" is determined separately at each node, depending upon the existence of 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 all its messages as IP datagrams without any reliability enhancement. Periodic transmission of refresh messages by hosts and routers is expected to replace any lost RSVP messages. However, the traffic control mechanism should be statically configured to grant high-reliability service to RSVP messages, to protect RSVP messages from severe congestion.

If the set of senders Si or receivers Rj changes, or if any of the receivers' reservation requests change, the RSVP state is adjusted accordingly. RSVP believes the latest *Path* and *Resv* messages (ignoring the possibility of reordering). To modify a reservation, a receiver simply starts sending the new values. It is not necessary (although it may sometimes be desirable, when the resources being consumed are "valuable"), to tear down the old reservation explicitly.

When a *Resv* message is received at a router or sender host, the RSVP module checks whether the message is a new or a modified 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), as determined by the path (or routing) state. If RSVP on the node rejects the reservation request due to admission control failure or to some processing error, it discards the *Resv* message and returns a RSVP error message to the originating receiver host. If the request modifies a previous reservation, RSVP may immediately remove the old state, or it may simply let the old state time out since it is no longer being refreshed; the details depend upon the style and the implementation.

```
 Previous    Incoming                Outgoing         Next
 Hops        Interfaces              Interfaces       Hops


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

Figure 3: Router Using RSVP

Figure 3 illustrates RSVP's model of a router node. Each data stream arrives from a *previous hop* through a corresponding *incoming interface* and departs through one or more *outgoing interface(s)*. Since the same host may be hosting both sender and receiver applications for a given session, the same physical interface may act in both the incoming and outgoing roles (for different data streams).

The interfaces shown in Figure 3 may be physical interfaces (e.g., to point-to-point links), or they may be logical interfaces that reach multiple nodes through the same physical interface. Multiple previous hops and/or next hops through a given physical interface can result from either the connected network being a shared medium (e.g., an Ethernet), or from the existence of non-RSVP routers in the path to the next RSVP hop (see Section 3.5). It is generally necessary for RSVP to track both logical and physical interfaces on both the incoming and outgoing sides.

2.3.3 Merging RSVP Messages

Whenever possible, the control information arriving in RSVP messages for a given session is combined into fewer outgoing messages; this is known generically as *merging*. Those messages that cause a state change are forwarded without delay, while the refresh messages may be merged into fewer messages, perhaps only one per session.

For *Path* messages, merging implies collecting together the Sender Descriptors from multiple incoming messages into a single outgoing *Path* message. For *Resv* messages, merging implies that only the essential (e.g., the largest) reservation requests need be forwarded, once per refresh period; redundant messages are *purged*. A successful reservation request will propagate as far as the closest point(s) 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 another, equal or larger, reservation request.

To allow merging, each node must save the state from received messages and then periodically generate cumulative *Path* and *Resv* messages from the saved state, to be forwarded in place of the received messages. Thus, new refresh messages are created hop-by-hop inside the network, at a rate determined by a *refresh period*. Since messages that modify the state in a node ("new" messages) are forwarded without delay, the refresh period does not affect the rate at which new state propagates from end to end (when packets are not lost).

Although flowspecs are opaque to RSVP, merging requires the ability to determine which of two flowspecs is "larger", i.e. whether one represents a stricter request (and hence represents a larger resource commitment) than the other. However, a flowspec may be a complex multi-dimensional vector, so the "larger-than" relationship may not be defined for a given pair of flowspecs. For example, consider two flowspecs Fls1 and Fls2, where Fls2 asks for a lower throughput but shorter delay that Fls1. It is not clear which is "larger", so we say they are *incompatible*.

There are several possible solutions to merging incompatible flowspecs.

(1) Compare on a single dimension, e.g., compare the throughput requirement (average bit rate) only.

(2) Construct a third flowspec that is greater than each of the two being compared. In the example above, we could construct a third flowspec Fls3 by combining the higher throughput from Fls1 with the lower delay from Fls2.

(3) Treat the compatibility as an error that should be avoided by applications.

The choice of one of these approaches should be governed by flags in the flowspec itself, not by RSVP.

Note that this problem cannot be avoided by refraining from merging flowspecs. If incompatible flowspecs were not merged at a particular node A, then they would arrive at the next node upstream, say B, in separate *Resv* messages. This may also happen if there are multiple next hops across the same outgoing interface. Node B would have to make a reservation for the largest flowspec, if that is defined, or one that dominates all the given flowspecs; that is, it must merge the unmerged reservations. Thus, failing to merge simply moves the problem one node upstream.

