Network Working Group
Request for Comments: 2205
Category: Standards Track

R. Braden, Ed.
ISI
L. Zhang
UCLA
S. Berson
ISI
S. Herzog
IBM Research
S. Jamin
Univ. of Michigan
September 1997

# Resource ReSerVation Protocol (RSVP) --

# Version 1 Functional Specification

## Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

## 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.



Motorola Solutions, Inc., Ex1006, p. 1

Table of Contents

1. Introduction ..... 4
   1.1 Data Flows ..... 7
   1.2 Reservation Model ..... 8
   1.3 Reservation Styles .....11
   1.4 Examples of Styles .....14
2. RSVP Protocol Mechanisms .....19
   2.1 RSVP Messages .....19
   2.2 Merging Flowspecs .....21
   2.3 Soft State .....22
   2.4 Teardown .....24
   2.5 Errors .....25
   2.6 Confirmation .....27
   2.7 Policy Control .....27
   2.8 Security .....28
   2.9 Non-RSVP Clouds .....29
   2.10 Host Model .....30
3. RSVP Functional Specification .....32
   3.1 RSVP Message Formats .....32
   3.2 Port Usage .....47
   3.3 Sending RSVP Messages .....48
   3.4 Avoiding RSVP Message Loops .....50
   3.5 Blockade State .....54
   3.6 Local Repair .....56
   3.7 Time Parameters .....57
   3.8 Traffic Policing and Non-Integrated Service Hops .....58
   3.9 Multihomed Hosts .....59
   3.10 Future Compatibility .....61
   3.11 RSVP Interfaces .....63
4. Acknowledgments .....76
APPENDIX A. Object Definitions .....77
APPENDIX B. Error Codes and Values .....92
APPENDIX C. UDP Encapsulation .....98
APPENDIX D. Glossary .....102
REFERENCES .....111
SECURITY CONSIDERATIONS .....111
AUTHORS' ADDRESSES .....112



Motorola Solutions, Inc., Ex1006, p. 2

What's Changed

This revision contains the following very minor changes from the ID14 version.

- For clarity, each message type is now defined separately in Section 3.1.
- We added more precise and complete rules for accepting Path messages for unicast and multicast destinations (Section 3.1.3).
- We added more precise and complete rules for processing and forwarding PathTear messages (Section 3.1.5).
- A note was added that a SCOPE object will be ignored if it appears in a ResvTear message (Section 3.1.6).
- A note was added that a SENDER_TSPEC or ADSPEC object will be ignored if it appears in a PathTear message (Section 3.1.5).
- The obsolete error code Ambiguous Filter Spec (09) was removed, and a new (and more consistent) name was given to error code 08 (Appendix B).
- In the generic interface to traffic control, the Adspec was added as a parameter to the AddFlow and ModFlow calls (3.11.2). This is needed to accommodate a node that updates the slack term (S) of Guaranteed service.
- An error subtype was added for an Adspec error (Appendix B).
- Additional explanation was added for handling a CONFIRM object (Section 3.1.4).
- The rules for forwarding objects with unknown class type were clarified.
- Additional discussion was added to the Introduction and to Section 3.11.2 about the relationship of RSVP to the link layer. (Section 3.10).
- Section 2.7 on Policy and Security was split into two sections, and some additional discussion of security was included.
- There were some minor editorial improvements.



Motorola Solutions, Inc., Ex1006, p. 3

## 1. Introduction

This document defines RSVP, a resource reservation setup protocol designed for an integrated services Internet [RSVP93, RFC 1633]. The RSVP protocol is used by a host to request specific qualities of service from the network for particular application data streams or flows. RSVP is also used by routers to deliver quality-of-service (QoS) requests to all nodes along the path(s) of the flows and to establish and maintain state to provide the requested service. RSVP requests will generally result in resources being reserved in each node along the data path.

RSVP requests resources for simplex flows, i.e., it requests resources in only one direction. Therefore, RSVP treats a sender as logically distinct from a receiver, although the same application process may act as both a sender and a receiver at the same time. RSVP operates on top of IPv4 or IPv6, occupying the place of a transport protocol in the protocol stack. However, RSVP does not transport application data but is rather an Internet control protocol, like ICMP, IGMP, or routing protocols. Like the implementations of routing and management protocols, an implementation of RSVP will typically execute in the background, not in the data forwarding path, as shown in Figure 1.

RSVP is not itself a routing protocol; RSVP is designed to operate with current and future unicast and multicast routing protocols. An RSVP process consults the local routing database(s) to obtain routes. In the multicast case, for example, a host sends IGMP messages to join a multicast group and then sends RSVP messages to reserve resources along the delivery path(s) of that group. Routing protocols determine where packets get forwarded; RSVP is only concerned with the QoS of those packets that are forwarded in accordance with routing.

In order to efficiently accommodate large groups, dynamic group membership, and heterogeneous receiver requirements, RSVP makes receivers responsible for requesting a specific QoS [RSVP93]. A QoS request from a receiver host application is passed to the local RSVP process. The RSVP protocol then carries the request to all the nodes (routers and hosts) along the reverse data path(s) to the data source(s), but only as far as the router where the receiver’s data path joins the multicast distribution tree. As a result, RSVP’s reservation overhead is in general logarithmic rather than linear in the number of receivers.



Motorola Solutions, Inc., Ex1006, p. 4

```
              HOST                                ROUTER

 ____________________________        ____________________________
|  _______                    |      |                            |
| |       |   _______         |      |            _______         |
| |Appli- |  |       |        |RSVP |           |       |        |
| | cation|  | RSVP <---------------------------> RSVP  <---------->
| |       <-->       |        |      | _______   |       |        |
| |       |  |process|  _____ |      ||Routing|  |process|  _____ |
| |_._____|  |       -->Polcy||      ||       <-->       -->Polcy||
|   |        |__.__._| |Cntrl||      ||process|  |__.__._| |Cntrl||
|   |data       |  |   |_____||      ||__.____|     |  |   |_____||
|===|===========|==|==========|      |===|==========|==|==========|
|   |   --------|  |    _____ |      |   |  --------|  |    _____ |
|   |  |        |  ---->Admis||      |   |  |       |  ---->Admis||
|  _V__V_    ___V____  |Cntrl||      |  _V__V_    __V_____ |Cntrl||
| |      |  |        | |_____||      | |      |  |        ||_____||
| |Class-|  | Packet |        |      | |Class-|  | Packet |       |
| | ifier|==>Schedulr|================> ifier|==>Schedulr|===========>
| |______|  |________|        |data | |______|  |________|       |data
|                              |      |                            |
|____________________________|      |____________________________|
```

Figure 1: RSVP in Hosts and Routers

Quality of service is implemented for a particular data flow by mechanisms collectively called "traffic control". These mechanisms include (1) a packet classifier, (2) admission control, and (3) a "packet scheduler" or some other link-layer-dependent mechanism to determine when particular packets are forwarded. The "packet classifier" determines the QoS class (and perhaps the route) for each packet. For each outgoing interface, the "packet scheduler" or other link-layer-dependent mechanism achieves the promised QoS. Traffic control implements QoS service models defined by the Integrated Services Working Group.

During reservation setup, an RSVP QoS request is passed to two local decision modules, "admission control" and "policy control". Admission control determines whether the node has sufficient available resources to supply the requested QoS. Policy control



Motorola Solutions, Inc., Ex1006, p. 5

determines whether the user has administrative permission to make the reservation. If both checks succeed, parameters are set in the packet classifier and in the link layer interface (e.g., in the packet scheduler) to obtain the desired QoS. If either check fails, the RSVP program returns an error notification to the application process that originated the request.

RSVP protocol mechanisms provide a general facility for creating and maintaining distributed reservation state across a mesh of multicast or unicast delivery paths. RSVP itself transfers and manipulates QoS and policy control parameters as opaque data, passing them to the appropriate traffic control and policy control modules for interpretation. The structure and contents of the QoS parameters are documented in specifications developed by the Integrated Services Working Group; see [RFC 2210]. The structure and contents of the policy parameters are under development.

Since the membership of a large multicast group and the resulting multicast tree topology are likely to change with time, the RSVP design assumes that state for RSVP and traffic control state is to be built and destroyed incrementally in routers and hosts. For this purpose, RSVP establishes "soft" state; that is, RSVP sends periodic refresh messages to maintain the state along the reserved path(s). In the absence of refresh messages, the state automatically times out and is deleted.

In summary, RSVP has the following attributes:

- RSVP makes resource reservations for both unicast and many-to-many multicast applications, adapting dynamically to changing group membership as well as to changing routes.

- RSVP is simplex, i.e., it makes reservations for unidirectional data flows.

- RSVP is receiver-oriented, i.e., the receiver of a data flow initiates and maintains the resource reservation used for that flow.

- RSVP maintains "soft" state in routers and hosts, providing graceful support for dynamic membership changes and automatic adaptation to routing changes.

- RSVP is not a routing protocol but depends upon present and future routing protocols.

- RSVP transports and maintains traffic control and policy control parameters that are opaque to RSVP.



Motorola Solutions, Inc., Ex1006, p. 6

- o RSVP provides several reservation models or "styles" (defined below) to fit a variety of applications.

- o RSVP provides transparent operation through routers that do not support it.

- o RSVP supports both IPv4 and IPv6.

Further discussion on the objectives and general justification for RSVP design are presented in [RSVP93] and [RFC 1633].

The remainder of this section describes the RSVP reservation services. Section 2 presents an overview of the RSVP protocol mechanisms. Section 3 contains the functional specification of RSVP, while Section 4 presents explicit message processing rules. Appendix A defines the variable-length typed data objects used in the RSVP protocol. Appendix B defines error codes and values. Appendix C defines a UDP encapsulation of RSVP messages, for hosts whose operating systems provide inadequate raw network I/O support.

## 1.1 Data Flows

RSVP defines a "session" to be a data flow with a particular destination and transport-layer protocol. RSVP treats each session independently, and this document often omits the implied qualification "for the same session".

An RSVP session is defined by the triple: (DestAddress, ProtocolId [, DstPort]). Here DestAddress, the IP destination address of the data packets, may be a unicast or multicast address. ProtocolId is the IP protocol ID. The optional DstPort parameter is a "generalized destination port", i.e., some further demultiplexing point in the transport or application protocol layer. DstPort could be defined by a UDP/TCP destination port field, by an equivalent field in another transport protocol, or by some application-specific information.

Although the RSVP protocol is designed to be easily extensible for greater generality, the basic protocol documented here supports only UDP/TCP ports as generalized ports. Note that it is not strictly necessary to include DstPort in the session definition when DestAddress is multicast, since different sessions can always have different multicast addresses. However, DstPort is necessary to allow more than one unicast session addressed to the same receiver host.



Motorola Solutions, Inc., Ex1006, p. 7

Figure 2 illustrates the flow of data packets in a single RSVP session, assuming multicast data distribution. 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 multicast routing. Multicast distribution forwards a copy of each data packet from a sender Si to every receiver Rj; a unicast distribution session has a single receiver R. Each sender Si may be running in a unique Internet host, or a single host may contain multiple senders distinguished by "generalized source ports".

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

Figure 2: Multicast Distribution Session

For unicast transmission, there will be a single destination host but there may be multiple senders; RSVP can set up reservations for multipoint-to-single-point transmission.

## 1.2 Reservation Model

An elementary RSVP reservation request consists of a "flowspec" together with a "filter spec"; this pair is called a "flow descriptor". The flowspec specifies a desired QoS. The filter spec, together with a session specification, defines the set of data packets -- the "flow" -- to receive the QoS defined by the flowspec. The flowspec is used to set parameters in the node’s packet scheduler or other link layer mechanism, while the filter spec is used to set parameters in the packet classifier. Data packets that are addressed to a particular session but do not match any of the filter specs for that session are handled as best-effort traffic.

The flowspec in a reservation request will generally include a service class and two sets of numeric parameters: (1) an "Rspec" (R for ‘reserve’) that defines the desired QoS, and (2) a "Tspec" (T for ‘traffic’) that describes the data flow. The formats and contents of Tspecs and Rspecs are determined by the integrated service models [RFC 2210] and are generally opaque to RSVP.



Motorola Solutions, Inc., Ex1006, p. 8

The exact format of a filter spec depends upon whether IPv4 or IPv6 is in use; see Appendix A. In the most general approach [RSVP93], filter specs may select arbitrary subsets of the packets in a given session. Such subsets might be defined in terms of senders (i.e., sender IP address and generalized source port), in terms of a higher-level protocol, or generally in terms of any fields in any protocol headers in the packet. For example, filter specs might be used to select different subflows of a hierarchically-encoded video stream by selecting on fields in an application-layer header. In the interest of simplicity (and to minimize layer violation), the basic filter spec format defined in the present RSVP specification has a very restricted form: sender IP address and optionally the UDP/TCP port number SrcPort.

Because the UDP/TCP port numbers are used for packet classification, each router must be able to examine these fields. This raises three potential problems.

1. It is necessary to avoid IP fragmentation of a data flow for which a resource reservation is desired.

   Document [RFC 2210] specifies a procedure for applications using RSVP facilities to compute the minimum MTU over a multicast tree and return the result to the senders.

2. IPv6 inserts a variable number of variable-length Internet-layer headers before the transport header, increasing the difficulty and cost of packet classification for QoS.

   Efficient classification of IPv6 data packets could be obtained using the Flow Label field of the IPv6 header. The details will be provided in the future.

3. IP-level Security, under either IPv4 or IPv6, may encrypt the entire transport header, hiding the port numbers of data packets from intermediate routers.

   A small extension to RSVP for IP Security under IPv4 and IPv6 is described separately in [RFC 2207].

RSVP messages carrying reservation requests originate at receivers and are passed upstream towards the sender(s). Note: in this document, we define the directional terms "upstream" vs. "downstream", "previous hop" vs. "next hop", and "incoming interface" vs "outgoing interface" with respect to the direction of data flow.



Motorola Solutions, Inc., Ex1006, p. 9

At each intermediate node, a reservation request triggers two general actions, as follows:

1. Make a reservation on a link

   The RSVP process passes the request to admission control and policy control. If either test fails, the reservation is rejected and the RSVP process returns an error message to the appropriate receiver(s). If both succeed, the node sets the packet classifier to select the data packets defined by the filter spec, and it interacts with the appropriate link layer to obtain the desired QoS defined by the flowspec.

   The detailed rules for satisfying an RSVP QoS request depend upon the particular link layer technology in use on each interface. Specifications are under development in the ISSLL Working Group for mapping reservation requests into popular link layer technologies. For a simple leased line, the desired QoS will be obtained from the packet scheduler in the link layer driver, for example. If the link-layer technology implements its own QoS management capability, then RSVP must negotiate with the link layer to obtain the requested QoS. Note that the action to control QoS occurs at the place where the data enters the link-layer medium, i.e., at the upstream end of the logical or physical link, although an RSVP reservation request originates from receiver(s) downstream.

2. Forward the request upstream

   A reservation request is propagated upstream towards the appropriate senders. The set of sender hosts to which a given reservation request is propagated is called the "scope" of that request.

   The reservation request that a node forwards upstream may differ from the request that it received from downstream, for two reasons. The traffic control mechanism may modify the flowspec hop-by-hop. More importantly, reservations from different downstream branches of the multicast tree(s) from the same sender (or set of senders) must be " merged" as reservations travel upstream.

When a receiver originates a reservation request, it can also request a confirmation message to indicate that its request was (probably) installed in the network. A successful reservation request propagates upstream along the multicast tree until it reaches a point where an existing reservation is equal or greater



Motorola Solutions, Inc., Ex1006, p. 10

than that being requested. At that point, the arriving request is merged with the reservation in place and need not be forwarded further; the node may then send a reservation confirmation message back to the receiver. Note that the receipt of a confirmation is only a high-probability indication, not a guarantee, that the requested service is in place all the way to the sender(s), as explained in Section 2.6.

The basic RSVP reservation model is "one pass": a receiver sends a reservation request upstream, and each node in the path either accepts or rejects the request. This scheme provides no easy way for a receiver to find out the resulting end-to-end service. Therefore, RSVP supports an enhancement to one-pass service known as "One Pass With Advertising" (OPWA) [OPWA95]. With OPWA, RSVP control packets are sent downstream, following the data paths, to gather information that may be used to predict the end-to-end QoS. The results ("advertisements") are delivered by RSVP to the receiver hosts and perhaps to the receiver applications. The advertisements may then be used by the receiver to construct, or to dynamically adjust, an appropriate reservation request.

### 1.3 Reservation Styles

A reservation request includes a set of options that are collectively called the reservation "style".

One reservation option concerns the treatment of reservations for different senders within the same session: establish a "distinct" reservation for each upstream sender, or else make a single reservation that is "shared" among all packets of selected senders.

Another reservation option controls the selection of senders; it may be an "explicit" list of all selected senders, or a "wildcard" that implicitly selects all the senders to the session. In an explicit sender-selection reservation, each filter spec must match exactly one sender, while in a wildcard sender-selection no filter spec is needed.



Motorola Solutions, Inc., Ex1006, p. 11

| Sender Selection | Reservations: Distinct | Reservations: Shared |
|---|---|---|
| Explicit | Fixed-Filter (FF) style | Shared-Explicit (SE) Style |
| Wildcard | (None defined) | Wildcard-Filter (WF) Style |

Figure 3: Reservation Attributes and Styles

The following styles are currently defined (see Figure 3):

o Wildcard-Filter (WF) Style

The WF style implies the options: "shared" reservation and "wildcard" sender selection. Thus, a WF-style reservation creates a single reservation shared by flows from all upstream senders. This reservation may be thought of as a shared "pipe", whose "size" is the largest of the resource requests from all receivers, independent of the number of senders using it. A WF-style reservation is propagated upstream towards all sender hosts, and it automatically extends to new senders as they appear.