This mechanism, reserving for the highest demand at each node, allows an application to increase an existing reservation request immediately (assuming admission control does not fail for the larger flowspec). Decreasing a reservation has to be handled more cautiously, however. The arrival of a *Resv* message with an apparently decreased reservation might be caused by the loss of a merged *Resv* message downstream. Therefore, an RSVP should not "believe" a reservation decrease until the cleanup timeout has passed.

The refresh period and the cleanup timeout must obey the following general principles:

A. The refresh period must be long enough to keep RSVP overhead at an acceptable level.

B. The refresh period should be short enough to allow quick adaptation to route and multicast membership changes.

Applications may differ in their sensitivity to service outages, and therefore they should be able to adjust the refresh period for their session state. However, the technique of *local repair* (see Section 3.3.3) can provide rapid adaptation despite a long refresh period.

C. The timeout period must be long enough to allow for loss of individual RSVP messages.

2.3.4 Teardown

As an optimization to release resources quickly, RSVP *teardown* messages remove path and reservation state without waiting for the cleanup timeout period. RSVP messages are not delivered reliably, but the state will eventually time out even if a teardown message is lost.

Teardown may be initiated either by an end system (sender or receiver), or by a router as the result of state timeout. A router may also initiate a teardown message as the result of router or link failures detected by the routing protocol. A teardown, once initiated, will be forwarded hop-by-hop without delay.

There are two types of RSVP Teardown message, *PathTear* and *ResvTear*. A *PathTear* message travels towards all receivers downstream from its point of initiation and tears down path state along the way, while an *ResvTear* message tears down reservation state and travels towards all senders upstream from its point of initiation.

A particular reservation on a node may be shared among multiple senders and/or receivers, but it must apply to a unique next hop (and outgoing interface). The receipt of an *ResvTear* message implies that the corresponding reservation state has been removed downstream, so that the reservation can safely be deleted locally. Again, the local node will only forward the teardown message upstream when the state named in the message has been entirely removed locally. As a result, an *ResvTear* message will prune the reservation state back (only) as far as possible. Note that the *ResvTear* message will cease to be forwarded at the same node where merging suppresses forwarding of the corresponding *Resv* messages.

Consider the router configuration shown in Figure 4 below. Assume that there are reservations for source S1 on both outgoing interfaces (c) and (d), and that the receiver R1 wants to tear down its reservation state for S1. R1's *ResvTear* message arriving through interface (c) indicates that all reservation state for (this session and) sender S1 has been removed downstream. The current node therefore removes the S1 reservation state from interface (c). However, since there will still be an S1 reservation on interface (d), the *ResvTear* message will not be forwarded any further.

However, if the outgoing interface connects to a shared medium or if there is a non-RSVP router immediately downstream, then there may be multiple next-hop RSVP nodes downstream that are reached through the same outgoing interface, say (c). Then a single reservation may be shared among multiple next hops. RSVP must tag each reservation with the next hop(s) from which the *Resv* messages came, for use by teardown to avoid deleting shared state.

Deletion of path state, whether as the result of a teardown message or because of timeout, may force adjustments in related reservation state to maintain consistency in the local node. Consider the path state for a sender S; the related reservation state would be as follows.

- Wildcard-Filter style: If S is the only sender to the session, delete the reservation.
- Fixed-Filter style: Delete reservations made for S.
- Dynamic-Filter style: Reduce total reservation if it now exceeds the total number of remaining senders.

2.4 Examples

We use the following notation for a *Resv* message:

1. Wildcard-Filter

   WF(*{r})

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

2. Fixed-Filter

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

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

3. Dynamic-Filter

   DF(n, {r} ;) or DF(n, {r} ; S1, S2, ...)

   This message carries the count n of channels to be reserved, each using common flowspec r. It also carries a list, perhaps empty, of filterspecs defining senders.

Figure 4 shows schematically a router with two previous hops labeled (a) and (b) and two outgoing interfaces labeled (c) and (d). This topology will be assumed in the examples that follow. 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).

In addition to the connectivity shown in 4, we must also specify the multicast routing within this node. Assume first that data packets (hence, *Path* messages) from each Si shown in Figure 4 is routed to both outgoing interfaces. Under this assumption, Figures 5, 6, and 7 illustrate Wildcard-Filter reservations, Fixed-Filter reservations, and Dynamic-Filter reservations, respectively.

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

Figure 4: Router Configuration

In Figure 5, the "Receive" column shows the *Resv* messages received over outgoing interfaces (c) and () 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.

Figure 6 shows Fixed-Filter style reservations. Merging takes place among the flow descriptors (i.e., filter spec, flowspec pairs). 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(S1{B}) and FF(S1{3B}) are merged, the single message FF(S1{3B}) is sent to previous hop (a), towards S1.