Symbolically, we can represent a WF-style reservation request by:

WF( * {Q})

where the asterisk represents wildcard sender selection and Q represents the flowspec.

o Fixed-Filter (FF) Style

The FF style implies the options: "distinct" reservations and "explicit" sender selection. Thus, an elementary FF-style reservation request creates a distinct reservation for data packets from a particular sender, not sharing them with other senders' packets for the same session.



Motorola Solutions, Inc., Ex1006, p. 12

Symbolically, we can represent an elementary FF reservation request by:

```
FF( S{Q})
```

where S is the selected sender and Q is the corresponding flowspec; the pair forms a flow descriptor. RSVP allows multiple elementary FF-style reservations to be requested at the same time, using a list of flow descriptors:

```
FF( S1{Q1}, S2{Q2}, ...)
```

The total reservation on a link for a given session is the 'sum' of Q1, Q2, ... for all requested senders.

o Shared Explicit (SE) Style

The SE style implies the options: "shared" reservation and "explicit" sender selection. Thus, an SE-style reservation creates a single reservation shared by selected upstream senders. Unlike the WF style, the SE style allows a receiver to explicitly specify the set of senders to be included.

We can represent an SE reservation request containing a flowspec Q and a list of senders S1, S2, ... by:

```
SE( (S1,S2,...){Q} )
```

Shared reservations, created by WF and SE styles, are appropriate for those multicast applications in which multiple data sources are unlikely to transmit simultaneously. Packetized audio is an example of an application suitable for shared reservations; since a limited number of people talk at once, each receiver might issue a WF or SE reservation request for twice the bandwidth required for one sender (to allow some over-speaking). On the other hand, the FF style, which creates distinct reservations for the flows from different senders, is appropriate for video signals.

The RSVP rules disallow merging of shared reservations with distinct reservations, since these modes are fundamentally incompatible. They also disallow merging explicit sender selection with wildcard sender selection, since this might produce an unexpected service for a receiver that specified explicit selection. As a result of these prohibitions, WF, SE, and FF styles are all mutually incompatible.



Motorola Solutions, Inc., Ex1006, p. 13

It would seem possible to simulate the effect of a WF reservation using the SE style. When an application asked for WF, the RSVP process on the receiver host could use local state to create an equivalent SE reservation that explicitly listed all senders. However, an SE reservation forces the packet classifier in each node to explicitly select each sender in the list, while a WF allows the packet classifier to simply "wild card" the sender address and port. When there is a large list of senders, a WF style reservation can therefore result in considerably less overhead than an equivalent SE style reservation. For this reason, both SE and WF are included in the protocol.

Other reservation options and styles may be defined in the future.

### 1.4 Examples of Styles

This section presents examples of each of the reservation styles and shows the effects of merging.

Figure 4 illustrates a router with two incoming interfaces, labeled (a) and (b), through which flows will arrive, and two outgoing interfaces, labeled (c) and (d), through which data will be forwarded. 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 (R2 and R3) are routed via outgoing interface (c) ((d), respectively). We furthermore assume that outgoing interface (d) is connected to a broadcast LAN, i.e., that replication occurs in the network; R2 and R3 are reached via different next hop routers (not shown).

We must also specify the multicast routes within the node of Figure 4. Assume first that data packets from each Si shown in Figure 4 are routed to both outgoing interfaces. Under this assumption, Figures 5, 6, and 7 illustrate Wildcard-Filter, Fixed-Filter, and Shared-Explicit reservations, respectively.



Motorola Solutions, Inc., Ex1006, p. 14

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

Figure 4: Router Configuration

For simplicity, these examples show flowspecs as one-dimensional multiples of some base resource quantity B. The "Receives" column shows the RSVP reservation requests received over outgoing interfaces (c) and (d), and the "Reserves" column shows the resulting reservation state for each interface. The "Sends" column shows the reservation requests that are sent upstream to previous hops (a) and (b). In the "Reserves" column, each box represents one reserved "pipe" on the outgoing link, with the corresponding flow descriptor.

Figure 5, showing the WF style, illustrates two distinct situations in which merging is required. (1) Each of the two next hops on interface (d) results in a separate RSVP reservation request, as shown; these two requests must be merged into the effective flowspec, 3B, that is used to make the reservation on interface (d). (2) The reservations on the interfaces (c) and (d) must be merged in order to forward the reservation requests upstream; as a result, the larger flowspec 4B is forwarded upstream to each previous hop.



Motorola Solutions, Inc., Ex1006, p. 15

```
                          |
            Sends         |       Reserves            Receives
                          |
                          |       _______
      WF( *{4B} ) <- (a)  |  (c) | * {4B}|    (c) <- WF( *{4B} )
                          |      |_______|
                          |
    ----------------------|-----------------------------------------
                          |       _______
      WF( *{4B} ) <- (b)  |  (d) | * {3B}|    (d) <- WF( *{3B} )
                          |      |_______|        <- WF( *{2B} )
```

Figure 5: Wildcard-Filter (WF) Reservation Example

Figure 6 shows Fixed-Filter (FF) style reservations. For each outgoing interface, there is a separate reservation for each source that has been requested, but this reservation will be shared among all the receivers that made the request. The flow descriptors for senders S2 and S3, received through outgoing interfaces (c) and (d), are packed (not merged) into the request forwarded to previous hop (b). On the other hand, the three different flow descriptors specifying sender S1 are merged into the single request FF( S1{4B} ) that is sent to previous hop (a).

```
                        |
        Sends           |       Reserves            Receives
                        |
                        |       ________
  FF( S1{4B} ) <- (a)   |  (c) | S1{4B} |  (c) <- FF( S1{4B}, S2{5B} )
                        |      |________|
                        |      | S2{5B} |
                        |      |________|
  --------------------|-------------------------------------------
                        |       ________
              <- (b)    |  (d) | S1{3B} |  (d) <- FF( S1{3B}, S3{B} )
  FF( S2{5B}, S3{B} )   |      |________|      <- FF( S1{B} )
                        |      | S3{B}  |
                        |      |________|
```

Figure 6: Fixed-Filter (FF) Reservation Example



Motorola Solutions, Inc., Ex1006, p. 16

Figure 7 shows an example of Shared-Explicit (SE) style reservations. When SE-style reservations are merged, the resulting filter spec is the union of the original filter specs, and the resulting flowspec is the largest flowspec.

```
                         |
        Sends            |       Reserves             Receives
                         |
                         |       ________
SE( S1{3B} ) <- (a)      |  (c) |(S1,S2) |   (c) <- SE( (S1,S2){B} )
                         |      |   {B}  |
                         |      |________|
---------------------|-------------------------------------------
                         |      __________
             <- (b)      | (d) |(S1,S2,S3)|  (d) <- SE( (S1,S3){3B} )
SE( (S2,S3){3B} )        |     |   {3B}   |      <- SE( S2{2B} )
                         |     |__________|
```

Figure 7: Shared-Explicit (SE) Reservation Example

The three examples just shown assume that data packets from S1, S2, and S3 are routed to both outgoing interfaces. The top part of Figure 8 shows another routing assumption: data packets from S2 and S3 are not forwarded to interface (c), e.g., because the network topology provides a shorter path for these senders towards R1, not traversing this node. The bottom part of Figure 8 shows WF style reservations under this assumption. Since there is no route from (b) to (c), the reservation forwarded out interface (b) considers only the reservation on interface (d).



Motorola Solutions, Inc., Ex1006, p. 17

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

Router Configuration

```
                                |
            Sends               |       Reserves             Receives
                                |
                                |       _______
      WF( *{4B} ) <- (a)        |  (c) | * {4B}|   (c) <- WF( *{4B} )
                                |      |_______|
                                |
    ----------------------------|-----------------------------------------
                                |       _______
      WF( *{3B} ) <- (b)        |  (d) | * {3B}|   (d) <- WF( * {3B} )
                                |      |_______|       <- WF( * {2B} )
```

Figure 8: WF Reservation Example -- Partial Routing



Motorola Solutions, Inc., Ex1006, p. 18

## 2. RSVP Protocol Mechanisms

### 2.1 RSVP Messages

```
     Previous       Incoming           Outgoing             Next
     Hops           Interfaces         Interfaces           Hops

     _____          ____________________                    _____
    |     | data -->  |                    |  data -->    |     |
    |  A  |-----------| a                c |--------------|  C  |
    |_____| Path -->  |                    |  Path -->    |_____|
            <-- Resv  |                    |  <-- Resv     _____
     _____            |       ROUTER       |           |  |     |
    |     |  |        |                    |           |--|  D  |
    |  B  |--| data-->|                    |  data --> |  |_____|
    |_____|  |--------| b                d |-----------|
             | Path-->|                    |  Path --> |   _____
     _____   | <--Resv|____________________|  <-- Resv |  |     |
    |     |  |                                        |--|  D' |
    |  B' |--|                                        |  |_____|
    |_____|  |                                        |
```

Figure 9: Router Using RSVP

Figure 9 illustrates RSVP's model of a router node. Each data flow arrives from a "previous hop" through a corresponding "incoming interface" and departs through one or more "outgoing interface"(s). The same interface may act in both the incoming and outgoing roles for different data flows in the same session. Multiple previous hops and/or next hops may be reached through a given physical interface; for example, the figure implies that D and D' are connected to (d) with a broadcast LAN.

There are two fundamental RSVP message types: Resv and Path.

Each receiver host sends RSVP reservation request (Resv) messages upstream towards the senders. These messages must follow exactly the reverse of the path(s) the data packets will use, upstream to all the sender hosts included in the sender selection. They create and maintain "reservation state" in each node along the path(s). Resv messages must finally be delivered to the sender hosts themselves, so that the hosts can set up appropriate traffic control parameters for the first hop. The processing of Resv messages was discussed previously in Section 1.2.



Motorola Solutions, Inc., Ex1006, p. 19

Each RSVP sender host transmits RSVP "Path" messages downstream along the uni-/multicast routes provided by the routing protocol(s), following the paths of the data. These Path messages store "path state" in each node along the way. This path state includes at least the unicast IP address of the previous hop node, which is used to route the Resv messages hop-by-hop in the reverse direction. (In the future, some routing protocols may supply reverse path forwarding information directly, replacing the reverse-routing function of path state).

A Path message contains the following information in addition to the previous hop address:

o Sender Template

A Path message is required to carry a Sender Template, which describes the format of data packets that the sender will originate. This template is in the form of a filter spec that could be used to select this sender’s packets from others in the same session on the same link.

Sender Templates have exactly the same expressive power and format as filter specs that appear in Resv messages. Therefore a Sender Template may specify only the sender IP address and optionally the UDP/TCP sender port, and it assumes the protocol Id specified for the session.

o Sender Tspec

A Path message is required to carry a Sender Tspec, which defines the traffic characteristics of the data flow that the sender will generate. This Tspec is used by traffic control to prevent over-reservation, and perhaps unnecessary Admission Control failures.

o Adspec

A Path message may carry a package of OPWA advertising information, known as an "Adspec". An Adspec received in a Path message is passed to the local traffic control, which returns an updated Adspec; the updated version is then forwarded in Path messages sent downstream.



Motorola Solutions, Inc., Ex1006, p. 20

Path messages are sent with the same source and destination addresses as the data, so that they will be routed correctly through non-RSVP clouds (see Section 2.9). On the other hand, Resv messages are sent hop-by-hop; each RSVP-speaking node forwards a Resv message to the unicast address of a previous RSVP hop.

## 2.2 Merging Flowspecs

A Resv message forwarded to a previous hop carries a flowspec that is the "largest" of the flowspecs requested by the next hops to which the data flow will be sent (however, see Section 3.5 for a different merging rule used in certain cases). We say the flowspecs have been "merged". The examples shown in Section 1.4 illustrated another case of merging, when there are multiple reservation requests from different next hops for the same session and with the same filter spec, but RSVP should install only one reservation on that interface. Here again, the installed reservation should have an effective flowspec that is the "largest" of the flowspecs requested by the different next hops.

Since flowspecs are opaque to RSVP, the actual rules for comparing flowspecs must be defined and implemented outside RSVP proper. The comparison rules are defined in the appropriate integrated service specification document. An RSVP implementation will need to call service-specific routines to perform flowspec merging.

Note that flowspecs are generally multi-dimensional vectors; they may contain both Tspec and Rspec components, each of which may itself be multi-dimensional. Therefore, it may not be possible to strictly order two flowspecs. For example, if one request calls for a higher bandwidth and another calls for a tighter delay bound, one is not "larger" than the other. In such a case, instead of taking the larger, the service-specific merging routines must be able to return a third flowspec that is at least as large as each; mathematically, this is the "least upper bound" (LUB). In some cases, a flowspec at least as small is needed; this is the "greatest lower bound" (GLB) GLB (Greatest Lower Bound).

The following steps are used to calculate the effective flowspec (Re, Te) to be installed on an interface [RFC 2210]. Here Te is the effective Tspec and Re is the effective Rspec.



Motorola Solutions, Inc., Ex1006, p. 21

1. An effective flowspec is determined for the outgoing interface. Depending upon the link-layer technology, this may require merging flowspecs from different next hops; this means computing the effective flowspec as the LUB of the flowspecs. Note that what flowspecs to merge is determined by the link layer medium (see Section 3.11.2), while how to merge them is determined by the service model in use [RFC 2210].

   The result is a flowspec that is opaque to RSVP but actually consists of the pair (Re, Resv_Te), where is Re is the effective Rspec and Resv_Te is the effective Tspec.

2. A service-specific calculation of Path_Te, the sum of all Tspecs that were supplied in Path messages from different previous hops (e.g., some or all of A, B, and B' in Figure 9), is performed.

3. (Re, Resv_Te) and Path_Te are passed to traffic control. Traffic control will compute the effective flowspec as the "minimum" of Path_Te and Resv_Te, in a service-dependent manner.

Section 3.11.6 defines a generic set of service-specific calls to compare flowspecs, to compute the LUB and GLB of flowspecs, and to compare and sum Tspecs.

## 2.3 Soft State

RSVP takes a "soft state" approach to managing the reservation state in routers and hosts. RSVP soft state is created and periodically refreshed by Path and Resv messages. The state is deleted if no matching refresh messages arrive before the expiration of a "cleanup timeout" interval. State may also be deleted by an explicit "teardown" message, described in the next section. At the expiration of each "refresh timeout" period and after a state change, RSVP scans its state to build and forward Path and Resv refresh messages to succeeding hops.

Path and Resv messages are idempotent. When a route changes, the next Path message will initialize the path state on the new route, and future Resv messages will establish reservation state there; 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 at each node, depending upon the existence of state at that node.



Motorola Solutions, Inc., Ex1006, p. 22

RSVP sends its messages as IP datagrams with no reliability enhancement. Periodic transmission of refresh messages by hosts and routers is expected to handle the occasional loss of an RSVP message. If the effective cleanup timeout is set to K times the refresh timeout period, then RSVP can tolerate K-1 successive RSVP packet losses without falsely deleting state. The network traffic control mechanism should be statically configured to grant some minimal bandwidth for RSVP messages to protect them from congestion losses.

The state maintained by RSVP is dynamic; to change the set of senders Si or to change any QoS request, a host simply starts sending revised Path and/or Resv messages. The result will be an appropriate adjustment in the RSVP state in all nodes along the path; unused state will time out if it is not explicitly torn down.

In steady state, state is refreshed hop-by-hop to allow merging. When the received state differs from the stored state, the stored state is updated. If this update results in modification of state to be forwarded in refresh messages, these refresh messages must be generated and forwarded immediately, so that state changes can be propagated end-to-end without delay. However, propagation of a change stops when and if it reaches a point where merging causes no resulting state change. This minimizes RSVP control traffic due to changes and is essential for scaling to large multicast groups.

State that is received through a particular interface I* should never be forwarded out the same interface. Conversely, state that is forwarded out interface I* must be computed using only state that arrived on interfaces different from I*. A trivial example of this rule is illustrated in Figure 10, which shows a transit router with one sender and one receiver on each interface (and assumes one next/previous hop per interface). Interfaces (a) and (c) serve as both outgoing and incoming interfaces for this session. Both receivers are making wildcard-style reservations, in which the Resv messages are forwarded to all previous hops for senders in the group, with the exception of the next hop from which they came. The result is independent reservations in the two directions.

There is an additional rule governing the forwarding of Resv messages: state from Resv messages received from outgoing interface Io should be forwarded to incoming interface Ii only if Path messages from Ii are forwarded to Io.



Motorola Solutions, Inc., Ex1006, p. 23

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

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

Figure 10: Independent Reservations

## 2.4 Teardown

RSVP "teardown" messages remove path or reservation state immediately. Although it is not necessary to explicitly tear down an old reservation, we recommend that all end hosts send a teardown request as soon as an application finishes.

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 deletes path state, as well as all dependent reservation state, along the way. An ResvTear message deletes reservation state and travels towards all senders upstream from its point of initiation. A PathTear (ResvTear) message may be conceptualized as a reversed-sense Path message (Resv message, respectively).

A teardown request may be initiated either by an application in an end system (sender or receiver), or by a router as the result of state timeout or service preemption. Once initiated, a teardown request must be forwarded hop-by-hop without delay. A teardown message deletes the specified state in the node where it is received. As always, this state change will be propagated immediately to the next node, but only if there will be a net change after merging. As a result, a ResvTear message will prune the reservation state back (only) as far as possible.



Motorola Solutions, Inc., Ex1006, p. 24

Like all other RSVP messages, teardown requests are not delivered reliably. The loss of a teardown request message will not cause a protocol failure because the unused state will eventually time out even though it is not explicitly deleted. If a teardown message is lost, the router that failed to receive that message will time out its state and initiate a new teardown message beyond the loss point. Assuming that RSVP message loss probability is small, the longest time to delete state will seldom exceed one refresh timeout period.

It should be possible to tear down any subset of the established state. For path state, the granularity for teardown is a single sender. For reservation state, the granularity is an individual filter spec. For example, refer to Figure 7. Receiver R1 could send a ResvTear message for sender S2 only (or for any subset of the filter spec list), leaving S1 in place.

A ResvTear message specifies the style and filters; any flowspec is ignored. Whatever flowspec is in place will be removed if all its filter specs are torn down.

## 2.5 Errors

There are two RSVP error messages, ResvErr and PathErr. PathErr messages are very simple; they are simply sent upstream to the sender that created the error, and they do not change path state in the nodes though which they pass. There are only a few possible causes of path errors.

However, there are a number of ways for a syntactically valid reservation request to fail at some node along the path. A node may also decide to preempt an established reservation. The handling of ResvErr messages is somewhat complex (Section 3.5). Since a request that fails may be the result of merging a number of requests, a reservation error must be reported to all of the responsible receivers. In addition, merging heterogeneous requests creates a potential difficulty known as the "killer reservation" problem, in which one request could deny service to another. There are actually two killer-reservation problems.

1. The first killer reservation problem (KR-I) arises when there is already a reservation Q0 in place. If another receiver now makes a larger reservation Q1 > Q0, the result of merging Q0 and Q1 may be rejected by admission control in some upstream node. This must not deny service to Q0.



Motorola Solutions, Inc., Ex1006, p. 25

The solution to this problem is simple: when admission control fails for a reservation request, any existing reservation is left in place.

2. The second killer reservation problem (KR-II) is the converse: the receiver making a reservation Q1 is persistent even though Admission Control is failing for Q1 in some node. This must not prevent a different receiver from now establishing a smaller reservation Q0 that would succeed if not merged with Q1.

   To solve this problem, a ResvErr message establishes additional state, called "blockade state", in each node through which it passes. Blockade state in a node modifies the merging procedure to omit the offending flowspec (Q1 in the example) from the merge, allowing a smaller request to be forwarded and established. The Q1 reservation state is said to be "blockaded". Detailed rules are presented in Section 3.5.

A reservation request that fails Admission Control creates blockade state but is left in place in nodes downstream of the failure point. It has been suggested that these reservations downstream from the failure represent "wasted" reservations and should be timed out if not actively deleted. However, the downstream reservations are left in place, for the following reasons:

- There are two possible reasons for a receiver persisting in a failed reservation: (1) it is polling for resource availability along the entire path, or (2) it wants to obtain the desired QoS along as much of the path as possible. Certainly in the second case, and perhaps in the first case, the receiver will want to hold onto the reservations it has made downstream from the failure.

- If these downstream reservations were not retained, the responsiveness of RSVP to certain transient failures would be impaired. For example, suppose a route "flaps" to an alternate route that is congested, so an existing reservation suddenly fails, then quickly recovers to the original route. The blockade state in each downstream router must not remove the state or prevent its immediate refresh.

- If we did not refresh the downstream reservations, they might time out, to be restored every Tb seconds (where Tb is the blockade state timeout interval). Such intermittent behavior might be very distressing for users.



Motorola Solutions, Inc., Ex1006, p. 26

## 2.6 Confirmation

To request a confirmation for its reservation request, a receiver Rj includes in the Resv message a confirmation-request object containing Rj's IP address. At each merge point, only the largest flowspec and any accompanying confirmation-request object is forwarded upstream. If the reservation request from Rj is equal to or smaller than the reservation in place on a node, its Resv is not forwarded further, and if the Resv included a confirmation-request object, a ResvConf message is sent back to Rj. If the confirmation request is forwarded, it is forwarded immediately, and no more than once for each request.

This confirmation mechanism has the following consequences:

- o A new reservation request with a flowspec larger than any in place for a session will normally result in either a ResvErr or a ResvConf message back to the receiver from each sender. In this case, the ResvConf message will be an end-to-end confirmation.

- o The receipt of a ResvConf gives no guarantees. Assume the first two reservation requests from receivers R1 and R2 arrive at the node where they are merged. R2, whose reservation was the second to arrive at that node, may receive a ResvConf from that node while R1's request has not yet propagated all the way to a matching sender and may still fail. Thus, R2 may receive a ResvConf although there is no end-to-end reservation in place; furthermore, R2 may receive a ResvConf followed by a ResvErr.

## 2.7 Policy Control

RSVP-mediated QoS requests allow particular user(s) to obtain preferential access to network resources. To prevent abuse, some form of back pressure will generally be required on users who make reservations. For example, such back pressure may be accomplished by administrative access policies, or it may depend upon some form of user feedback such as real or virtual billing for the "cost" of a reservation. In any case, reliable user identification and selective admission will generally be needed when a reservation is requested.

The term "policy control" is used for the mechanisms required to support access policies and back pressure for RSVP reservations. When a new reservation is requested, each node must answer two questions: "Are enough resources available to meet this request?"



Motorola Solutions, Inc., Ex1006, p. 27

and "Is this user allowed to make this reservation?" These two decisions are termed the "admission control" decision and the "policy control" decision, respectively, and both must be favorable in order for RSVP to make a reservation. Different administrative domains in the Internet may have different reservation policies.

The input to policy control is referred to as "policy data", which RSVP carries in POLICY_DATA objects. Policy data may include credentials identifying users or user classes, account numbers, limits, quotas, etc. Like flowspecs, policy data is opaque to RSVP, which simply passes it to policy control when required. Similarly, merging of policy data must be done by the policy control mechanism rather than by RSVP itself. Note that the merge points for policy data are likely to be at the boundaries of administrative domains. It may therefore be necessary to carry accumulated and unmerged policy data upstream through multiple nodes before reaching one of these merge points.

Carrying user-provided policy data in Resv messages presents a potential scaling problem. When a multicast group has a large number of receivers, it will be impossible or undesirable to carry all receivers' policy data upstream. The policy data will have to be administratively merged at places near the receivers, to avoid excessive policy data. Further discussion of these issues and an example of a policy control scheme will be found in [PolArch96]. Specifications for the format of policy data objects and RSVP processing rules for them are under development.

## 2.8 Security

RSVP raises the following security issues.

- o Message integrity and node authentication

  Corrupted or spoofed reservation requests could lead to theft of service by unauthorized parties or to denial of service caused by locking up network resources. RSVP protects against such attacks with a hop-by-hop authentication mechanism using an encrypted hash function. The mechanism is supported by INTEGRITY objects that may appear in any RSVP message. These objects use a keyed cryptographic digest technique, which assumes that RSVP neighbors share a secret. Although this mechanism is part of the base RSVP specification, it is described in a companion document [Baker96].



Motorola Solutions, Inc., Ex1006, p. 28

Widespread use of the RSVP integrity mechanism will require the availability of the long-sought key management and distribution infrastructure for routers. Until that infrastructure becomes available, manual key management will be required to secure RSVP message integrity.

- User authentication

  Policy control will depend upon positive authentication of the user responsible for each reservation request. Policy data may therefore include cryptographically protected user certificates. Specification of such certificates is a future issue.

  Even without globally-verifiable user certificates, it may be possible to provide practical user authentication in many cases by establishing a chain of trust, using the hop-by-hop INTEGRITY mechanism described earlier.

- Secure data streams

  The first two security issues concerned RSVP's operation. A third security issue concerns resource reservations for secure data streams. In particular, the use of IPSEC (IP Security) in the data stream poses a problem for RSVP: if the transport and higher level headers are encrypted, RSVP's generalized port numbers cannot be used to define a session or a sender.

  To solve this problem, an RSVP extension has been defined in which the security association identifier (IPSEC SPI) plays a role roughly equivalent to the generalized ports [RFC 2207].

## 2.9 Non-RSVP Clouds

It is impossible to deploy RSVP (or any new protocol) at the same moment throughout the entire Internet. Furthermore, 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. Of course, an intermediate cloud that does not support RSVP is unable to perform resource reservation. However, if such a cloud has sufficient capacity, it may still provide useful realtime service.

RSVP is designed to operate correctly 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



Motorola Solutions, Inc., Ex1006, p. 29

unaffected by non-RSVP routers in the path. When a Path message traverses a non-RSVP cloud, it carries to the next RSVP-capable node the IP address of the last RSVP-capable router before entering the cloud. An Resv message is then forwarded directly to the next RSVP-capable router on the path(s) back towards the source.

Even though RSVP operates correctly through a non-RSVP cloud, the non-RSVP-capable nodes will in general perturb the QoS provided to a receiver. Therefore, RSVP passes a ‘NonRSVP’ flag bit to the local traffic control mechanism when there are non-RSVP-capable hops in the path to a given sender. Traffic control combines this flag bit with its own sources of information, and forwards the composed information on integrated service capability along the path to receivers using Adspecs [RFC 2210].

Some topologies of RSVP routers and non-RSVP routers can cause Resv messages to arrive at the wrong RSVP-capable node, or to arrive at the wrong interface of the correct node. An RSVP process must be prepared to handle either situation. If the destination address does not match any local interface and the message is not a Path or PathTear, the message must be forwarded without further processing by this node. To handle the wrong interface case, a "Logical Interface Handle" (LIH) is used. The previous hop information included in a Path message includes not only the IP address of the previous node but also an LIH defining the logical outgoing interface; both values are stored in the path state. A Resv message arriving at the addressed node carries both the IP address and the LIH of the correct outgoing interface, i.e, the interface that should receive the requested reservation, regardless of which interface it arrives on.

The LIH may also be useful when RSVP reservations are made over a complex link layer, to map between IP layer and link layer flow entities.

## 2.10 Host Model

Before a session can be created, the session identification (DestAddress, ProtocolId [, DstPort]) must be assigned and communicated to all the senders and receivers by some out-of-band mechanism. When an RSVP session is being set up, the following events happen at the end systems.



Motorola Solutions, Inc., Ex1006, p. 30

H1 A receiver joins the multicast group specified by DestAddress, using IGMP.

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

H3 A receiver application receives a Path message.

H4 A receiver starts sending appropriate Resv messages, specifying the desired flow descriptors.

H5 A sender application receives a Resv message.

H6 A sender starts sending data packets.

There are several synchronization considerations.

- o H1 and H2 may happen in either order.

- o Suppose that a new sender starts sending data (H6) but there are no multicast routes because no receivers have joined the group (H1). Then the data will be dropped at some router node (which node depends upon the routing protocol) until receivers(s) appear.

- o Suppose that a new sender starts sending Path messages (H2) and data (H6) simultaneously, 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. The sender could mitigate this problem by awaiting arrival of the first Resv message (H5); however, receivers that are farther away may not have reservations in place yet.

- o If a receiver starts sending Resv messages (H4) before receiving any Path messages (H3), 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 3.11.1 discusses the general requirements and outlines a generic interface.



Motorola Solutions, Inc., Ex1006, p. 31

# 3. RSVP Functional Specification

## 3.1 RSVP Message Formats

An RSVP message consists of a common header, followed by a body consisting of a variable number of variable-length, typed "objects". The following subsections define the formats of the common header, the standard object header, and each of the RSVP message types.

For each RSVP message type, there is a set of rules for the permissible choice of object types. These rules are specified using Backus-Naur Form (BNF) augmented with square brackets surrounding optional sub-sequences. The BNF implies an order for the objects in a message. However, in many (but not all) cases, object order makes no logical difference. An implementation should create messages with the objects in the order shown here, but accept the objects in any permissible order.

### 3.1.1 Common Header

```
         0             1              2             3
+-------------+-------------+-------------+-------------+
| Vers | Flags|  Msg Type   |       RSVP Checksum       |
+-------------+-------------+-------------+-------------+
|  Send_TTL   | (Reserved)  |        RSVP Length        |
+-------------+-------------+-------------+-------------+
```

The fields in the common header are as follows:

Vers: 4 bits

Protocol version number. This is version 1.

Flags: 4 bits

0x01-0x08: Reserved

No flag bits are defined yet.

Msg Type: 8 bits

1 = Path

2 = Resv



Motorola Solutions, Inc., Ex1006, p. 32

3 = PathErr

4 = ResvErr

5 = PathTear

6 = ResvTear

7 = ResvConf

RSVP Checksum: 16 bits

The one's complement of the one's complement sum of the message, with the checksum field replaced by zero for the purpose of computing the checksum. An all-zero value means that no checksum was transmitted.

Send_TTL: 8 bits

The IP TTL value with which the message was sent. See Section 3.8.

RSVP Length: 16 bits

The total length of this RSVP message in bytes, including the common header and the variable-length objects that follow.

### 3.1.2 Object Formats

Every object consists of one or more 32-bit words with a one-word header, with the following format:

```
       0             1              2             3
+-------------+-------------+-------------+-------------+
|       Length (bytes)      |  Class-Num  |   C-Type    |
+-------------+-------------+-------------+-------------+
|                                                       |
//                  (Object contents)                  //
|                                                       |
+-------------+-------------+-------------+-------------+
```



Motorola Solutions, Inc., Ex1006, p. 33

An object header has the following fields:

Length

A 16-bit field containing the total object length in bytes. Must always be a multiple of 4, and at least 4.

Class-Num

Identifies the object class; values of this field are defined in Appendix A. Each object class has a name, which is always capitalized in this document. An RSVP implementation must recognize the following classes:

NULL

A NULL object has a Class-Num of zero, and its C-Type is ignored. Its length must be at least 4, but can be any multiple of 4. A NULL object may appear anywhere in a sequence of objects, and its contents will be ignored by the receiver.

SESSION

Contains the IP destination address (DestAddress), the IP protocol id, and some form of generalized destination port, to define a specific session for the other objects that follow. Required in every RSVP message.

RSVP_HOP

Carries the IP address of the RSVP-capable node that sent this message and a logical outgoing interface handle (LIH; see Section 3.3). This document refers to a RSVP_HOP object as a PHOP ("previous hop") object for downstream messages or as a NHOP (" next hop") object for upstream messages.

TIME_VALUES

Contains the value for the refresh period R used by the creator of the message; see Section 3.7. Required in every Path and Resv message.



Motorola Solutions, Inc., Ex1006, p. 34

STYLE

Defines the reservation style plus style-specific information that is not in FLOWSPEC or FILTER_SPEC objects. Required in every Resv message.

FLOWSPEC

Defines a desired QoS, in a Resv message.

FILTER_SPEC

Defines a subset of session data packets that should receive the desired QoS (specified by a FLOWSPEC object), in a Resv message.

SENDER_TEMPLATE

Contains a sender IP address and perhaps some additional demultiplexing information to identify a sender. Required in a Path message.

SENDER_TSPEC

Defines the traffic characteristics of a sender's data flow. Required in a Path message.

ADSPEC

Carries OPWA data, in a Path message.

ERROR_SPEC

Specifies an error in a PathErr, ResvErr, or a confirmation in a ResvConf message.

POLICY_DATA

Carries information that will allow a local policy module to decide whether an associated reservation is administratively permitted. May appear in Path, Resv, PathErr, or ResvErr message.

The use of POLICY_DATA objects is not fully specified at this time; a future document will fill this gap.



Motorola Solutions, Inc., Ex1006, p. 35

INTEGRITY

Carries cryptographic data to authenticate the originating node and to verify the contents of this RSVP message. The use of the INTEGRITY object is described in [Baker96].

SCOPE

Carries an explicit list of sender hosts towards which the information in the message is to be forwarded. May appear in a Resv, ResvErr, or ResvTear message. See Section 3.4.

RESV_CONFIRM

Carries the IP address of a receiver that requested a confirmation. May appear in a Resv or ResvConf message.

C-Type

Object type, unique within Class-Num. Values are defined in Appendix A.

The maximum object content length is 65528 bytes. The Class-Num and C-Type fields may be used together as a 16-bit number to define a unique type for each object.

The high-order two bits of the Class-Num is used to determine what action a node should take if it does not recognize the Class-Num of an object; see Section 3.10.

### 3.1.3 Path Messages

Each sender host periodically sends a Path message for each data flow it originates. It contains a SENDER_TEMPLATE object defining the format of the data packets and a SENDER_TSPEC object specifying the traffic characteristics of the flow. Optionally, it may contain may be an ADSPEC object carrying advertising (OPWA) data for the flow.

A Path message travels from a sender to receiver(s) along the same path(s) used by the data packets. The IP source address of a Path message must be an address of the sender it describes, while the destination address must be the DestAddress for the session. These addresses assure that the message will be correctly routed through a non-RSVP cloud.



Motorola Solutions, Inc., Ex1006, p. 36

The format of a Path message is as follows:

```
<Path Message> ::= <Common Header> [ <INTEGRITY> ]

                          <SESSION> <RSVP_HOP>

                          <TIME_VALUES>

                         [ <POLICY_DATA> ... ]

                         [ <sender descriptor> ]

<sender descriptor> ::= <SENDER_TEMPLATE> <SENDER_TSPEC>

                         [ <ADSPEC> ]
```

If the INTEGRITY object is present, it must immediately follow the common header. There are no other requirements on transmission order, although the above order is recommended. Any number of POLICY_DATA objects may appear.

The PHOP (i.e., RSVP_HOP) object of each Path message contains the previous hop address, i.e., the IP address of the interface through which the Path message was most recently sent. It also carries a logical interface handle (LIH).

Each RSVP-capable node along the path(s) captures a Path message and processes it to create path state for the sender defined by the SENDER_TEMPLATE and SESSION objects. Any POLICY_DATA, SENDER_TSPEC, and ADSPEC objects are also saved in the path state. If an error is encountered while processing a Path message, a PathErr message is sent to the originating sender of the Path message. Path messages must satisfy the rules on SrcPort and DstPort in Section 3.2.

Periodically, the RSVP process at a node scans the path state to create new Path messages to forward towards the receiver(s). Each message contains a sender descriptor defining one sender, and carries the original sender’s IP address as its IP source address. Path messages eventually reach the applications on all receivers; however, they are not looped back to a receiver running in the same application process as the sender.