```
                      |
        Send          |       Reserve              Receive
                      |
                      |       -------
  WF( *{3B} ) <- (a)  |  (c) | * {3B}|    (c) <- WF( *{B} )
                      |      |_______|
                      |
----------------------|---------------------------------------
                      |       -------
  WF( *{3B} ) <- (b)  |  (d) | * {B} |    (d) <- WF( *{3B} )
                      |      |_______|
```

Figure 5: Wildcard-Filter Reservation Example 1

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
                      |
                      |       --------
 FF( S1{3B} ) <- (a)  |  (c) | S1{B}  |   (c) <- FF( S1{B}, S2{5B} )
                      |      |________|
                      |      | S2{5B} |
                      |      |________|
----------------------|--------------------------------------------
                      |       --------
             <- (b)   |  (d) | S1{3B} |   (d) <- FF( S1{3B}, S3{B} )
 FF( S2{5B}, S3{B} )  |      |________|
                      |      | S3{B}  |
                      |      |________|
```

Figure 6: Fixed-Filter Reservation Example

```
                       |
        Send           |      Reserve             Receive
                       |
                       |     ________
 DF( 1,{B}; S1) <- (a) |  (c) |  S1{B} |  (c) <- DF( 2,{B}; S1, S2)
                       |      |________|
                       |      |  S2{B} |
                       |      |________|
                       |
-----------------------|-----------------------------------------
                       |     ________
 DF( 2,{B}; S2) <- (b) |  (d) |  S1{B} |   (d) <- DF( 2,{B}; S1)
                       |      |________|
                       |      |   ?{B} |
                       |      |________|
```

Figure 7: Dynamic-Filter Reservation Example

Figure 7 shows an example of Dynamic-Filter reservations. The receivers downstream from interface (d) have requested two reserved channels, but selected only one sender, S1. The node reserves $min(2, 3) = 2$ channels of size B on interface (d), and it then applies any specified filters to these channels. Since only one sender was specified, one channel has no corresponding filter, as shown by '?'.

Similarly, the receivers downstream of interface (c) have requested two channels and selected senders S1 and S2. The two channels might have been one channel each from R1 and R2, or two channels requested by one of them, for example.

A router should not reserve more Dynamic-Filter channels than the number of upstream sources (three, in the router of Figure 7). Since there is only one source upstream from previous hop (a), the first parameter of the DF message (the count of channels to be reserved) was decreased to 1 in the forwarded reservations. However, this is unnecessary, because the routers upstream will reserve only one channel, regardless.

When a DF reservation is received, it is labeled with the IP address of the next hop (RSVP-capable) router, downstream from the current node. Since the outgoing interface may be directly connected to a shared medium network or to a non-RSVP-capable router, there may be more than one next-hop node downstream; if so, each sends independent DF *Resv* messages for a given session. The number N' of DF channels reserved on an outgoing interface is given by the formula:

$N' = min(D1 + D2 + ...Dn, Ns)$,

where Di is the D value (channel reservation count) in a *Resv* from the ith next-hop node.

The three examples just shown assume full routing, i.e., data packets from S1, S2, and S3 are routed to both outgoing interfaces. Assume the routing shown in Figure 8, in which data packets from S1 are not forwarded to interface (d) (because the mesh topology provides a shorter path for $S1- > R3$ that does not traverse this node).

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

Figure 8: Router Configuration

Under this assumption, Figure 9 shows Wildcard-Filter reservations. Since there is no route from (a) to (d), the reservation forwarded out interface (a) considers only the reservation on interface (c), so no merging takes place in this case.

2.5 Host Model

Before a session can be created, the session socket, comprised of DestAddress and ResvID, 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.

```
                       |
       Send            |       Reserve             Receive
                       |
                       |       -------
  WF( *{B} ) <- (a)    |  (c) | * {3B}|    (c) <- WF( *{B} )
                       |      |_______|
                       |
-----------------------|----------------------------------------
                       |       -------
  WF( *{3B} ) <- (b)   |  (d) | * {B} |    (d) <- WF( * {3B} )
                       |      |_______|
```

Figure 9: Wildcard-Filter Reservation Example – Partial Routing

- If a receiver starts sending *Resv* messages (H4) before any *Path* messages have reached it (and if path state is being used to route *Resv* messages), RSVP will return error messages to the receiver.

  The receiver may simply choose to ignore such error messages, or it may avoid them by waiting for *Path* messages before sending *Resv* messages.

A specific application program interface (API) for RSVP 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.

3 Functional Specification

There are currently 6 types of RSVP messages: *Path*, *Resv*, *PathTear*, *ResvTear*, *PathErr*, and *ResvErr*.