The RSVP process forwards Path messages and replicates them as required by multicast sessions, using routing information it obtains from the appropriate uni-/multicast routing process. The route depends upon the session DestAddress, and for some



Motorola Solutions, Inc., Ex1006, p. 37

routing protocols also upon the source (sender's IP) address. The routing information generally includes the list of zero or more outgoing interfaces to which the Path message is to be forwarded. Because each outgoing interface has a different IP address, the Path messages sent out different interfaces contain different PHOP addresses. In addition, ADSPEC objects carried in Path messages will also generally differ for different outgoing interfaces.

Path state for a given session and sender may not necessarily have a unique PHOP or unique incoming interface. There are two cases, corresponding to multicast and unicast sessions.

- o Multicast Sessions

  Multicast routing allows a stable distribution tree in which Path messages from the same sender arrive from more than one PHOP, and RSVP must be prepared to maintain all such path state. The RSVP rules for handling this situation are contained in Section 3.9. RSVP must not forward (according to the rules of Section 3.9) Path messages that arrive on an incoming interface different from that provided by routing.

- o Unicast Sessions

  For a short period following a unicast route change upstream, a node may receive Path messages from multiple PHOPs for a given (session, sender) pair. The node cannot reliably determine which is the right PHOP, although the node will receive data from only one of the PHOPs at a time. One implementation choice for RSVP is to ignore PHOP in matching unicast past state, and allow the PHOP to flip among the candidates. Another implementation choice is to maintain path state for each PHOP and to send Resv messages upstream towards all such PHOPs. In either case, the situation is a transient; the unused path state will time out or be torn down (because upstream path state timed out).

### 3.1.4 Resv Messages

Resv messages carry reservation requests hop-by-hop from receivers to senders, along the reverse paths of data flows for the session. The IP destination address of a Resv message is the unicast address of a previous-hop node, obtained from the path state. The IP source address is an address of the node that sent the message.



Motorola Solutions, Inc., Ex1006, p. 38

The Resv message format is as follows:

```
<Resv Message> ::= <Common Header> [ <INTEGRITY> ]

                        <SESSION>  <RSVP_HOP>

                        <TIME_VALUES>

                        [ <RESV_CONFIRM> ]  [ <SCOPE> ]

                        [ <POLICY_DATA> ... ]

                        <STYLE> <flow descriptor list>

<flow descriptor list> ::=  <empty> |

                  <flow descriptor list> <flow descriptor>
```

If the INTEGRITY object is present, it must immediately follow the common header. The STYLE object followed by the flow descriptor list must occur at the end of the message, and objects within the flow descriptor list must follow the BNF given below. There are no other requirements on transmission order, although the above order is recommended.

The NHOP (i.e., the RSVP_HOP) object contains the IP address of the interface through which the Resv message was sent and the LIH for the logical interface on which the reservation is required.

The appearance of a RESV_CONFIRM object signals a request for a reservation confirmation and carries the IP address of the receiver to which the ResvConf should be sent. Any number of POLICY_DATA objects may appear.

The BNF above defines a flow descriptor list as simply a list of flow descriptors. The following style-dependent rules specify in more detail the composition of a valid flow descriptor list for each of the reservation styles.

o WF Style:

```
<flow descriptor list> ::=  <WF flow descriptor>

<WF flow descriptor> ::= <FLOWSPEC>
```



Motorola Solutions, Inc., Ex1006, p. 39

o FF style:

```
<flow descriptor list> ::=

          <FLOWSPEC>  <FILTER_SPEC>  |

          <flow descriptor list> <FF flow descriptor>

<FF flow descriptor> ::=

          [ <FLOWSPEC> ] <FILTER_SPEC>
```

Each elementary FF style request is defined by a single (FLOWSPEC, FILTER_SPEC) pair, and multiple such requests may be packed into the flow descriptor list of a single Resv message. A FLOWSPEC object can be omitted if it is identical to the most recent such object that appeared in the list; the first FF flow descriptor must contain a FLOWSPEC.

o SE style:

```
<flow descriptor list> ::= <SE flow descriptor>

<SE flow descriptor> ::=

                        <FLOWSPEC> <filter spec list>

<filter spec list> ::=  <FILTER_SPEC>

                   |  <filter spec list> <FILTER_SPEC>
```

The reservation scope, i.e., the set of senders towards which a particular reservation is to be forwarded (after merging), is determined as follows:

o Explicit sender selection

The reservation is forwarded to all senders whose SENDER_TEMPLATE objects recorded in the path state match a FILTER_SPEC object in the reservation. This match must follow the rules of Section 3.2.



Motorola Solutions, Inc., Ex1006, p. 40

o Wildcard sender selection

A request with wildcard sender selection will match all senders that route to the given outgoing interface.

Whenever a Resv message with wildcard sender selection is forwarded to more than one previous hop, a SCOPE object must be included in the message (see Section 3.4 below); in this case, the scope for forwarding the reservation is constrained to just the sender IP addresses explicitly listed in the SCOPE object.

A Resv message that is forwarded by a node is generally the result of merging a set of incoming Resv messages (that are not blockaded; see Section 3.5). If one of these merged messages contains a RESV_CONFIRM object and has a FLOWSPEC larger than the FLOWSPECs of the other merged reservation requests, then this RESV_CONFIRM object is forwarded in the outgoing Resv message. A RESV_CONFIRM object in one of the other merged requests (whose flowspecs are equal to, smaller than, or incomparable to, the merged flowspec, and which is not blockaded) will trigger the generation of an ResvConf message containing the RESV_CONFIRM. A RESV_CONFIRM object in a request that is blockaded will be neither forwarded nor returned; it will be dropped in the current node.

### 3.1.5 Path Teardown Messages

Receipt of a PathTear (path teardown) message deletes matching path state. Matching state must have match the SESSION, SENDER_TEMPLATE, and PHOP objects. In addition, a PathTear message for a multicast session can only match path state for the incoming interface on which the PathTear arrived. If there is no matching path state, a PathTear message should be discarded and not forwarded.

PathTear messages are initiated explicitly by senders or by path state timeout in any node, and they travel downstream towards all receivers. A unicast PathTear must not be forwarded if there is path state for the same (session, sender) pair but a different PHOP. Forwarding of multicast PathTear messages is governed by the rules of Section 3.9.



Motorola Solutions, Inc., Ex1006, p. 41

A PathTear message must be routed exactly like the corresponding Path message. Therefore, its IP destination address must be the session DestAddress, and its IP source address must be the sender address from the path state being torn down.

```
<PathTear Message> ::= <Common Header> [ <INTEGRITY> ]

                            <SESSION> <RSVP_HOP>

                           [ <sender descriptor> ]

<sender descriptor> ::= (see earlier definition)
```

A PathTear message may include a SENDER_TSPEC or ADSPEC object in its sender descriptor, but these must be ignored. The order requirements are as given earlier for a Path message, but the above order is recommended.

Deletion of path state as the result of a PathTear message or a timeout must also adjust related reservation state as required to maintain consistency in the local node. The adjustment depends upon the reservation style. For example, suppose a PathTear deletes the path state for a sender S. If the style specifies explicit sender selection (FF or SE), any reservation with a filter spec matching S should be deleted; if the style has wildcard sender selection (WF), the reservation should be deleted if S is the last sender to the session. These reservation changes should not trigger an immediate Resv refresh message, since the PathTear message has already made the required changes upstream. They should not trigger a ResvErr message, since the result could be to generate a shower of such messages.

### 3.1.6 Resv Teardown Messages

Receipt of a ResvTear (reservation teardown) message deletes matching reservation state. Matching reservation state must match the SESSION, STYLE, and FILTER_SPEC objects as well as the LIH in the RSVP_HOP object. If there is no matching reservation state, a ResvTear message should be discarded. A ResvTear message may tear down any subset of the filter specs in FF-style or SE-style reservation state.

ResvTear messages are initiated explicitly by receivers or by any node in which reservation state has timed out, and they travel upstream towards all matching senders.



Motorola Solutions, Inc., Ex1006, p. 42

A ResvTear message must be routed like the corresponding Resv message, and its IP destination address will be the unicast address of a previous hop.

```
<ResvTear Message> ::= <Common Header> [<INTEGRITY>]

                       <SESSION> <RSVP_HOP>

                       [ <SCOPE> ] <STYLE>

                       <flow descriptor list>

<flow descriptor list> ::= (see earlier definition)
```

FLOWSPEC objects in the flow descriptor list of a ResvTear message will be ignored and may be omitted. The order requirements for INTEGRITY object, sender descriptor, STYLE object, and flow descriptor list are as given earlier for a Resv message, but the above order is recommended. A ResvTear message may include a SCOPE object, but it must be ignored.

A ResvTear message will cease to be forwarded at the node where merging would have suppressed forwarding of the corresponding Resv message. Depending upon the resulting state change in a node, receipt of a ResvTear message may cause a ResvTear message to be forwarded, a modified Resv message to be forwarded, or no message to be forwarded. These three cases can be illustrated in the case of the FF-style reservations shown in Figure 6.

- o If receiver R2 sends a ResvTear message for its reservation S3{B}, the corresponding reservation is removed from interface (d) and a ResvTear for S3{B} is forwarded out (b).

- o If receiver R1 sends a ResvTear for its reservation S1{4B}, the corresponding reservation is removed from interface (c) and a modified Resv message FF( S1{3B} ) is immediately forwarded out (a).

- o If receiver R3 sends a ResvTear message for S1{B}, there is no change in the effective reservation S1{3B} on (d) and no message is forwarded.



Motorola Solutions, Inc., Ex1006, p. 43

### 3.1.7 Path Error Messages

PathErr (path error) messages report errors in processing Path messages. They are travel upstream towards senders and are routed hop-by-hop using the path state. At each hop, the IP destination address is the unicast address of a previous hop. PathErr messages do not modify the state of any node through which they pass; they are only reported to the sender application.

```
<PathErr message> ::= <Common Header> [ <INTEGRITY> ]

                          <SESSION> <ERROR_SPEC>

                          [ <POLICY_DATA> ...]

                         [ <sender descriptor> ]

<sender descriptor> ::= (see earlier definition)
```

The ERROR_SPEC object specifies the error and includes the IP address of the node that detected the error (Error Node Address). One or more POLICY_DATA objects may be included message to provide relevant information. The sender descriptor is copied from the message in error. The object order requirements are as given earlier for a Path message, but the above order is recommended.

### 3.1.8 Resv Error Messages

ResvErr (reservation error) messages report errors in processing Resv messages, or they may report the spontaneous disruption of a reservation, e.g., by administrative preemption.

ResvErr messages travel downstream towards the appropriate receivers, routed hop-by-hop using the reservation state. At each hop, the IP destination address is the unicast address of a next-hop node.



Motorola Solutions, Inc., Ex1006, p. 44

```
<ResvErr Message> ::= <Common Header> [ <INTEGRITY> ]

                          <SESSION>  <RSVP_HOP>

                          <ERROR_SPEC>  [ <SCOPE> ]

                          [ <POLICY_DATA> ...]

                         <STYLE> [ <error flow descriptor> ]
```

The ERROR_SPEC object specifies the error and includes the IP address of the node that detected the error (Error Node Address). One or more POLICY_DATA objects may be included in an error message to provide relevant information (e.g.,, when a policy control error is being reported). The RSVP_HOP object contains the previous hop address, and the STYLE object is copied from the Resv message in error. The use of the SCOPE object in a ResvErr message is defined below in Section 3.4. The object order requirements are as given for Resv messages, but the above order is recommended.

The following style-dependent rules define the composition of a valid error flow descriptor; the object order requirements are as given earlier for flow descriptor.

o WF Style:

```
<error flow descriptor> ::= <WF flow descriptor>
```

o FF style:

```
<error flow descriptor> ::= <FF flow descriptor>
```

Each flow descriptor in a FF-style Resv message must be processed independently, and a separate ResvErr message must be generated for each one that is in error.

o SE style:

```
<error flow descriptor> ::= <SE flow descriptor>
```

An SE-style ResvErr message may list the subset of the filter specs in the corresponding Resv message to which the error applies.



Motorola Solutions, Inc., Ex1006, p. 45

Note that a ResvErr message contains only one flow descriptor. Therefore, a Resv message that contains N > 1 flow descriptors (FF style) may create up to N separate ResvErr messages.

Generally speaking, a ResvErr message should be forwarded towards all receivers that may have caused the error being reported. More specifically:

- o The node that detects an error in a reservation request sends a ResvErr message to the next hop node from which the erroneous reservation came.

  This ResvErr message must contain the information required to define the error and to route the error message in later hops. It therefore includes an ERROR_SPEC object, a copy of the STYLE object, and the appropriate error flow descriptor. If the error is an admission control failure while attempting to increase an existing reservation, then the existing reservation must be left in place and the InPlace flag bit must be on in the ERROR_SPEC of the ResvErr message.

- o Succeeding nodes forward the ResvErr message to next hops that have local reservation state. For reservations with wildcard scope, there is an additional limitation on forwarding ResvErr messages, to avoid loops; see Section 3.4. There is also a rule restricting the forwarding of a Resv message after an Admission Control failure; see Section 3.5.

  A ResvErr message that is forwarded should carry the FILTER_SPEC(s) from the corresponding reservation state.

- o When a ResvErr message reaches a receiver, the STYLE object, flow descriptor list, and ERROR_SPEC object (including its flags) should be delivered to the receiver application.

### 3.1.9 Confirmation Messages

ResvConf messages are sent to (probabilistically) acknowledge reservation requests. A ResvConf message is sent as the result of the appearance of a RESV_CONFIRM object in a Resv message.



Motorola Solutions, Inc., Ex1006, p. 46

A ResvConf message is sent to the unicast address of a receiver host; the address is obtained from the RESV_CONFIRM object. However, a ResvConf message is forwarded to the receiver hop-by-hop, to accommodate the hop-by-hop integrity check mechanism.

```
<ResvConf message> ::= <Common Header> [ <INTEGRITY> ]

                            <SESSION> <ERROR_SPEC>

                            <RESV_CONFIRM>

                            <STYLE> <flow descriptor list>

<flow descriptor list> ::= (see earlier definition)
```

The object order requirements are the same as those given earlier for a Resv message, but the above order is recommended.

The RESV_CONFIRM object is a copy of that object in the Resv message that triggered the confirmation. The ERROR_SPEC is used only to carry the IP address of the originating node, in the Error Node Address; the Error Code and Value are zero to indicate a confirmation. The flow descriptor list specifies the particular reservations that are being confirmed; it may be a subset of flow descriptor list of the Resv that requested the confirmation.

## 3.2 Port Usage

An RSVP session is normally defined by the triple: (DestAddress, ProtocolId, DstPort). Here DstPort is a UDP/TCP destination port field (i.e., a 16-bit quantity carried at octet offset +2 in the transport header). DstPort may be omitted (set to zero) if the ProtocolId specifies a protocol that does not have a destination port field in the format used by UDP and TCP.

RSVP allows any value for ProtocolId. However, end-system implementations of RSVP may know about certain values for this field, and in particular the values for UDP and TCP (17 and 6, respectively). An end system may give an error to an application that either:

- o specifies a non-zero DstPort for a protocol that does not have UDP/TCP-like ports, or



Motorola Solutions, Inc., Ex1006, p. 47

o specifies a zero DstPort for a protocol that does have UDP/TCP-like ports.

Filter specs and sender templates specify the pair: (SrcAddress, SrcPort), where SrcPort is a UDP/TCP source port field (i.e., a 16-bit quantity carried at octet offset +0 in the transport header). SrcPort may be omitted (set to zero) in certain cases.

The following rules hold for the use of zero DstPort and/or SrcPort fields in RSVP.

1. Destination ports must be consistent.

   Path state and reservation state for the same DestAddress and ProtocolId must each have DstPort values that are all zero or all non-zero. Violation of this condition in a node is a "Conflicting Dest Ports" error.

2. Destination ports rule.

   If DstPort in a session definition is zero, all SrcPort fields used for that session must also be zero. The assumption here is that the protocol does not have UDP/TCP-like ports. Violation of this condition in a node is a "Bad Src Ports" error.

3. Source Ports must be consistent.

   A sender host must not send path state both with and without a zero SrcPort. Violation of this condition is a "Conflicting Sender Port" error.

Note that RSVP has no "wildcard" ports, i.e., a zero port cannot match a non-zero port.

### 3.3 Sending RSVP Messages

RSVP messages are sent hop-by-hop between RSVP-capable routers as "raw" IP datagrams with protocol number 46. Raw IP datagrams are also intended to be used between an end system and the first/last hop router, although it is also possible to encapsulate RSVP messages as UDP datagrams for end-system communication, as described in Appendix C. UDP encapsulation is needed for systems that cannot do raw network I/O.



Motorola Solutions, Inc., Ex1006, p. 48

Path, PathTear, and ResvConf messages must be sent with the Router Alert IP option [RFC 2113] in their IP headers. This option may be used in the fast forwarding path of a high-speed router to detect datagrams that require special processing.

Upon the arrival of an RSVP message M that changes the state, a node must forward the state modification immediately. However, this must not trigger sending a message out the interface through which M arrived (which could happen if the implementation simply triggered an immediate refresh of all state for the session). This rule is necessary to prevent packet storms on broadcast LANs.

In this version of the spec, each RSVP message must occupy exactly one IP datagram. If it exceeds the MTU, such a datagram will be fragmented by IP and reassembled at the recipient node. This has several consequences:

- o A single RSVP message may not exceed the maximum IP datagram size, approximately 64K bytes.

- o A congested non-RSVP cloud could lose individual message fragments, and any lost fragment will lose the entire message.

Future versions of the protocol will provide solutions for these problems if they prove burdensome. The most likely direction will be to perform "semantic fragmentation", i.e., break the path or reservation state being transmitted into multiple self-contained messages, each of an acceptable size.

RSVP uses its periodic refresh mechanisms to recover from occasional packet losses. Under network overload, however, substantial losses of RSVP messages could cause a failure of resource reservations. To control the queuing delay and dropping of RSVP packets, routers should be configured to offer them a preferred class of service. If RSVP packets experience noticeable losses when crossing a congested non-RSVP cloud, a larger value can be used for the timeout factor K (see section 3.7).

Some multicast routing protocols provide for "multicast tunnels", which do IP encapsulation of multicast packets for transmission through routers that do not have multicast capability. A multicast tunnel looks like a logical outgoing interface that is mapped into some physical interface. A multicast routing protocol that supports tunnels will describe a route using a list of logical rather than physical interfaces. RSVP can operate across such multicast tunnels in the following manner:



Motorola Solutions, Inc., Ex1006, p. 49

1. When a node N forwards a Path message out a logical outgoing interface L, it includes in the message some encoding of the identity of L, called the "logical interface handle" or LIH. The LIH value is carried in the RSVP_HOP object.

2. The next hop node N' stores the LIH value in its path state.

3. When N' sends a Resv message to N, it includes the LIH value from the path state (again, in the RSVP_HOP object).

4. When the Resv message arrives at N, its LIH value provides the information necessary to attach the reservation to the appropriate logical interface. Note that N creates and interprets the LIH; it is an opaque value to N'.

Note that this only solves the routing problem posed by tunnels. The tunnel appears to RSVP as a non-RSVP cloud. To establish RSVP reservations within the tunnel, additional machinery will be required, to be defined in the future.

### 3.4 Avoiding RSVP Message Loops

Forwarding of RSVP messages must avoid looping. In steady state, Path and Resv messages are forwarded on each hop only once per refresh period. This avoids looping packets, but there is still the possibility of an "auto-refresh" loop, clocked by the refresh period. Such auto-refresh loops keep state active "forever", even if the end nodes have ceased refreshing it, until the receivers leave the multicast group and/or the senders stop sending Path messages. On the other hand, error and teardown messages are forwarded immediately and are therefore subject to direct looping.

Consider each message type.

- Path Messages

  Path messages are forwarded in exactly the same way as IP data packets. Therefore there should be no loops of Path messages (except perhaps for transient routing loops, which we ignore here), even in a topology with cycles.

- PathTear Messages

  PathTear messages use the same routing as Path messages and therefore cannot loop.



Motorola Solutions, Inc., Ex1006, p. 50

- PathErr Messages

  Since Path messages do not loop, they create path state defining a loop-free reverse path to each sender. PathErr messages are always directed to particular senders and therefore cannot loop.

- Resv Messages

  Resv messages directed to particular senders (i.e., with explicit sender selection) cannot loop. However, Resv messages with wildcard sender selection (WF style) have a potential for auto-refresh looping.

- ResvTear Messages

  Although ResvTear messages are routed the same as Resv messages, during the second pass around a loop there will be no state so any ResvTear message will be dropped. Hence there is no looping problem here.

- ResvErr Messages

  ResvErr messages for WF style reservations may loop for essentially the same reasons that Resv messages loop.

- ResvConf Messages

  ResvConf messages are forwarded towards a fixed unicast receiver address and cannot loop.

If the topology has no loops, then looping of Resv and ResvErr messages with wildcard sender selection can be avoided by simply enforcing the rule given earlier: state that is received through a particular interface must never be forwarded out the same interface. However, when the topology does have cycles, further effort is needed to prevent auto-refresh loops of wildcard Resv messages and fast loops of wildcard ResvErr messages. The solution to this problem adopted by this protocol specification is for such messages to carry an explicit sender address list in a SCOPE object.



Motorola Solutions, Inc., Ex1006, p. 51

When a Resv message with WF style is to be forwarded to a particular previous hop, a new SCOPE object is computed from the SCOPE objects that were received in matching Resv messages. If the computed SCOPE object is empty, the message is not forwarded to the previous hop; otherwise, the message is sent containing the new SCOPE object. The rules for computing a new SCOPE object for a Resv message are as follows:

1. The union is formed of the sets of sender IP addresses listed in all SCOPE objects in the reservation state for the given session.

   If reservation state from some NHOP does not contain a SCOPE object, a substitute sender list must be created and included in the union. For a message that arrived on outgoing interface OI, the substitute list is the set of senders that route to OI.

2. Any local senders (i.e., any sender applications on this node) are removed from this set.

3. If the SCOPE object is to be sent to PHOP, remove from the set any senders that did not come from PHOP.

Figure 11 shows an example of wildcard-scoped (WF style) Resv messages. The address lists within SCOPE objects are shown in square brackets. Note that there may be additional connections among the nodes, creating looping topology that is not shown.



Motorola Solutions, Inc., Ex1006, p. 52

```
                  _________________
              a |                 | c
R4, S4<----->|     Router      |<-----> R2, S2, S3
               |                 |
              b |                 |
R1, S1<----->|                 |
               |_________________|

Send on (a):            |    Receive on (c):
                        |
   <-- WF( [S4] )       |       <-- WF( [S4, S1])
                        |
Send on (b):            |
                        |
   <-- WF( [S1] )       |
                        |
Receive on (a):         |    Send on (c):
                        |
   WF( [S1,S2,S3]) --> |       WF( [S2, S3]) -->
                        |
Receive on (b):         |
                        |
   WF( [S2,S3,S4]) --> |
                        |
```

Figure 11: SCOPE Objects in Wildcard-Scope Reservations

SCOPE objects are not necessary if the multicast routing uses shared trees or if the reservation style has explicit sender selection. Furthermore, attaching a SCOPE object to a reservation should be deferred to a node which has more than one previous hop for the reservation state.

The following rules are used for SCOPE objects in ResvErr messages with WF style:

1. The node that detected the error initiates an ResvErr message containing a copy of the SCOPE object associated with the reservation state or message in error.

2. Suppose a wildcard-style ResvErr message arrives at a node with a SCOPE object containing the sender host address list L. The node forwards the ResvErr message using the rules of Section 3.1.8. However,



Motorola Solutions, Inc., Ex1006, p. 53

the ResvErr message forwarded out OI must contain a SCOPE object derived from L by including only those senders that route to OI. If this SCOPE object is empty, the ResvErr message should not be sent out OI.

### 3.5 Blockade State

The basic rule for creating a Resv refresh message is to merge the flowspecs of the reservation requests in place in the node, by computing their LUB. However, this rule is modified by the existence of "blockade state" resulting from ResvErr messages, to solve the KR-II problem (see Section 2.5). The blockade state also enters into the routing of ResvErr messages for Admission Control failure.

When a ResvErr message for an Admission Control failure is received, its flowspec Qe is used to create or refresh an element of local blockade state. Each element of blockade state consists of a blockade flowspec Qb taken from the flowspec of the ResvErr message, and an associated blockade timer Tb. When a blockade timer expires, the corresponding blockade state is deleted.

The granularity of blockade state depends upon the style of the ResvErr message that created it. For an explicit style, there may be a blockade state element (Qb(S),Tb(S)) for each sender S. For a wildcard style, blockade state is per previous hop P.

An element of blockade state with flowspec Qb is said to "blockade" a reservation with flowspec Qi if Qb is not (strictly) greater than Qi. For example, suppose that the LUB of two flowspecs is computed by taking the max of each of their corresponding components. Then Qb blockades Qi if for some component j, Qb[j] <= Qi[j].

Suppose that a node receives a ResvErr message from previous hop P (or, if style is explicit, sender S) as the result of an Admission Control failure upstream. Then:

1. An element of blockade state is created for P (or S) if it did not exist.

2. Qb(P) (or Qb(S)) is set equal to the flowspec Qe from the ResvErr message.

3. A corresponding blockade timer Tb(P) (or Tb(S)) is started or restarted for a time Kb*R. Here Kb is a fixed multiplier and R is the refresh interval for reservation state. Kb should be configurable.



Motorola Solutions, Inc., Ex1006, p. 54

4. If there is some local reservation state that is not blockaded (see below), an immediate reservation refresh for P (or S) is generated.

5. The ResvErr message is forwarded to next hops in the following way. If the InPlace bit is off, the ResvErr message is forwarded to all next hops for which there is reservation state. If the InPlace bit is on, the ResvErr message is forwarded only to the next hops whose Qi is blockaded by Qb.

Finally, we present the modified rule for merging flowspecs to create a reservation refresh message.

- If there are any local reservation requests Qi that are not blockaded, these are merged by computing their LUB. The blockaded reservations are ignored; this allows forwarding of a smaller reservation that has not failed and may perhaps succeed, after a larger reservation fails.

- Otherwise (all local requests Qi are blockaded), they are merged by taking the GLB (Greatest Lower Bound) of the Qi's.

  (The use of some definition of "minimum" improves performance by bracketing the failure level between the largest that succeeds and the smallest that fails. The choice of GLB in particular was made because it is simple to define and implement, and no reason is known for using a different definition of "minimum" here).

This refresh merging algorithm is applied separately to each flow (each sender or PHOP) contributing to a shared reservation (WF or SE style).

Figure 12 shows an example of the the application of blockade state for a shared reservation (WF style). There are two previous hops labeled (a) and (b), and two next hops labeled (c) and (d). The larger reservation 4B arrived from (c) first, but it failed somewhere upstream via PHOP (a), but not via PHOP (b). The figures show the final "steady state" after the smaller reservation 2B subsequently arrived from (d). This steady state is perturbed roughly every Kb*R seconds, when the blockade state times out. The next refresh then sends 4B to previous hop (a); presumably this will fail, sending a ResvErr message that will re-establish the blockade state, returning to the situation shown in the figure. At the same time, the ResvErr message will be forwarded to next hop (c) and to all receivers downstream responsible for the 4B reservations.



Motorola Solutions, Inc., Ex1006, p. 55

```
                Send     Blockade |   Reserve       Receive
                         State {Qb}|
                                   |   ________
       (a) <- WF(*{2B})    {4B} |  | * {4B} | WF(*{4B}) <- (c)
                                   |  |________|
                                   |
     --------------------------|-------------------------------
                                   |
                                   |   ________
       (b) <- WF(*{4B})   (none)|  | * {2B} | WF(*{2B}) <- (d)
                                   |  |________|
```

Figure 12: Blockading with Shared Style

### 3.6 Local Repair

When a route changes, the next Path or Resv refresh message will establish path or reservation state (respectively) along the new route. To provide fast adaptation to routing changes without the overhead of short refresh periods, the local routing protocol module can notify the RSVP process of route changes for particular destinations. The RSVP process should use this information to trigger a quick refresh of state for these destinations, using the new route.

The specific rules are as follows:

- When routing detects a change of the set of outgoing interfaces for destination G, RSVP should update the path state, wait for a short period W, and then send Path refreshes for all sessions G/* (i.e., for any session with destination G, regardless of destination port).

  The short wait period before sending Path refreshes is to allow the routing protocol to settle, and the value for W should be chosen accordingly. Currently W = 2 sec is suggested; however, this value should be configurable per interface.

- When a Path message arrives with a Previous Hop address that differs from the one stored in the path state, RSVP should send immediate Resv refreshes to that PHOP.



Motorola Solutions, Inc., Ex1006, p. 56

### 3.7 Time Parameters

There are two time parameters relevant to each element of RSVP path or reservation state in a node: the refresh period R between generation of successive refreshes for the state by the neighbor node, and the local state's lifetime L. Each RSVP Resv or Path message may contain a TIME_VALUES object specifying the R value that was used to generate this (refresh) message. This R value is then used to determine the value for L when the state is received and stored. The values for R and L may vary from hop to hop.

In more detail:

1. Floyd and Jacobson [FJ94] have shown that periodic messages generated by independent network nodes can become synchronized. This can lead to disruption in network services as the periodic messages contend with other network traffic for link and forwarding resources. Since RSVP sends periodic refresh messages, it must avoid message synchronization and ensure that any synchronization that may occur is not stable.

   For this reason, the refresh timer should be randomly set to a value in the range [0.5R, 1.5R].

2. To avoid premature loss of state, L must satisfy L >= (K + 0.5)*1.5*R, where K is a small integer. Then in the worst case, K-1 successive messages may be lost without state being deleted. To compute a lifetime L for a collection of state with different R values R0, R1, ..., replace R by max(Ri).

   Currently K = 3 is suggested as the default. However, it may be necessary to set a larger K value for hops with high loss rate. K may be set either by manual configuration per interface, or by some adaptive technique that has not yet been specified.

3. Each Path or Resv message carries a TIME_VALUES object containing the refresh time R used to generate refreshes. The recipient node uses this R to determine the lifetime L of the stored state created or refreshed by the message.

4. The refresh time R is chosen locally by each node. If the node does not implement local repair of reservations disrupted by route changes, a smaller R speeds up adaptation to routing changes, while increasing the RSVP overhead. With local repair, a router can be more relaxed about R since the periodic refresh becomes only a backstop robustness



Motorola Solutions, Inc., Ex1006, p. 57

mechanism. A node may therefore adjust the effective R dynamically to control the amount of overhead due to refresh messages.

The current suggested default for R is 30 seconds. However, the default value Rdef should be configurable per interface.

5. When R is changed dynamically, there is a limit on how fast it may increase. Specifically, the ratio of two successive values R2/R1 must not exceed 1 + Slew.Max.

   Currently, Slew.Max is 0.30. With K = 3, one packet may be lost without state timeout while R is increasing 30 percent per refresh cycle.

6. To improve robustness, a node may temporarily send refreshes more often than R after a state change (including initial state establishment).

7. The values of Rdef, K, and Slew.Max used in an implementation should be easily modifiable per interface, as experience may lead to different values. The possibility of dynamically adapting K and/or Slew.Max in response to measured loss rates is for future study.

### 3.8 Traffic Policing and Non-Integrated Service Hops

Some QoS services may require traffic policing at some or all of (1) the edge of the network, (2) a merging point for data from multiple senders, and/or (3) a branch point where traffic flow from upstream may be greater than the downstream reservation being requested. RSVP knows where such points occur and must so indicate to the traffic control mechanism. On the other hand, RSVP does not interpret the service embodied in the flowspec and therefore does not know whether policing will actually be applied in any particular case.

The RSVP process passes to traffic control a separate policing flag for each of these three situations.

- E_Police_Flag -- Entry Policing

  This flag is set in the first-hop RSVP node that implements traffic control (and is therefore capable of policing).

  For example, sender hosts must implement RSVP but currently many of them do not implement traffic control. In this case, the E_Police_Flag should be off in the sender host, and it



Motorola Solutions, Inc., Ex1006, p. 58

should only be set on when the first node capable of traffic control is reached. This is controlled by the E_Police flag in SESSION objects.

- o M_Police_Flag -- Merge Policing

  This flag should be set on for a reservation using a shared style (WF or SE) when flows from more than one sender are being merged.

- o B_Police_Flag -- Branch Policing

  This flag should be set on when the flowspec being installed is smaller than, or incomparable to, a FLOWSPEC in place on any other interface, for the same FILTER_SPEC and SESSION.

RSVP must also test for the presence of non-RSVP hops in the path and pass this information to traffic control. From this flag bit that the RSVP process supplies and from its own local knowledge, traffic control can detect the presence of a hop in the path that is not capable of QoS control, and it passes this information to the receivers in Adspecs [RFC 2210].

With normal IP forwarding, RSVP can detect a non-RSVP hop by comparing the IP TTL with which a Path message is sent to the TTL with which it is received; for this purpose, the transmission TTL is placed in the common header. However, the TTL is not always a reliable indicator of non-RSVP hops, and other means must sometimes be used. For example, if the routing protocol uses IP encapsulating tunnels, then the routing protocol must inform RSVP when non-RSVP hops are included. If no automatic mechanism will work, manual configuration will be required.

### 3.9 Multihomed Hosts

Accommodating multihomed hosts requires some special rules in RSVP. We use the term ‘multihomed host’ to cover both hosts (end systems) with more than one network interface and routers that are supporting local application programs.

An application executing on a multihomed host may explicitly specify which interface any given flow will use for sending and/or for receiving data packets, to override the system-specified default interface. The RSVP process must be aware of the default, and if an application sets a specific interface, it must also pass that information to RSVP.



Motorola Solutions, Inc., Ex1006, p. 59

- Sending Data

  A sender application uses an API call (SENDER in Section 3.11.1) to declare to RSVP the characteristics of the data flow it will originate. This call may optionally include the local IP address of the sender. If it is set by the application, this parameter must be the interface address for sending the data packets; otherwise, the system default interface is implied.

  The RSVP process on the host then sends Path messages for this application out the specified interface (only).

- Making Reservations

  A receiver application uses an API call (RESERVE in Section 3.11.1) to request a reservation from RSVP. This call may optionally include the local IP address of the receiver, i.e., the interface address for receiving data packets. In the case of multicast sessions, this is the interface on which the group has been joined. If the parameter is omitted, the system default interface is used.

  In general, the RSVP process should send Resv messages for an application out the specified interface. However, when the application is executing on a router and the session is multicast, a more complex situation arises. Suppose in this case that a receiver application joins the group on an interface Iapp that differs from Isp, the shortest-path interface to the sender. Then there are two possible ways for multicast routing to deliver data packets to the application. The RSVP process must determine which case holds by examining the path state, to decide which incoming interface to use for sending Resv messages.

  1. The multicast routing protocol may create a separate branch of the multicast distribution ‘tree’ to deliver to Iapp. In this case, there will be path state for both interfaces Isp and Iapp. The path state on Iapp should only match a reservation from the local application; it must be marked "Local_only" by the RSVP process. If "Local_only" path state for Iapp exists, the Resv message should be sent out Iapp.

     Note that it is possible for the path state blocks for Isp and Iapp to have the same next hop, if there is an intervening non-RSVP cloud.



Motorola Solutions, Inc., Ex1006, p. 60

2. The multicast routing protocol may forward data within the router from Isp to Iapp. In this case, Iapp will appear in the list of outgoing interfaces of the path state for Isp, and the Resv message should be sent out Isp.