3.1 Message Formats

3.1.1 Path Message

```
      0             1              2             3
+-------------+-------------+-------------+-------------+
| Vers | Type |    Flags    |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+
|                      DestAddress                      |
+-------------+-------------+-------------+-------------+
|                        ResvID                         |
+-------------+-------------+-------------+-------------+
|                    Refresh Period                     |
+-------------+-------------+-------------+-------------+
|                    State TTL Time                     |
+-------------+-------------+-------------+-------------+
|                 Previous Hop Address                  |
+-------------+-------------+-------------+-------------+
|       ///////////////     |         SD Count          |
+-------------+-------------+-------------+-------------+
|                 Authentication Field                  |
//                          ...                        //
+-------------+-------------+-------------+-------------+
|                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

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, ResvID

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.

SD Count

Count of Sender Descriptors that follow.

Authentication Field

A variable-length authentication field to identify and perhaps authenticate the principal making this reservation request. The field has the following form:

```
+-------------+-------------+-------------+-------------+
|  AuthLen    |  AuthType   |                           |
+-------------+-------------+                           +
//                   Authentication Info               //
+-------------+-------------+-------------+-------------+
```

The AuthLen octet contains the integer length of the field in fullwords, and AuthType specifies the format of the field. See Section 3.6.1 for currently defined authentication field formats. If there is no authentication information, AuthLen will be zero, but the Authentication Field will still occupy one fullword in the message.

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 error 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 two variable-length fields: a sender template that defines the format of data packets and a corresponding Flowspec that describes the traffic characteristics. The sender template has the form of a filterspec, and a Sender Descriptor has the form defined below for a Flow Descriptor (see also Section 3.6.1). The flowspec may be omitted, in which case its length field will be zero (but it will still occupy one fullword in the Sender Descriptor).

The Sender template is retained in the Path state in order to validate filterspecs in *Resv* messages. Suppose that a filterspec consisted of a simple (value,mask) pair (Vf,Mf) to be applied to the headers of the data packets (the actual format is slightly more complex; see Section 3.6.1). Then the corresponding template would be a (value,mask) pair defining those bits of the data packet headers that are fixed. While processing a reservation using filterspec (Vf,Mf) for the sender with template (Vs,Ms), RSVP can then test whether $Vf\&(Mf\&Ms) = Vs\&(M\&Ms)$. If not, this filterspec cannot possibly match the data stream from this sender at any node upstream, and the reservation can be rejected with an error message back to the receiver.

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                     |
+-------------+-------------+-------------+-------------+
|                         ResvID                        |
+-------------+-------------+-------------+-------------+
|                     Refresh Period                    |
+-------------+-------------+-------------+-------------+
|                     State TTL Time                    |
+-------------+-------------+-------------+-------------+
|                    Next Hop Address                   |
+-------------+-------------+-------------+-------------+
|                       RecvAddress                     |
+-------------+-------------+-------------+-------------+
| Dynamic Reservation Count |          FD Count         |
+-------------+-------------+-------------+-------------+
|                  Authentication Field                 |
//                           ...                       //
+-------------+-------------+-------------+-------------+
|                  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 node 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

The following flag bit combinations define the reservation style:

001xxxxx = Wildcard-Filter
010xxxxx = Fixed-Filter
011xxxxx = Dynamic-Filter

Next Hop Address

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

The Next Hop Address is used to support teardown. This field is initialized by a receiver to its IP address and must be updated at each router hop as the *Resv* message is forwarded.

RecvAddress

The IP address of (one of the) receiver(s) that originated this message, or one of the *Resv* messages that was merged to form 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 Flow Descriptors in the Flow Descriptor List.

Flow Descriptor List

A list of Flow Descriptors, i.e., (Filterspec, flowspec) pairs, to define individual reservation requests. 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   |  FiltSType  |                           |
+-------------+-------------+                           +
//                   Filter Spec ...                   //
+-------------+-------------+-------------+-------------+
|  FlowSLen   |  FlowSType  |                           |
+-------------+-------------+                           +
//                   Flow Spec ...                     //
+-------------+-------------+-------------+-------------+
```

Here FiltSLen and FlowSLen are one-octet fields specifying the lengths in fullwords (including the length byte) of the filterspec and flowspec, respectively, and FiltSType and FlowSType are one-octet fields defining the corresponding field formats. See Section 3.6.1 for currently defined formats.

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 Error Messages

There are two types of RSVP error messages: *PathErr* messages result from *Path* messages and travel towards senders, while *ResvErr* messages result from *Resv* messages and travel towards receivers. RSVP error messages are triggered only by processing of *Path* and *Resv* messages; errors encountered while processing error or teardown messages must not create error messages.