3. When Path and PathTear messages are forwarded, path state marked "Local_Only" must be ignored.

### 3.10 Future Compatibility

We may expect that in the future new object C-Types will be defined for existing object classes, and perhaps new object classes will be defined. It will be desirable to employ such new objects within the Internet using older implementations that do not recognize them. Unfortunately, this is only possible to a limited degree with reasonable complexity. The rules are as follows (‘b’ represents a bit).

1. Unknown Class

   There are three possible ways that an RSVP implementation can treat an object with unknown class. This choice is determined by the two high-order bits of the Class-Num octet, as follows.

   o Class-Num = 0bbbbbbb

     The entire message should be rejected and an "Unknown Object Class" error returned.

   o Class-Num = 10bbbbbb

     The node should ignore the object, neither forwarding it nor sending an error message.

   o Class-Num = 11bbbbbb

     The node should ignore the object but forward it, unexamined and unmodified, in all messages resulting from this message.

   The following more detailed rules hold for unknown-class objects with a Class-Num of the form 11bbbbbb:

   1. Such unknown-class objects received in PathTear, ResvTear, PathErr, or ResvErr messages should be forwarded immediately in the same messages.



Motorola Solutions, Inc., Ex1006, p. 61

2. Such unknown-class objects received in Path or Resv messages should be saved with the corresponding state and forwarded in any refresh message resulting from that state.

3. When a Resv refresh is generated by merging multiple reservation requests, the refresh message should include the union of unknown-class objects from the component requests. Only one copy of each unique unknown-class object should be included in this union.

4. The original order of such unknown-class objects need not be retained; however, the message that is forwarded must obey the general order requirements for its message type.

Although objects with unknown class cannot be merged, these rules will forward such objects until they reach a node that knows how to merge them. Forwarding objects with unknown class enables incremental deployment of new objects; however, the scaling limitations of doing so must be carefully examined before a new object class is deployed with both high bits on.

2. Unknown C-Type for Known Class

One might expect the known Class-Num to provide information that could allow intelligent handling of such an object. However, in practice such class-dependent handling is complex, and in many cases it is not useful.

Generally, the appearance of an object with unknown C-Type should result in rejection of the entire message and generation of an error message (ResvErr or PathErr as appropriate). The error message will include the Class-Num and C-Type that failed (see Appendix B); the end system that originated the failed message may be able to use this information to retry the request using a different C-Type object, repeating this process until it runs out of alternatives or succeeds.

Objects of certain classes (FLOWSPEC, ADSPEC, and POLICY_DATA) are opaque to RSVP, which simply hands them to traffic control or policy modules. Depending upon its internal rules, either of the latter modules may reject a C-Type and inform the RSVP process; RSVP should then reject the message and send an error, as described in the previous paragraph.



Motorola Solutions, Inc., Ex1006, p. 62

## 3.11 RSVP Interfaces

RSVP on a router has interfaces to routing and to traffic control. RSVP on a host has an interface to applications (i.e, an API) and also an interface to traffic control (if it exists on the host).

### 3.11.1 Application/RSVP Interface

This section describes a generic interface between an application and 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.

o Register Session

```
Call: SESSION( DestAddress , ProtocolId, DstPort

           [ , SESSION_object ]

           [ , Upcall_Proc_addr ] )  -> Session-id
```

This call initiates RSVP processing for a session, defined by DestAddress together with ProtocolId and possibly a port number DstPort. If successful, the SESSION call returns immediately with a local session identifier Session-id, which may be used in subsequent calls.

The Upcall_Proc_addr parameter defines the address of an upcall procedure to receive asynchronous error or event notification; see below. The SESSION_object parameter is included as an escape mechanism to support some more general definition of the session ("generalized destination port"), should that be necessary in the future. Normally SESSION_object will be omitted.

o Define Sender

```
Call: SENDER( Session-id

           [ , Source_Address ]  [ , Source_Port ]

           [ , Sender_Template ]

           [ , Sender_Tspec ]    [ , Adspec ]

           [ , Data_TTL ]        [ , Policy_data ] )
```



Motorola Solutions, Inc., Ex1006, p. 63

A sender uses this call to define, or to modify the definition of, the attributes of the data flow. The first SENDER call for the session registered as ‘Session-id’ will cause RSVP to begin sending Path messages for this session; later calls will modify the path information.

The SENDER parameters are interpreted as follows:

- Source_Address

  This is the address of the interface from which the data will be sent. If it is omitted, a default interface will be used. This parameter is needed only on a multihomed sender host.

- Source_Port

  This is the UDP/TCP port from which the data will be sent.

- Sender_Template

  This parameter is included as an escape mechanism to support a more general definition of the sender ("generalized source port"). Normally this parameter may be omitted.

- Sender_Tspec

  This parameter describes the traffic flow to be sent; see [RFC 2210].

- Adspec

  This parameter may be specified to initialize the computation of QoS properties along the path; see [RFC 2210].

- Data_TTL

  This is the (non-default) IP Time-To-Live parameter that is being supplied on the data packets. It is needed to ensure that Path messages do not have a scope larger than multicast data packets.



Motorola Solutions, Inc., Ex1006, p. 64

- Policy_data

  This optional parameter passes policy data for the sender. This data may be supplied by a system service, with the application treating it as opaque.

o Reserve

```
Call: RESERVE( session-id, [ receiver_address , ]

          [ CONF_flag, ] [ Policy_data, ]

           style, style-dependent-parms )
```

A receiver uses this call to make or to modify a resource reservation for the session registered as ‘session-id’. 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 existing reservations may result in admission control failures).

The optional ‘receiver_address’ parameter may be used by a receiver on a multihomed host (or router); it is the IP address of one of the node’s interfaces. The CONF_flag should be set on if a reservation confirmation is desired, off otherwise. The ‘Policy_data’ parameter specifies policy data for the receiver, while the ‘style’ parameter indicates the reservation style. The rest of the parameters depend upon the style; generally these will be appropriate flowspecs and filter specs.

The RESERVE call returns immediately. Following a RESERVE call, an asynchronous ERROR/EVENT upcall may occur at any time.

o Release

```
Call: RELEASE( session-id )
```

This call removes RSVP state for the session specified by session-id. The node then sends appropriate teardown messages and ceases sending refreshes for this session-id.



Motorola Solutions, Inc., Ex1006, p. 65

o Error/Event Upcalls

The general form of a upcall is as follows:

```
Upcall: <Upcall_Proc>( ) -> session-id, Info_type,

              information_parameters
```

Here "Upcall_Proc" represents the upcall procedure whose address was supplied in the SESSION call. This upcall may occur asynchronously at any time after a SESSION call and before a RELEASE call, to indicate an error or an event.

Currently there are five upcall types, distinguished by the Info_type parameter. The selection of information parameters depends upon the type.

1. Info_type = PATH_EVENT

   A Path Event upcall results from receipt of the first Path message for this session, indicating to a receiver application that there is at least one active sender, or if the path state changes.

   ```
   Upcall: <Upcall_Proc>( ) -> session-id,

                Info_type=PATH_EVENT,

                Sender_Tspec, Sender_Template

                [ , Adspec ] [ , Policy_data ]
   ```

   This upcall presents the Sender_Tspec, the Sender_Template, the Adspec, and any policy data from a Path message.

2. Info_type = RESV_EVENT

   A Resv Event upcall is triggered by the receipt of the first RESV message, or by modification of a previous reservation state, for this session.



Motorola Solutions, Inc., Ex1006, p. 66

```
Upcall: <Upcall_Proc>( ) -> session-id,

            Info_type=RESV_EVENT,

            Style, Flowspec, Filter_Spec_list

            [ , Policy_data ]
```

Here 'Flowspec' will be the effective QoS that has been received. Note that an FF-style Resv message may result in multiple RESV_EVENT upcalls, one for each flow descriptor.

3. Info_type = PATH_ERROR

An Path Error event indicates an error in sender information that was specified in a SENDER call.

```
Upcall: <Upcall_Proc>( ) -> session-id,

              Info_type=PATH_ERROR,

              Error_code , Error_value ,

              Error_Node , Sender_Template

              [ , Policy_data_list ]
```

The Error_code parameter will define the error, and Error_value may supply some additional (perhaps system-specific) data about the error. The Error_Node parameter will specify the IP address of the node that detected the error. The Policy_data_list parameter, if present, will contain any POLICY_DATA objects from the failed Path message.

4. Info_type = RESV_ERR

An Resv Error event indicates an error in a reservation message to which this application contributed.

```
Upcall: <Upcall_Proc>( ) -> session-id,

              Info_type=RESV_ERROR,
```



Motorola Solutions, Inc., Ex1006, p. 67

```
                    Error_code , Error_value ,

                    Error_Node , Error_flags ,

                    Flowspec, Filter_spec_list

                    [ , Policy_data_list ]
```

The Error_code parameter will define the error and Error_value may supply some additional (perhaps system-specific) data. The Error_Node parameter will specify the IP address of the node that detected the event being reported.

There are two Error_flags:

- InPlace

  This flag may be on for an Admission Control failure, to indicate that there was, and is, a reservation in place at the failure node. This flag is set at the failure point and forwarded in ResvErr messages.

- NotGuilty

  This flag may be on for an Admission Control failure, to indicate that the flowspec requested by this receiver was strictly less than the flowspec that got the error. This flag is set at the receiver API.

Filter_spec_list and Flowspec will contain the corresponding objects from the error flow descriptor (see Section 3.1.8). List_count will specify the number of FILTER_SPECS in Filter_spec_list. The Policy_data_list parameter will contain any POLICY_DATA objects from the ResvErr message.

5. Info_type = RESV_CONFIRM

   A Confirmation event indicates that a ResvConf message was received.

   Upcall: <Upcall_Proc>( ) -> session-id,

```
                    Info_type=RESV_CONFIRM,
```



Motorola Solutions, Inc., Ex1006, p. 68

Style, List_count,

Flowspec, Filter_spec_list

[ , Policy_data ]

The parameters are interpreted as in the Resv Error upcall.

Although RSVP messages indicating path or resv events may be received periodically, the API should make the corresponding asynchronous upcall to the application only on the first occurrence or when the information to be reported changes. All error and confirmation events should be reported to the application.

### 3.11.2 RSVP/Traffic Control Interface

It is difficult to present a generic interface to traffic control, because the details of establishing a reservation depend strongly upon the particular link layer technology in use on an interface.

Merging of RSVP reservations is required because of multicast data delivery, which replicates data packets for delivery to different next-hop nodes. At each such replication point, RSVP must merge reservation requests from the corresponding next hops by computing the "maximum" of their flowspecs. At a given router or host, one or more of the following three replication locations may be in use.

1. IP layer

   IP multicast forwarding performs replication in the IP layer. In this case, RSVP must merge the reservations that are in place on the corresponding outgoing interfaces in order to forward a request upstream.

2. "The network"

   Replication might take place downstream from the node, e.g., in a broadcast LAN, in link-layer switches, or in a mesh of non-RSVP-capable routers (see Section 2.8). In



Motorola Solutions, Inc., Ex1006, p. 69

these cases, RSVP must merge the reservations from the different next hops in order to make the reservation on the single outgoing interface. It must also merge reservations requests from all outgoing interfaces in order to forward a request upstream.

3. Link-layer driver

For a multi-access technology, replication may occur in the link layer driver or interface card. For example, this case might arise when there is a separate ATM point-to-point VC towards each next hop. RSVP may need to apply traffic control independently to each VC, without merging requests from different next hops.

In general, these complexities do not impact the protocol processing that is required by RSVP, except to determine exactly what reservation requests need to be merged. It may be desirable to organize an RSVP implementation into two parts: a core that performs link-layer-independent processing, and a link-layer-dependent adaptation layer. However, we present here a generic interface that assumes that replication can occur only at the IP layer or in "the network".

o Make a Reservation

```
Call: TC_AddFlowspec( Interface, TC_Flowspec,

                      TC_Tspec, TC_Adspec, Police_Flags )

                              -> RHandle [, Fwd_Flowspec]
```

The TC_Flowspec parameter defines the desired effective QoS to admission control; its value is computed as the maximum over the flowspecs of different next hops (see the Compare_Flowspecs call below). The TC_Tspec parameter defines the effective sender Tspec Path_Te (see Section 2.2). The TC_Adspec parameter defines the effective Adspec. The Police_Flags parameter carries the three flags E_Police_Flag, M_Police_Flag, and B_Police_Flag; see Section 3.8.

If this call is successful, it establishes a new reservation channel corresponding to RHandle; otherwise, it returns an error code. The opaque number RHandle is used by the caller for subsequent references to this reservation. If the traffic control service updates the



Motorola Solutions, Inc., Ex1006, p. 70

flowspec, the call will also return the updated object as Fwd_Flowspec.

o Modify Reservation

```
Call: TC_ModFlowspec( Interface, RHandle, TC_Flowspec,

                      TC_Tspec, TC_Adspec, Police_flags )

                            [ -> Fwd_Flowspec ]
```

This call is used to modify an existing reservation. TC_Flowspec is passed to Admission Control; if it is rejected, the current flowspec is left in force. The corresponding filter specs, if any, are not affected. The other parameters are defined as in TC_AddFlowspec. If the service updates the flowspec, the call will also return the updated object as Fwd_Flowspec.

o Delete Flowspec

```
Call: TC_DelFlowspec( Interface, RHandle )
```

This call will delete an existing reservation, including the flowspec and all associated filter specs.

o Add Filter Spec

```
Call: TC_AddFilter( Interface, RHandle,

                 Session , FilterSpec ) -> FHandle
```

This call is used to associate an additional filter spec with the reservation specified by the given RHandle, following a successful TC_AddFlowspec call. This call returns a filter handle FHandle.

o Delete Filter Spec

```
Call: TC_DelFilter( Interface, FHandle )
```

This call is used to remove a specific filter, specified by FHandle.



Motorola Solutions, Inc., Ex1006, p. 71

o OPWA Update

Call: TC_Advertise( Interface, Adspec,

Non_RSVP_Hop_flag ) -> New_Adspec

This call is used for OPWA to compute the outgoing advertisement New_Adspec for a specified interface. The flag bit Non_RSVP_Hop_flag should be set whenever the RSVP daemon detects that the previous RSVP hop included one or more non-RSVP-capable routers. TC_Advertise will insert this information into New_Adspec to indicate that a non-integrated-service hop was found; see Section 3.8.

o Preemption Upcall

Upcall: TC_Preempt() -> RHandle, Reason_code

In order to grant a new reservation request, the admission control and/or policy control modules may preempt one or more existing reservations. This will trigger a TC_Preempt() upcall to RSVP for each preempted reservation, passing the RHandle of the reservation and a sub-code indicating the reason.

### 3.11.3 RSVP/Policy Control Interface

This interface will be specified in a future document.

### 3.11.4 RSVP/Routing Interface

An RSVP implementation needs the following support from the routing mechanisms of the node.

o Route Query

To forward Path and PathTear messages, an RSVP process must be able to query the routing process(s) for routes.

Ucast_Route_Query( [ SrcAddress, ] DestAddress,

Notify_flag ) -> OutInterface

Mcast_Route_Query( [ SrcAddress, ] DestAddress,



Motorola Solutions, Inc., Ex1006, p. 72

```
                                   Notify_flag )

                                -> [ IncInterface, ] OutInterface_list
```

Depending upon the routing protocol, the query may or may not depend upon SrcAddress, i.e., upon the sender host IP address, which is also the IP source address of the message. Here IncInterface is the interface through which the packet is expected to arrive; some multicast routing protocols may not provide it. If the Notify_flag is True, routing will save state necessary to issue unsolicited route change notification callbacks (see below) whenever the specified route changes.

A multicast route query may return an empty OutInterface_list if there are no receivers downstream of a particular router. A route query may also return a 'No such route' error, probably as a result of a transient inconsistency in the routing (since a Path or PathTear message for the requested route did arrive at this node). In either case, the local state should be updated as requested by the message, which cannot be forwarded further. Updating local state will make path state available immediately for a new local receiver, or it will tear down path state immediately.

o Route Change Notification

If requested by a route query with the Notify_flag True, the routing process may provide an asynchronous callback to the RSVP process that a specified route has changed.

```
   Ucast_Route_Change( ) -> [ SrcAddress, ] DestAddress,

                                     OutInterface

   Mcast_Route_Change( ) -> [ SrcAddress, ] DestAddress,

                   [ IncInterface, ] OutInterface_list
```

o Interface List Discovery

RSVP must be able to learn what real and virtual interfaces are active, with their IP addresses.



Motorola Solutions, Inc., Ex1006, p. 73

It should be possible to logically disable an interface for RSVP. When an interface is disabled for RSVP, a Path message should never be forwarded out that interface, and if an RSVP message is received on that interface, the message should be silently discarded (perhaps with local logging).

### 3.11.5 RSVP/Packet I/O Interface

An RSVP implementation needs the following support from the packet I/O and forwarding mechanisms of the node.

- o Promiscuous Receive Mode for RSVP Messages

  Packets received for IP protocol 46 but not addressed to the node must be diverted to the RSVP program for processing, without being forwarded. The RSVP messages to be diverted in this manner will include Path, PathTear, and ResvConf messages. These message types carry the Router Alert IP option, which can be used to pick them out of a high-speed forwarding path. Alternatively, the node can intercept all protocol 46 packets.

  On a router or multi-homed host, the identity of the interface (real or virtual) on which a diverted message is received, as well as the IP source address and IP TTL with which it arrived, must also be available to the RSVP process.

- o Outgoing Link Specification

  RSVP must be able to force a (multicast) datagram to be sent on a specific outgoing real or virtual link, bypassing the normal routing mechanism. A virtual link might be a multicast tunnel, for example. Outgoing link specification is necessary to send different versions of an outgoing Path message on different interfaces, and to avoid routing loops in some cases.

- o Source Address and TTL Specification

  RSVP must be able to specify the IP source address and IP TTL to be used when sending Path messages.

- o Router Alert

  RSVP must be able to cause Path, PathTear, and ResvConf message to be sent with the Router Alert IP option.



Motorola Solutions, Inc., Ex1006, p. 74

### 3.11.6 Service-Dependent Manipulations

Flowspecs, Tspecs, and Adspecs are opaque objects to RSVP; their contents are defined in service specification documents. In order to manipulate these objects, RSVP process must have available to it the following service-dependent routines.

- o Compare Flowspecs

```
Compare_Flowspecs( Flowspec_1, Flowspec_2 ) ->

                                              result_code
```

  The possible result_codes indicate: flowspecs are equal, Flowspec_1 is greater, Flowspec_2 is greater, flowspecs are incomparable but LUB can be computed, or flowspecs are incompatible.

  Note that comparing two flowspecs implicitly compares the Tspecs that are contained. Although the RSVP process cannot itself parse a flowspec to extract the Tspec, it can use the Compare_Flowspecs call to implicitly calculate Resv_Te (see Section 2.2).

- o Compute LUB of Flowspecs

```
LUB_of_Flowspecs( Flowspec_1, Flowspec_2 ) ->

                                         Flowspec_LUB
```

- o Compute GLB of Flowspecs

```
GLB_of_Flowspecs( Flowspec_1, Flowspec_2 ) ->

                                         Flowspec_GLB
```

- o Compare Tspecs

```
Compare_Tspecs( Tspec_1, Tspec_2 ) -> result_code
```



Motorola Solutions, Inc., Ex1006, p. 75

The possible result_codes indicate: Tspecs are equal, or Tspecs are unequal.

o Sum Tspecs

```
Sum_Tspecs( Tspec_1, Tspec_2 ) -> Tspec_sum
```

This call is used to compute Path_Te (see Section 2.2).

## 4. Acknowledgments

The design of RSVP is based upon research performed in 1992-1993 by a collaboration including Lixia Zhang (UCLA), Deborah Estrin (USC/ISI), Scott Shenker (Xerox PARC), Sugih Jamin (USC/Xerox PARC), and Daniel Zappala (USC). Sugih Jamin developed the first prototype implementation of RSVP and successfully demonstrated it in May 1993. Shai Herzog, and later Steve Berson, continued development of RSVP prototypes.

Since 1993, many members of the Internet research community have contributed to the design and development of RSVP; these include (in alphabetical order) Steve Berson, Bob Braden, Lee Breslau, Dave Clark, Deborah Estrin, Shai Herzog, Craig Partridge, Scott Shenker, John Wroclawski, Daniel Zappala, and Lixia Zhang. In addition, a number of host and router vendors have made valuable contributions to the RSVP documents, particularly Fred Baker (Cisco), Mark Baugher (Intel), Lou Berger (Fore Systems), Don Hoffman (Sun), Steve Jakowski (NetManage), John Krawczyk (Bay Networks), and Bill Nowicki (SGI), as well as many others.



Motorola Solutions, Inc., Ex1006, p. 76

# APPENDIX A. Object Definitions

C-Types are defined for the two Internet address families IPv4 and IPv6. To accommodate other address families, additional C-Types could easily be defined. These definitions are contained as an Appendix, to ease updating.

All unused fields should be sent as zero and ignored on receipt.

## A.1 SESSION Class

SESSION Class = 1.

o IPv4/UDP SESSION object: Class = 1, C-Type = 1

```
+-------------+-------------+-------------+-------------+
|             IPv4 DestAddress (4 bytes)                |
+-------------+-------------+-------------+-------------+
| Protocol Id |    Flags    |          DstPort          |
+-------------+-------------+-------------+-------------+
```

o IPv6/UDP SESSION object: Class = 1, C-Type = 2

```
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+               IPv6 DestAddress (16 bytes)             +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
| Protocol Id |     Flags   |          DstPort          |
+-------------+-------------+-------------+-------------+
```

DestAddress

The IP unicast or multicast destination address of the session. This field must be non-zero.

Protocol Id

The IP Protocol Identifier for the data flow. This field must be non-zero.



Motorola Solutions, Inc., Ex1006, p. 77

Flags

0x01 = E_Police flag

The E_Police flag is used in Path messages to determine the effective "edge" of the network, to control traffic policing. If the sender host is not itself capable of traffic policing, it will set this bit on in Path messages it sends. The first node whose RSVP is capable of traffic policing will do so (if appropriate to the service) and turn the flag off.

DstPort

The UDP/TCP destination port for the session. Zero may be used to indicate ‘none’.

Other SESSION C-Types could be defined in the future to support other demultiplexing conventions in the transport-layer or application layer.



Motorola Solutions, Inc., Ex1006, p. 78

## A.2 RSVP_HOP Class

RSVP_HOP class = 3.

o IPv4 RSVP_HOP object: Class = 3, C-Type = 1

```
+-------------+-------------+-------------+-------------+
|             IPv4 Next/Previous Hop Address             |
+-------------+-------------+-------------+-------------+
|                 Logical Interface Handle               |
+-------------+-------------+-------------+-------------+
```

o IPv6 RSVP_HOP object: Class = 3, C-Type = 2

```
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+             IPv6 Next/Previous Hop Address            +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
|                Logical Interface Handle               |
+-------------+-------------+-------------+-------------+
```

This object carries the IP address of the interface through which the last RSVP-knowledgeable hop forwarded this message. The Logical Interface Handle (LIH) is used to distinguish logical outgoing interfaces, as discussed in Sections 3.3 and 3.9. A node receiving an LIH in a Path message saves its value and returns it in the HOP objects of subsequent Resv messages sent to the node that originated the LIH. The LIH should be identically zero if there is no logical interface handle.



Motorola Solutions, Inc., Ex1006, p. 79

### A.3 INTEGRITY Class

INTEGRITY class = 4.

See [Baker96].

### A.4 TIME_VALUES Class

TIME_VALUES class = 5.

o TIME_VALUES Object: Class = 5, C-Type = 1

```
+-------------+-------------+-------------+-------------+
|                   Refresh Period R                    |
+-------------+-------------+-------------+-------------+
```

Refresh Period

The refresh timeout period R used to generate this message; in milliseconds.



Motorola Solutions, Inc., Ex1006, p. 80

## A.5 ERROR_SPEC Class

ERROR_SPEC class = 6.

o IPv4 ERROR_SPEC object: Class = 6, C-Type = 1

```
+-------------+-------------+-------------+-------------+
|            IPv4 Error Node Address (4 bytes)          |
+-------------+-------------+-------------+-------------+
|    Flags    |  Error Code |        Error Value        |
+-------------+-------------+-------------+-------------+
```

o IPv6 ERROR_SPEC object: Class = 6, C-Type = 2

```
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+           IPv6 Error Node Address (16 bytes)          +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
|    Flags    |  Error Code |        Error Value        |
+-------------+-------------+-------------+-------------+
```

Error Node Address

The IP address of the node in which the error was detected.

Flags

0x01 = InPlace

This flag is used only for an ERROR_SPEC object in a ResvErr message. If it on, this flag indicates that there was, and still is, a reservation in place at the failure point.

0x02 = NotGuilty

This flag is used only for an ERROR_SPEC object in a ResvErr message, and it is only set in the interface to



Motorola Solutions, Inc., Ex1006, p. 81

the receiver application. If it on, this flag indicates that the FLOWSPEC that failed was strictly greater than the FLOWSPEC requested by this receiver.

Error Code

A one-octet error description.

Error Value

A two-octet field containing additional information about the error. Its contents depend upon the Error Type.

The values for Error Code and Error Value are defined in Appendix B.



Motorola Solutions, Inc., Ex1006, p. 82

## A.6 SCOPE Class

SCOPE class = 7.

This object contains a list of IP addresses, used for routing messages with wildcard scope without loops. The addresses must be listed in ascending numerical order.

o IPv4 SCOPE List object: Class = 7, C-Type = 1

```
+-------------+-------------+-------------+-------------+
|                IPv4 Src Address (4 bytes)             |
+-------------+-------------+-------------+-------------+
//                                                      //
+-------------+-------------+-------------+-------------+
|                IPv4 Src Address (4 bytes)             |
+-------------+-------------+-------------+-------------+
```

o IPv6 SCOPE list object: Class = 7, C-Type = 2

```
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+                IPv6 Src Address (16 bytes)            +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
//                                                      //
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+                IPv6 Src Address (16 bytes)            +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
```



Motorola Solutions, Inc., Ex1006, p. 83

## A.7 STYLE Class

STYLE class = 8.

o STYLE object: Class = 8, C-Type = 1

```
+-------------+-------------+-------------+-------------+
|   Flags     |              Option Vector              |
+-------------+-------------+-------------+-------------+
```

Flags: 8 bits

(None assigned yet)

Option Vector: 24 bits

A set of bit fields giving values for the reservation options. If new options are added in the future, corresponding fields in the option vector will be assigned from the least-significant end. If a node does not recognize a style ID, it may interpret as much of the option vector as it can, ignoring new fields that may have been defined.

The option vector bits are assigned (from the left) as follows:

19 bits: Reserved

2 bits: Sharing control

00b: Reserved

01b: Distinct reservations

10b: Shared reservations

11b: Reserved

3 bits: Sender selection control

000b: Reserved

001b: Wildcard

010b: Explicit



Motorola Solutions, Inc., Ex1006, p. 84

011b - 111b: Reserved

The low order bits of the option vector are determined by the style, as follows:

WF 10001b
FF 01010b
SE 10010b



Motorola Solutions, Inc., Ex1006, p. 85

## A.8 FLOWSPEC Class

FLOWSPEC class = 9.

- o Reserved (obsolete) flowspec object: Class = 9, C-Type = 1

- o Inv-serv Flowspec object: Class = 9, C-Type = 2

  The contents and encoding rules for this object are specified in documents prepared by the int-serv working group [RFC 2210].



Motorola Solutions, Inc., Ex1006, p. 86

## A.9 FILTER_SPEC Class

FILTER_SPEC class = 10.

o IPv4 FILTER_SPEC object: Class = 10, C-Type = 1

```
+-------------+-------------+-------------+-------------+
|               IPv4 SrcAddress (4 bytes)               |
+-------------+-------------+-------------+-------------+
|    ######   |    ######   |          SrcPort          |
+-------------+-------------+-------------+-------------+
```

o IPv6 FILTER_SPEC object: Class = 10, C-Type = 2

```
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+               IPv6 SrcAddress (16 bytes)              +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
|    ######   |    ######   |          SrcPort          |
+-------------+-------------+-------------+-------------+
```

o IPv6 Flow-label FILTER_SPEC object: Class = 10, C-Type = 3

```
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+               IPv6 SrcAddress (16 bytes)              +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
|   #######   |         Flow Label (24 bits)            |
+-------------+-------------+-------------+-------------+
```

SrcAddress

The IP source address for a sender host. Must be non-zero.



Motorola Solutions, Inc., Ex1006, p. 87

SrcPort

The UDP/TCP source port for a sender, or zero to indicate 'none'.

Flow Label

A 24-bit Flow Label, defined in IPv6. This value may be used by the packet classifier to efficiently identify the packets belonging to a particular (sender->destination) data flow.



Motorola Solutions, Inc., Ex1006, p. 88

### A.10 SENDER_TEMPLATE Class

SENDER_TEMPLATE class = 11.

- o IPv4 SENDER_TEMPLATE object: Class = 11, C-Type = 1

  Definition same as IPv4/UDP FILTER_SPEC object.

- o IPv6 SENDER_TEMPLATE object: Class = 11, C-Type = 2

  Definition same as IPv6/UDP FILTER_SPEC object.

- o IPv6 Flow-label SENDER_TEMPLATE object: Class = 11, C-Type = 3

### A.11 SENDER_TSPEC Class

SENDER_TSPEC class = 12.

- o Intserv SENDER_TSPEC object: Class = 12, C-Type = 2

  The contents and encoding rules for this object are specified in documents prepared by the int-serv working group.

### A.12 ADSPEC Class

ADSPEC class = 13.

- o Intserv ADSPEC object: Class = 13, C-Type = 2

  The contents and format for this object are specified in documents prepared by the int-serv working group.



Motorola Solutions, Inc., Ex1006, p. 89

### A.13 POLICY_DATA Class

POLICY_DATA class = 14.

- o Type 1 POLICY_DATA object: Class = 14, C-Type = 1

  The contents of this object are for further study.



Motorola Solutions, Inc., Ex1006, p. 90

### A.14 Resv_CONFIRM Class

RESV_CONFIRM class = 15.

o IPv4 RESV_CONFIRM object: Class = 15, C-Type = 1

```
+-------------+-------------+-------------+-------------+
|            IPv4 Receiver Address (4 bytes)            |
+-------------+-------------+-------------+-------------+
```

o IPv6 RESV_CONFIRM object: Class = 15, C-Type = 2

```
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+            IPv6 Receiver Address (16 bytes)           +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
```



Motorola Solutions, Inc., Ex1006, p. 91

## APPENDIX B. Error Codes and Values

The following Error Codes may appear in ERROR_SPEC objects and be passed to end systems. Except where noted, these Error Codes may appear only in ResvErr messages.

o Error Code = 00: Confirmation

This code is reserved for use in the ERROR_SPEC object of a ResvConf message. The Error Value will also be zero.

o Error Code = 01: Admission Control failure

Reservation request was rejected by Admission Control due to unavailable resources.

For this Error Code, the 16 bits of the Error Value field are:

```
ssur cccc cccc cccc
```

where the bits are:

ss = 00: Low order 12 bits contain a globally-defined sub-code (values listed below).

ss = 10: Low order 12 bits contain a organization-specific sub-code. RSVP is not expected to be able to interpret this except as a numeric value.

ss = 11: Low order 12 bits contain a service-specific sub-code. RSVP is not expected to be able to interpret this except as a numeric value.

Since the traffic control mechanism might substitute a different service, this encoding may include some representation of the service in use.

u = 0: RSVP rejects the message without updating local state.

u = 1: RSVP may use message to update local state and forward the message. This means that the message is informational.



Motorola Solutions, Inc., Ex1006, p. 92

r: Reserved bit, should be zero.

cccc cccc cccc: 12 bit code.

The following globally-defined sub-codes may appear in the low-order 12 bits when ssur = 0000:

- Sub-code = 1: Delay bound cannot be met

- Sub-code = 2: Requested bandwidth unavailable

- Sub-code = 3: MTU in flowspec larger than interface MTU.

o Error Code = 02: Policy Control failure

Reservation or path message has been rejected for administrative reasons, for example, required credentials not submitted, insufficient quota or balance, or administrative preemption. This Error Code may appear in a PathErr or ResvErr message.

Contents of the Error Value field are to be determined in the future.

o Error Code = 03: No path information for this Resv message.

No path state for this session. Resv message cannot be forwarded.

o Error Code = 04: No sender information for this Resv message.

There is path state for this session, but it does not include the sender matching some flow descriptor contained in the Resv message. Resv message cannot be forwarded.

o Error Code = 05: Conflicting reservation style

Reservation style conflicts with style(s) of existing reservation state. The Error Value field contains the low-order 16 bits of the Option Vector of the existing style with which the conflict occurred. This Resv message cannot be forwarded.

o Error Code = 06: Unknown reservation style

Reservation style is unknown. This Resv message cannot be forwarded.



Motorola Solutions, Inc., Ex1006, p. 93

- Error Code = 07: Conflicting dest ports

  Sessions for same destination address and protocol have appeared with both zero and non-zero dest port fields. This Error Code may appear in a PathErr or ResvErr message.

- Error Code = 08: Conflicting sender ports

  Sender port is both zero and non-zero in Path messages for the same session. This Error Code may appear only in a PathErr message.

- Error Code = 09, 10, 11: (reserved)

- Error Code = 12: Service preempted

  The service request defined by the STYLE object and the flow descriptor has been administratively preempted.

  For this Error Code, the 16 bits of the Error Value field are:

```
ssur cccc cccc cccc
```

  Here the high-order bits ssur are as defined under Error Code 01. The globally-defined sub-codes that may appear in the low-order 12 bits when ssur = 0000 are to be defined in the future.

- Error Code = 13: Unknown object class

  Error Value contains 16-bit value composed of (Class-Num, C-Type) of unknown object. This error should be sent only if RSVP is going to reject the message, as determined by the high-order bits of the Class-Num. This Error Code may appear in a PathErr or ResvErr message.

- Error Code = 14: Unknown object C-Type

  Error Value contains 16-bit value composed of (Class-Num, C-Type) of object.

- Error Code = 15-19: (reserved)

- Error Code = 20: Reserved for API

  Error Value field contains an API error code, for an API error that was detected asynchronously and must be reported via an upcall.



Motorola Solutions, Inc., Ex1006, p. 94

o Error Code = 21: Traffic Control Error

Traffic Control call failed due to the format or contents of the parameters to the request. The Resv or Path message that caused the call cannot be forwarded, and repeating the call would be futile.

For this Error Code, the 16 bits of the Error Value field are:

```
ss00 cccc cccc cccc
```

Here the high-order bits ss are as defined under Error Code 01.

The following globally-defined sub-codes may appear in the low order 12 bits (cccc cccc cccc) when ss = 00:

- Sub-code = 01: Service conflict

  Trying to merge two incompatible service requests.

- Sub-code = 02: Service unsupported

  Traffic control can provide neither the requested service nor an acceptable replacement.

- Sub-code = 03: Bad Flowspec value

  Malformed or unreasonable request.

- Sub-code = 04: Bad Tspec value

  Malformed or unreasonable request.

- Sub-code = 05: Bad Adspec value

  Malformed or unreasonable request.

o Error Code = 22: Traffic Control System error

A system error was detected and reported by the traffic control modules. The Error Value will contain a system-specific value giving more information about the error. RSVP is not expected to be able to interpret this value.