A *PathErr* message has the following form:

```
       0             1              2             3
+-------------+-------------+-------------+-------------+
| Vers | Type |    Flags    |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+
|                       DestAddress                     |
+-------------+-------------+-------------+-------------+
|                         ResvID                        |
+-------------+-------------+-------------+-------------+
|  Error Code | Error Index |        Error Value        |
+-------------+-------------+-------------+-------------+
|       ////////////// (ignored) //////////////////     |
+-------------+-------------+-------------+-------------+
|       ////////////// (ignored) //////////////////     |
+-------------+-------------+-------------+-------------+
|     /// Reserved ///      |         SD Count          |
+-------------+-------------+-------------+-------------+
|                 Authentication Field                  |
//                           ...                       //
+-------------+-------------+-------------+-------------+
|                Sender Descriptor List                 |
//                           ...                       //
+-------------+-------------+-------------+-------------+
```

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

RSVP Fields:

RSVPType

3 = *PathErr* message

Error Code

A one-octet error description.

01 = Insufficient memory

02 = Count Wrong

The SD Count field does not match length of message.

Error Index

Position of Sender Descriptor that caused the error within Sender Descriptor List. An integer between zero and SD Count - 1.

Error Value

(Unused)

A *ResvErr* message has the following form:

```
       0             1              2             3
+-------------+-------------+-------------+-------------+
| Vers | Type |    Flags    |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+
|                      DestAddress                      |
+-------------+-------------+-------------+-------------+
|                        ResvID                         |
+-------------+-------------+-------------+-------------+
|  Error Code | Error Index |        Error Value        |
+-------------+-------------+-------------+-------------+
|       ////////////// (ignored) //////////////////     |
+-------------+-------------+-------------+-------------+
|       ////////////// (ignored) //////////////////     |
+-------------+-------------+-------------+-------------+
|                      RecvAddress                      |
+-------------+-------------+-------------+-------------+
| Dynamic Reservation Count |         FD Count          |
+-------------+-------------+-------------+-------------+
|                 Authentication Field                  |
//                          ...                        //
+-------------+-------------+-------------+-------------+
|                 Flow Descriptor List                  |
//                          ...                        //
+-------------+-------------+-------------+-------------+
```

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

RSVP Fields:

RSVPType

4 = *ResvErr* message

Error Code

A one-octet error description.

DEFINE THESE VALUES IN AN APPENDIX??

01 = Insufficient memory

02 = Count Wrong
The FD Count field does not match length of message.
03 = No path information for this Resv
04 = No Sender information for this Resv
There is path information, but it does not include the sender specified in any of the Filterspecs listed in the Resv messager.
05 = Incorrect Dynamic Reservation Count
Dynamic Reservation Count is zero or less than FD Count.
06 = Filterspec error
07 = Flowspec syntax error
08 = Flowspec value error
Internal inconsistency of values.
[What should be done with Flowspec Feature Not Supported?]
09 = Resources unavailable
[Sub-reasons? Depend upon traffic control and admission control algorithms?]

Error Index

Position of Flow Descriptor that caused the error within Flow Descriptor List. An integer between zero and FD Count - 1.

Error Value

Specific cause of the error described by the Error Code.
DEFINE THESE VALUES IN AN APPENDIX??

An error message may be duplicated and forwarded unchanged. Since *Path* and *Resv* messages may be merged, an error condition must be disseminated to all RSVP client applications whose requests may have contributed to the error situation. Therefore, RSVP error messages must be propagated and perhaps duplicated hop-by-hop. For this purpose, an error message must include all the information used to route the original message that caused the error: the Sender Descriptor List, Flags, RecvAddress, and Flow Descriptor List fields, as appropriate. In particular, a *ResvErr* message carries the same style flags as the *Resv* message that caused the error.

To ease implementation, the error message formats are chosen to match the formats of the messages whose processing caused the error. In particular, a *Path* or *Resv* message that encounters an error can be simply converted to the corresponding error message by overwriting the Type and the Refresh Period fields.

A *PathErr* message is forwarded to all previous hops for all senders listed in the Sender Descriptor List. The routing of a *ResvErr* message is more complex.

- An error in a filterspec should be detected at the first RSVP hop from the receiver application, normally within the receiver host. However, an error caused by a flowspec, normally an admission control failure, may be detected somewhere along the path(s) to the sender(s).

- The router that creates a *ResvErr* message as the result of processing a *Resv* message should send the *ResvErr* message out the interface through which the *Resv* arrived.
- In succeeding hops, the routing of a *ResvErr* message depends upon its style. In general, a *ResvErr* message is sent on a pruned version of the multicast distribution tree for the session; those branches that do not have reservations for any of the specified senders are pruned off.