Motorola Solutions, Inc., Ex1006, p. 95

- Error Code = 23: RSVP System error

  The Error Value field will provide implementation-dependent information on the error. RSVP is not expected to be able to interpret this value.

In general, every RSVP message is rebuilt at each hop, and the node that creates an RSVP message is responsible for its correct construction. Similarly, each node is required to verify the correct construction of each RSVP message it receives. Should a programming error allow an RSVP to create a malformed message, the error is not generally reported to end systems in an ERROR_SPEC object; instead, the error is simply logged locally, and perhaps reported through network management mechanisms.

The only message formatting errors that are reported to end systems are those that may reflect version mismatches, and which the end system might be able to circumvent, e.g., by falling back to a previous CType for an object; see code 13 and 14 above.

The choice of message formatting errors that an RSVP may detect and log locally is implementation-specific, but it will typically include the following:

- Wrong-length message: RSVP Length field does not match message length.

- Unknown or unsupported RSVP version.

- Bad RSVP checksum

- INTEGRITY failure

- Illegal RSVP message Type

- Illegal object length: not a multiple of 4, or less than 4.

- Next hop/Previous hop address in HOP object is illegal.

- Bad source port: Source port is non-zero in a filter spec or sender template for a session with destination port zero.

- Required object class (specify) missing

- Illegal object class (specify) in this message type.

- Violation of required object order



Motorola Solutions, Inc., Ex1006, p. 96

- Flow descriptor count wrong for style or message type
- Logical Interface Handle invalid
- Unknown object Class-Num.
- Destination address of ResvConf message does not match Receiver Address in the RESV_CONFIRM object it contains.



Motorola Solutions, Inc., Ex1006, p. 97

## APPENDIX C. UDP Encapsulation

An RSVP implementation will generally require the ability to perform "raw" network I/O, i.e., to send and receive IP datagrams using protocol 46. However, some important classes of host systems may not support raw network I/O. To use RSVP, such hosts must encapsulate RSVP messages in UDP.

The basic UDP encapsulation scheme makes two assumptions:

1. All hosts are capable of sending and receiving multicast packets if multicast destinations are to be supported.

2. The first/last-hop routers are RSVP-capable.

A method of relaxing the second assumption is given later.

Let Hu be a "UDP-only" host that requires UDP encapsulation, and Hr a host that can do raw network I/O. The UDP encapsulation scheme must allow RSVP interoperation among an arbitrary topology of Hr hosts, Hu hosts, and routers.

Resv, ResvErr, ResvTear, and PathErr messages are sent to unicast addresses learned from the path or reservation state in the node. If the node keeps track of which previous hops and which interfaces need UDP encapsulation, these messages can be sent using UDP encapsulation when necessary. On the other hand, Path and PathTear messages are sent to the destination address for the session, which may be unicast or multicast.

The tables in Figures 13 and 14 show the basic rules for UDP encapsulation of Path and PathTear messages, for unicast DestAddress and multicast DestAddress, respectively. The other message types, which are sent unicast, should follow the unicast rules in Figure 13. Under the ‘RSVP Send’ columns in these figures, the notation is ‘mode(destaddr, destport)’; destport is omitted for raw packets. The ‘Receive’ columns show the group that is joined and, where relevant, the UDP Listen port.

It is useful to define two flavors of UDP encapsulation, one to be sent by Hu and the other to be sent by Hr and R, to avoid double processing by the recipient. In practice, these two flavors are distinguished by differing UDP port numbers Pu and Pu’.



Motorola Solutions, Inc., Ex1006, p. 98

The following symbols are used in the tables.

- o D is the DestAddress for the particular session.
- o G* is a well-known group address of the form 224.0.0.14, i.e., a group that is limited to the local connected network.
- o Pu and Pu' are two well-known UDP ports for UDP encapsulation of RSVP, with values 1698 and 1699.
- o Ra is the IP address of the router interface 'a'.
- o Router interface 'a' is on the local network connected to Hu and Hr.
- o

The following notes apply to these figures:

[Note 1] Hu sends a unicast Path message either to the destination address D, if D is local, or to the address Ra of the first-hop router. Ra is presumably known to the host.

[Note 2] Here D is the address of the local interface through which the message arrived.

[Note 3] This assumes that the application has joined the group D.



Motorola Solutions, Inc., Ex1006, p. 99

UNICAST DESTINATION D:

| Node | RSVP Send | RSVP Receive |
|---|---|---|
| Hu | UDP(D/Ra,Pu) [Note 1] | UDP(D,Pu) and UDP(D,Pu') [Note 2] |
| Hr | Raw(D) and if (UDP) then UDP(D,Pu') | Raw() and UDP(D, Pu) [Note 2] (Ignore Pu') |
| R (Interface a): | Raw(D) and if (UDP) then UDP(D,Pu') | Raw() and UDP(Ra, Pu) (Ignore Pu') |

Figure 13: UDP Encapsulation Rules for Unicast Path and Resv Messages

MULTICAST DESTINATION D:

| Node | RSVP Send | RSVP Receive |
|---|---|---|
| Hu | UDP(G*,Pu) | UDP(D,Pu') [Note 3] and UDP(G*,Pu) |
| Hr | Raw(D,Tr) and if (UDP) then UDP(D,Pu') | Raw() and UDP(G*,Pu) (Ignore Pu') |
| R (Interface a): | Raw(D,Tr) and if (UDP) then UDP(D,Pu') | Raw() and UDP(G*,Pu) (Ignore Pu') |

Figure 14: UDP Encapsulation Rules for Multicast Path Messages



Motorola Solutions, Inc., Ex1006, p. 100

A router may determine if its interface X needs UDP encapsulation by listening for UDP-encapsulated Path messages that were sent to either G* (multicast D) or to the address of interface X (unicast D). There is one failure mode for this scheme: if no host on the connected network acts as an RSVP sender, there will be no Path messages to trigger UDP encapsulation. In this (unlikely) case, it will be necessary to explicitly configure UDP encapsulation on the local network interface of the router.

When a UDP-encapsulated packet is received, the IP TTL is not available to the application on most systems. The RSVP process that receives a UDP-encapsulated Path or PathTear message should therefore use the Send_TTL field of the RSVP common header as the effective receive TTL. This may be overridden by manual configuration.

We have assumed that the first-hop RSVP-capable router R is on the directly-connected network. There are several possible approaches if this is not the case.

1. Hu can send both unicast and multicast sessions to UDP(Ra,Pu) with TTL=Ta

   Here Ta must be the TTL to exactly reach R. If Ta is too small, the Path message will not reach R. If Ta is too large, R and succeeding routers may forward the UDP packet until its hop count expires. This will turn on UDP encapsulation between routers within the Internet, perhaps causing bogus UDP traffic. The host Hu must be explicitly configured with Ra and Ta.

2. A particular host on the LAN connected to Hu could be designated as an "RSVP relay host". A relay host would listen on (G*,Pu) and forward any Path messages directly to R, although it would not be in the data path. The relay host would have to be configured with Ra and Ta.



Motorola Solutions, Inc., Ex1006, p. 101

## APPENDIX D. Glossary

- o Admission control

  A traffic control function that decides whether the packet scheduler in the node can supply the requested QoS while continuing to provide the QoS requested by previously-admitted requests. See also "policy control" and "traffic control".

- o Adspec

  An Adspec is a data element (object) in a Path message that carries a package of OPWA advertising information. See "OPWA".

- o Auto-refresh loop

  An auto-refresh loop is an error condition that occurs when a topological loop of routers continues to refresh existing reservation state even though all receivers have stopped requesting these reservations. See section 3.4 for more information.

- o Blockade state

  Blockade state helps to solve a "killer reservation" problem. See sections 2.5 and 3.5, and "killer reservation".

- o Branch policing

  Traffic policing at a multicast branching point on an outgoing interface that has "less" resources reserved than another outgoing interface for the same flow. See "traffic policing".

- o C-Type

  The class type of an object; unique within class-name. See "class-name".

- o Class-name

  The class of an object. See "object".

- o DestAddress

  The IP destination address; part of session identification. See "session".



Motorola Solutions, Inc., Ex1006, p. 102

- Distinct style

  A (reservation) style attribute; separate resources are reserved for each different sender. See also "shared style".

- Downstream

  Towards the data receiver(s).

- DstPort

  The IP (generalized) destination port used as part of a session. See "generalized destination port".

- Entry policing

  Traffic policing done at the first RSVP- (and policing-) capable router on a data path.

- ERROR_SPEC

  Object that carries the error report in a PathErr or ResvErr message.

- Explicit sender selection

  A (reservation) style attribute; all reserved senders are to be listed explicitly in the reservation message. See also "wildcard sender selection".

- FF style

  Fixed Filter reservation style, which has explicit sender selection and distinct attributes.

- FilterSpec

  Together with the session information, defines the set of data packets to receive the QoS specified in a flowspec. The filterspec is used to set parameters in the packet classifier function. A filterspec may be carried in a FILTER_SPEC or SENDER_TEMPLATE object.

- Flow descriptor

  The combination of a flowspec and a filterspec.



Motorola Solutions, Inc., Ex1006, p. 103

- Flowspec

  Defines the QoS to be provided for a flow. The flowspec is used to set parameters in the packet scheduling function to provide the requested quality of service. A flowspec is carried in a FLOWSPEC object. The flowspec format is opaque to RSVP and is defined by the Integrated Services Working Group.

- Generalized destination port

  The component of a session definition that provides further transport or application protocol layer demultiplexing beyond DestAddress. See "session".

- Generalized source port

  The component of a filter spec that provides further transport or application protocol layer demultiplexing beyond the sender address.

- GLB

  Greatest Lower Bound

- Incoming interface

  The interface on which data packets are expected to arrive, and on which Resv messages are sent.

- INTEGRITY

  Object of an RSVP control message that contains cryptographic data to authenticate the originating node and to verify the contents of an RSVP message.

- Killer reservation problem

  The killer reservation problem describes a case where a receiver attempting and failing to make a large QoS reservation prevents smaller QoS reservations from being established. See Sections 2.5 and 3.5 for more information.

- LIH

  The LIH (Logical Interface Handle) is used to help deal with non-RSVP clouds. See Section 2.9 for more information.



Motorola Solutions, Inc., Ex1006, p. 104

- o Local repair

  Allows RSVP to rapidly adapt its reservations to changes in routing. See Section 3.6 for more information.

- o LPM

  Local Policy Module. the function that exerts policy control.

- o LUB

  Least Upper Bound.

- o Merge policing

  Traffic policing that takes place at data merge point of a shared reservation.

- o Merging

  The process of taking the maximum (or more generally the least upper bound) of the reservations arriving on outgoing interfaces, and forwarding this maximum on the incoming interface. See Section 2.2 for more information.

- o MTU

  Maximum Transmission Unit.

- o Next hop

  The next router in the direction of traffic flow.

- o NHOP

  An object that carries the Next Hop information in RSVP control messages.

- o Node

  A router or host system.

- o Non-RSVP clouds

  Groups of hosts and routers that do not run RSVP. Dealing with nodes that do not support RSVP is important for backwards compatibility. See section 2.9.



Motorola Solutions, Inc., Ex1006, p. 105

- o Object

  An element of an RSVP control message; a type, length, value triplet.

- o OPWA

  Abbreviation for "One Pass With Advertising". Describes a reservation setup model in which (Path) messages sent downstream gather information that the receiver(s) can use to predict the end-to-end service. The information that is gathered is called an advertisement. See also "Adspec".

- o Outgoing interface

  Interface through which data packets and Path messages are forwarded.

- o Packet classifier

  Traffic control function in the primary data packet forwarding path that selects a service class for each packet, in accordance with the reservation state set up by RSVP. The packet classifier may be combined with the routing function. See also "traffic control".

- o Packet scheduler

  Traffic control function in the primary data packet forwarding path that implements QoS for each flow, using one of the service models defined by the Integrated Services Working Group. See also " traffic control".

- o Path state

  Information kept in routers and hosts about all RSVP senders.

- o PathErr

  Path Error RSVP control message.

- o PathTear

  Path Teardown RSVP control message.



Motorola Solutions, Inc., Ex1006, p. 106

- o PHOP

  An object that carries the Previous Hop information in RSVP control messages.

- o Police

  See traffic policing.

- o Policy control

  A function that determines whether a new request for quality of service has administrative permission to make the requested reservation. Policy control may also perform accounting (usage feedback) for a reservation.

- o Policy data

  Data carried in a Path or Resv message and used as input to policy control to determine authorization and/or usage feedback for the given flow.

- o Previous hop

  The previous router in the direction of traffic flow. Resv messages flow towards previous hops.

- o ProtocolId

  The component of session identification that specifies the IP protocol number used by the data stream.

- o QoS

  Quality of Service.

- o Reservation state

  Information kept in RSVP-capable nodes about successful RSVP reservation requests.

- o Reservation style

  Describes a set of attributes for a reservation, including the sharing attributes and sender selection attributes. See Section 1.3 for details.



Motorola Solutions, Inc., Ex1006, p. 107

- o Resv message

  Reservation request RSVP control message.

- o ResvConf

  Reservation Confirmation RSVP control message, confirms successful installation of a reservation at some upstream node.

- o ResvErr

  Reservation Error control message, indicates that a reservation request has failed or an active reservation has been preempted.

- o ResvTear

  Reservation Teardown RSVP control message, deletes reservation state.

- o Rspec

  The component of a flowspec that defines a desired QoS. The Rspec format is opaque to RSVP and is defined by the Integrated Services Working Group of the IETF.

- o RSVP_HOP

  Object of an RSVP control message that carries the PHOP or NHOP address of the source of the message.

- o Scope

  The set of sender hosts to which a given reservation request is to be propagated.

- o SE style

  Shared Explicit reservation style, which has explicit sender selection and shared attributes.

- o Semantic fragmentation

  A method of fragmenting a large RSVP message using information about the structure and contents of the message, so that each fragment is a logically complete RSVP message.



Motorola Solutions, Inc., Ex1006, p. 108

- o Sender template

  Parameter in a Path message that defines a sender; carried in a SENDER_TEMPLATE object. It has the form of a filter spec that can be used to select this sender's packets from other packets in the same session on the same link.

- o Sender Tspec

  Parameter in a Path message, a Tspec that characterizes the traffic parameters for the data flow from the corresponding sender. It is carried in a SENDER_TSPEC object.

- o Session

  An RSVP session defines one simplex unicast or multicast data flow for which reservations are required. A session is identified by the destination address, transport-layer protocol, and an optional (generalized) destination port.

- o Shared style

  A (reservation) style attribute: all reserved senders share the same reserved resources. See also "distinct style".

- o Soft state

  Control state in hosts and routers that will expire if not refreshed within a specified amount of time.

- o STYLE

  Object of an RSVP message that specifies the desired reservation style.

- o Style

  See "reservation style"

- o TIME_VALUES

  Object in an RSVP control message that specifies the time period timer used for refreshing the state in this message.



Motorola Solutions, Inc., Ex1006, p. 109

- Traffic control

  The entire set of machinery in the node that supplies requested QoS to data streams. Traffic control includes packet classifier, packet scheduler, and admission control functions.

- Traffic policing

  The function, performed by traffic control, of forcing a given data flow into compliance with the traffic parameters implied by the reservation. It may involve dropping non-compliant packets or sending them with lower priority, for example.

- TSpec

  A traffic parameter set that describes a flow. The format of a Tspec is opaque to RSVP and is defined by the Integrated Service Working Group.

- UDP encapsulation

  A way for hosts that cannot use raw sockets to participate in RSVP by encapsulating the RSVP protocol (raw) packets in ordinary UDP packets. See Section APPENDIX C for more information.

- Upstream

  Towards the traffic source. RSVP Resv messages flow upstream.

- WF style

  Wildcard Filter reservation style, which has wildcard sender selection and shared attributes.

- Wildcard sender selection

  A (reservation) style attribute: traffic from any sender to a specific session receives the same QoS. See also "explicit sender selection".



Motorola Solutions, Inc., Ex1006, p. 110

## References

[Baker96] Baker, F., "RSVP Cryptographic Authentication", Work in Progress.

[RFC 1633] Braden, R., Clark, D., and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, ISI, MIT, and PARC, June 1994.

[FJ94] Floyd, S. and V. Jacobson, "Synchronization of Periodic Routing Messages", IEEE/ACM Transactions on Networking, Vol. 2, No. 2, April, 1994.

[RFC 2207] Berger, L. and T. O'Malley, "RSVP Extensions for IPSEC Data Flows", RFC 2207, September 1997.

[RFC 2113] Katz, D., "IP Router Alert Option", RFC 2113, cisco Systems, February 1997.

[RFC 2210] Wroclawski, J., "The Use of RSVP with Integrated Services", RFC 2210, September 1997.

[PolArch96] Herzog, S., "Policy Control for RSVP: Architectural Overview". Work in Progress.

[OPWA95] Shenker, S. and L. Breslau, "Two Issues in Reservation Establishment", Proc. ACM SIGCOMM '95, Cambridge, MA, August 1995.

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

## Security Considerations

See Section 2.8.



Motorola Solutions, Inc., Ex1006, p. 111

Authors' Addresses

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

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

Lixia Zhang
UCLA Computer Science Department
4531G Boelter Hall
Los Angeles, CA 90095-1596 USA

Phone: 310-825-2695
EMail: lixia@cs.ucla.edu

Steve Berson
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

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

Shai Herzog
IBM T. J. Watson Research Center
P.O Box 704
Yorktown Heights, NY 10598

Phone: (914) 784-6059
EMail: Herzog@WATSON.IBM.COM

Sugih Jamin
University of Michigan
CSE/EECS
1301 Beal Ave.
Ann Arbor, MI 48109-2122

Phone: (313) 763-1583

EMail: jamin@EECS.UMICH.EDU



Motorola Solutions, Inc., Ex1006, p. 112