  A DF-style or WF-style *ResvErr* message is forwarded on all outgoing interfaces for which there is already a reservation of the corresponding style.

  A FF-style *ResvErr* message is forwarded on all outgoing interfaces for which there is already a FF-style reservation for the sender (filterspec) corresponding to the error.

At the end host, RSVP delivers a copy of every relevant error message to its local application clients. It examines the set of RSVP requests that local clients have made through the API, and notifies every client that contributed to the error message. A match is required between the session, filters (senders), and reservation styles of the error message and the corresponding state in the latest API requests. A particular notification should include only the information (e.g., filters) relevant to that application.

3.1.4 Teardown Messages

There are two types of RSVP Teardown message, *PathTear* and *ResvTear*. A *PathTear* message tears down path state and travels towards all receivers downstream from its point of initiation. A *ResvTear* message tears down reservation state and travels towards all senders upstream from its point of initiation.

A *PathTear* message has the same format as a *Path* message, except that in a *PathTear* message:

- Type field = 5
- Refresh Period and State TTL Time fields are ignored.

A *ResvTear* message has the same format as a *Resv* message, except that in a *ResvTear* message:

- Type field = 6
- Refresh Period and State TTL Time fields are ignored.

Any Flowspec components of Flow Descriptors in a *ResvTear* or *PathTear* message are ignored.

Teardown messages are processed in the following way.

- *PathTear*

  Processing a *PathTear* message is straightforward. For each sender S in the message, the node removes path state for S and also deletes all related reservations. Finally, the node forwards the original *PathTear* message to all outgoing interfaces through which data packets from some S in the packet would be routed. That is, *PathTear* forwarding rules are the same as those for *Path* messages.

- *ResvTear*

  Processing an *ResvTear* message is more complex. Suppose a *ResvTear* message arrives through outgoing interface OI from next hop NH. For each sender S listed in the *ResvTear* message, the node checks the reservation, if any, for S on OI. If there is a reservation and if this reservation is shared among more than one next hop, then the only action is to remove NH from the list of next hops sharing this reservation. If there is only a single next hop, then the reservation is deleted. Finally, the node forwards the original *ResvTear* message to all incoming interfaces for senders listed in the message. That is, *ResvTear* forwarding rules are the same as those for *Resv* messages.

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.

L1: When an RSVP message is received through a 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.

L2: Upon receipt of a *Path* message in particular, 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, DestAddress) pairs. Each route consists of a list of outgoing interfaces; these lists (with the incoming interfaces deleted by rule L1) 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.

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

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

Figure 10: Example: Rule (1) for Preventing Loops.

Since *Path* messages don't loop, they create path state defining a loop-free path to each sender. Similarly, *Resv* messages directed to particular senders cannot loop. However, rules L1 and L2 cannot protect against looping *Resv* messages that are directed towards all senders (WF or DF styles). The following three rules are needed for this purpose.

L3: Each *Resv* message carries a receiver address in the RecvAddress field. 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.

L4: When a *Resv* message is received, the Reverse Path Forwarding rule is applied to the receiver address in the message; that is, the message must be discarded unless it arrives on the interface that is the preferred route to the receiver.

L5: A *Resv* message whose RecvAddress matches one of the IP addresses of the local node must be discarded without processing.

Figure 10 illustrates the effect of the rule L1 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.

The loop-suppression rules for *Resv* messages also prevent looping of *ResvTear* messages. Note that *ResvTear* messages are otherwise subject to fast loops, since they are not delayed by a refresh timeout period.

PathErr messages are routed upstream by the same rules used for FF and DF *Resv* messages (there is no equivalent of wildcard-filter for routing a *PathErr* message). Similarly, *ResvErr* messages are routed by the rules for *Path* messages. For reasons explained above, no special loop-suppression rules are required in either case.

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.

3.3.1 Time Parameters

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: $Rmin <= R <= T <= Tmax$.

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.

3.3.2 Teardown

Teardown messages, like other RSVP messages, are sent as datagrams and may be lost (although a QoS is used that should minimize the chances of congestive loss of RSVP messages). To increase the reliability of teardown, Q copies of any given teardown message can be sent. Note that if the iteration count Q on initiating teardown messages is > 1, then the state cannot actually be deleted until Q teardowns have been sent. The state would be placed in a "moribund" status meanwhile.

The appropriate value of Q is an engineering issue. $Q = 1$ would be the simplest and may be adequate, since unrefreshed state will time out anyway; teardown is an optimization. Note that if one or more teardown hops are lost, the router that failed to receive a teardown message will time out its state and initiate a new teardown message beyond the loss point. Assuming that RSVP message loss probability is small (but non-zero), the longest time to delete state will seldom exceed one state timeout time K*Tab.

Here is an example. Here G1, G2, G3, and G4 are routers between a sender S and a receiver R. S initiates a *PathTear* message (denoted by "PT"), but this message is lost between routers G2 and G3. Since G2 has deleted its state for S, G2 will cease refreshing G3 (though G3 is still refreshing G4, etc.)

```
     PT       PT       PT
S ---> G1 ---> G2 -->x   G3         G4         R
```

After a time K*Tab, G3's state will time out, and G3 will initiate a teardown for S path state:

```
          PT        PT
    G3 ----> G4 ---->  R
```

If some hop of this chain is lost, there will again be state timeout to continue the teardown. This process should terminate in a few timeout periods.

3.3.3 Local Repair

To accommodate merging, RSVP uses hop-by-hop refreshing of state, where each node sends refreshes to its next/previous hops periodically. However, as an optimization, local events could be used to trigger the RSVP module to send such refreshes to any time. For example, suppose that the local routing protocol module were able to notify the RSVP module that a route has changed for particular destinations. The RSVP module could use this information to trigger an immediate refresh of state for these destinations along the new route. This would allow fast adaptation to routing changes without the overhead of a short refresh period.

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 the delay of RSVP packets 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.

The largest RSVP message is 556 bytes.

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 may still be possible.

3.6 Interfaces

3.6.1 Reservation Parameters

All variable-length RSVP parameters use the same general format. They begin with a length octet followed by a type octet, and occupy an integral number of fullwords. The length octet specifies the total length of the parameter in fullwords or zero to indicate no parameter. An RSVP implementation can store and pass such parameters as opaque objects.

- Flowspec Format

  Flowspec type 1 is specific to the CSZ packet scheduler [CSZ92]. It has the following format:

```
+-----------+-----------+-----------+-----------+
| FlowSLen=6|FlowSType=1|      VFoffset         |
+-----------+-----------+-----------+-----------+
|    QoS Type (Guaranteed, Predictive, ...)     |
+-----------+-----------+-----------+-----------+
```

```
|           Max end-to-end delay (ms)           |
+-----------+-----------+-----------+-----------+
|           Average data rate (bits/ms)         |
+-----------+-----------+-----------+-----------+
|           Token Bucket Depth (bits)           |
+-----------+-----------+-----------+-----------+
|           Global Share Handle                 |
+-----------+-----------+-----------+-----------+
```

Flowspec format 2 is defined in RFC-1363 [Partridge92].

- Filterspec Format

  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 an optional variable-length mask-and-value pair VF, in the following format:

```
+-----------+-----------+-----------+-----------+
|  FiltSLen |FiltSType=1|      VFoffset         |
+-----------+-----------+-----------+-----------+
|               Sender IP Address               |
+-----------+-----------+-----------+-----------+  ---
|                 V: VF Value Part              |   Nf
/                                               / octets
/                                               /
+-----------+-----------+-----------+-----------+  ---
|                 M: VF Mask Part               |   Nf
/                                               / octets
/                                               /
+-----------+-----------+-----------+-----------+  ---
```

  The value M and the mask V each have length:

  $Nf = (4 * FiltSLen - 8)/2$ octets.

  M and V define a filter that uses a mask-and-match algorithm applied to the packet at VFoffset octets from the beginning of the IP header. The minimum length of this format of sender template is 7 octets (FiltSLen = 2).

  A wildcard Filterspec, which will match any sender host, has zero for the Sender IP Address [If VM part zero also, could shorten to FiltSLen = 2].

  To speed RSVP processing, a filterspec that appears in an RSVP message use the following *canonical form.*

- The high-order octet of the mask M must be non-zero (this can always be achieved by adjusting VFoffset).
- The (V,M) part must not include either the sender or receiver address of the IP header; these are carried explicitly.

ISSUE:

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

- Authenticator Format

  The following simple form of authenticator is defined:

```
+-----------+-----------+-----------+-----------+
|   AuthLen | AuthType=1|                       |
+-----------+-----------+                       +
|             Mailbox name: user@domain         |
//                                             //
+-----------+-----------+-----------+-----------+
```

  The rules for merging and interpreting this field require 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. 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 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.

Register

```
Call: REGISTER( DestAddress, ResvID, SND-flag, RCV-flag,
            [, DROP-flag] [, rsvpTTL] [, SenderTemplate] [, flowspec]
                        [, UserCredentials] )  -> session-id
```

This call initiates RSVP processing for the session (DestAddress, ResvID). If successful, the call returns immediately with a local session identifier “session-id”, which may be used in subsequent calls. Following this call, an asynchronous ERROR or EVENT call (see below) may occur at any time.

SND-flag should be set true if the host will send data, and RCV-flag should be set true if the host will receive data. Setting neither true is an error. The optional parameters DROP-flag, rsvpTTL, SenderTemplate, and Flowspec should be supplied only if SND-flag is true.

DROP-flag indicates that session data packets that do not match any active filter in some node should be dropped at that node; otherwise, such packets will be forwarded using a best-effort QoS. The rsvp-TTL parameter specifies the IP Time-to-Live field that will be used in *Path* messages. The value of rsvp-TTL should match the TTL field to be sent in data packets, so they will have the same multicast scope.

A REGISTER call with SND-flag equals TRUE will initiate the transmission of *Path* messages.

Reserve

```
Call: RESERVE( session-id, style [, DF-chan-count]
                 Flowspec-list, Filterspec-list)
```

A receiver uses this call to make a resource reservation for the session registered as ‘session-id’. The style parameter is an integer index indicating the reservation style. The DF-chan-count parameter, indicating the number of Dynamic Filter channels to be reserved, should only be included if the style is DF.

The first RESERVE call will initiate the periodic transmission of *Resv* messages. A later RESERVE call may be given to modify the parameters of the earlier call (but note that changing the reservations may result in admission control failure, depending upon the style).

The RESERVE call returns immediately. Following this call, an asynchronous ERROR or EVENT call may come at any time.

Release

Call: RELEASE(session-id)

This call will terminate RSVP state for the session specified by session-id. It will send appropriate *Teardown* messages and cease sending refreshes.

Error Upcall

```
Call: ERROR( ) -> session-id, error-type, error-code
                        [, flowspec] [, filterspec]
```

This upcall may occur asynchronously at any time after a REGISTER call and before a RELEASE call, to indicate an error. The allowed values of error-type and error-code depend on whether the node is sending, receiving, or both.

The ERROR upcall reporting an admission control failure to a receiver will specify in 'flowspec' the flowspec that actually failed. This may differ from the flowspec specified by this application in a RESERVE call, due to upstream merging with reservation requests from other receivers.

Event Upcall

```
Call: EVENT( ) -> session-id, info-type,
                  [, flowspec-list] [, filterspec-list]
```

This upcall may occur asynchronously at any time after a REGISTER call and before a RELEASE call, to signal an event and to pass information to the application.

The 'info-type' field indicates one of two possible event types. A Path event indicates the receipt of a *Path* message, indicating to the application that there is at least one active sender. A Reservation event indicates the receipt of a *Resv* message, indicating to the application that there is at least one receiver. Although these messages are repeatedly received, the API should make the corresponding asynchronous upcall to the application only on the first event, or when the information to be reported changes.

ISSUE:

> The precise form and function of the flowspec-list and filterspec-list parameters are to be determined.

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.

1. Make a Reservation

   Call: $Rhandle = TCAddFlow(Flowspec, DropFlag, [SessionFilterspec[, SenderFilterspec]])$

   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 FF reservation requests, RSVP knows about sharing and calls AddFlow only for distinct source pipes.

For DF 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 may send different versions of outgoing *Path* messages on different interfaces, for the same source and destination addresses.

- Discover (Virtual) Interface List

  RSVP must be able to learn what real and virtual interfaces exist.

4 ACKNOWLEDGMENTS

Lixia Zhang, Scott Shenker, Deborah Estrin, Dave Clark, Sugih Jamin, Shai Herzog, Steve Berson, Steve Deering, Bob Braden, and Daniel Zappala have all made contributions to the design of RSVP. We are grateful to Jamin, Herzog, and Berson 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*, RFC 1633, ISI, MIT, and PARC, June 1994.

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

[Partridge92] Partridge, C., *A Proposed Flow Specification*, RFC 1363, BBN, September 1992.

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

Security Considerations

As noted in Section 2.1, the ability to reserve resources will create a requirement for authentication of users who request reservations. An authentication field has been included in this version of the protocol spec, but further study on its format and usage will be required.

Authors' Addresses

Lixia Zhang
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Phone: (415) 812-4415
EMail: Lixia@PARC.XEROX.COM

Bob Braden
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Phone: (310) 822-1511
EMail: Braden@ISI.EDU

Deborah Estrin
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0871

Phone: (213) 740-4524
EMail: estrin@USC.EDU

Shai Herzog

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
Palo Alto, CA 94304

Phone: (310) 822 1511
EMail: Herzog@ISI.EDU

Sugih Jamin
Computer Science Department

University of Southern California
Los Angeles, CA 90089-0871

Phone: (213) 740-6578
EMail: jamin@catarina.usc.edu