DECLARATION OF SANDY GINOZA FOR IETF

<u>RFC 2961: RSVP Refresh Overhead Reduction Extensions</u>

<u>RFC 3175: Aggregation of RSVP for IPv4 and IPv6 Reservations</u>

I, Sandy Ginoza, hereby declare that all statements made herein are of my own knowledge and are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code:

1. I am an employee of Association Management Solutions, LLC (AMS), which acts under contract to the IETF Administration LLC (IETF) as the operator of the RFC Production Center. The RFC Production Center is part of the "RFC Editor" function, which prepares documents for publication and places files in an online repository for the authoritative Request for Comments (RFC) series of documents (RFC Series), and preserves records relating to these documents. The RFC Series includes, among other things, the series of Internet standards developed by the IETF. I hold the position of Director of the RFC Production Center. I began employment with AMS in this capacity on 6 January 2010.

2. Among my responsibilities as Director of the RFC Production Center, I act as the custodian of records relating to the RFC Series, and I am familiar with the record keeping practices relating to the RFC Series, including the creation and maintenance of such records.

3. From June 1999 to 5 January 2010, I was an employee of the Information Sciences Institute at University of Southern California (ISI). I held various position titles with the RFC Editor project at ISI, ending with Senior Editor.

4. The RFC Editor function was conducted by ISI under contract to the United States government prior to 1998. In 1998, ISOC, in furtherance of its IETF activity, entered into the first in a series of contracts with ISI providing for ISI's performance of the RFC Editor function. Beginning in 2010, certain aspects of the RFC Editor function were assumed by the RFC Production Center operation of AMS under contract to ISOC (acting through its IETF function and, in particular, the IETF Administrative Oversight Committee (now the IETF Administration LLC (IETF)). The business records of the RFC Editor function as it was conducted by ISI are currently housed on the computer systems of AMS, as contractor to the IETF.

5. I make this declaration based on my personal knowledge and information contained in the business records of the RFC Editor as they are currently housed at AMS, or confirmation with other responsible RFC Editor personnel with such knowledge.

6. Prior to 1998, the RFC Editor's regular practice was to publish RFCs, making them available from a repository via FTP. When a new RFC was published, an announcement of its publication, with information on how to access the RFC, would be typically sent out within 24 hours of the publication.

7. Since 1998, the RFC Editor's regular practice was to publish RFCs, making them available on the RFC Editor website or via FTP. When a new RFC was published, an announcement of its publication, with information on how to access the RFC, would be typically sent out within 24 hours of the publication. The announcement would go out to all subscribers and a contemporaneous electronic record of the announcement is kept in the RFC Editor mail archive that is available online.

8. Beginning in 1998, any RFC published on the RFC Editor website or via FTP was reasonably accessible to the public and was disseminated or otherwise available to the extent that persons interested and ordinarily skilled in the subject matter or art exercising reasonable diligence could have located it. In particular, the RFCs were indexed and placed in a public repository.

9. The RFCs are kept in an online repository in the course of the RFC Editor's regularly conducted activity and ordinary course of business. The records are made pursuant to established procedures and are relied upon by the RFC Editor in the performance of its functions.

10. It is the regular practice of the RFC Editor to make and keep the RFC records.

11. Based on the business records for the RFC Editor and the RFC Editor's course of conduct in publishing RFCs, I have determined that the publication date of RFC 2961 was no later than April 2001, at which time it was reasonably accessible to the public either on the RFC Editor website or via FTP from a repository. An announcement of its publication also would have been sent out to subscribers within 24 hours of its publication. A copy of that RFC is attached to this declaration as Exhibit 1.

12. Based on the business records for the RFC Editor and the RFC Editor's course of conduct in publishing RFCs, I have determined that the publication date of RFC 3175 was no later than September 2001, at which time it was reasonably accessible to the public either on the RFC Editor website or via FTP from a repository. An announcement of its publication also would have been sent out to subscribers within 24 hours of its publication. A copy of that RFC is attached to this declaration as Exhibit 2.

Pursuant to Section 1746 of Title 28 of United States Code, I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct and that the foregoing is based upon personal knowledge and information and is believed to be true.

Date: 19 March 2021 By: ______________________

Sandy Ginoza

4820-8233-5201

Network Working Group
Request for Comments: 2961
Category: Standards Track

L. Berger
LabN Consulting, LLC
D. Gan
Juniper Networks, Inc.
G. Swallow
Cisco Systems, Inc.
P. Pan
Juniper Networks, Inc.
F. Tommasi
S. Molendini
University of Lecce

April 2001

RSVP Refresh Overhead Reduction Extensions

Status of this Memo

Copyright Notice

Abstract

This document describes a number of mechanisms that can be used to reduce processing overhead requirements of refresh messages, eliminate the state synchronization latency incurred when an RSVP (Resource ReserVation Protocol) message is lost and, when desired, refreshing state without the transmission of whole refresh messages. The same extensions also support reliable RSVP message delivery on a per hop basis. These extension present no backwards compatibility issues.

Table of Contents

1. Introduction and Background

Standard RSVP [RFC2205] maintains state via the generation of RSVP refresh messages. Refresh messages are used to both synchronize state between RSVP neighbors and to recover from lost RSVP messages. The use of Refresh messages to cover many possible failures has resulted in a number of operational problems. One problem relates to scaling, another relates to the reliability and latency of RSVP Signaling.

The scaling problems are linked to the resource requirements (in terms of processing and memory) of running RSVP. The resource requirements increase proportionally with the number of sessions. Each session requires the generation, transmission, reception and processing of RSVP Path and Resv messages per refresh period. Supporting a large number of sessions, and the corresponding volume of refresh messages, presents a scaling problem.

The reliability and latency problem occurs when a non-refresh RSVP message is lost in transmission. Standard RSVP [RFC2205] recovers from a lost message via RSVP refresh messages. In the face of transmission loss of RSVP messages, the end-to-end latency of RSVP signaling is tied to the refresh interval of the node(s) experiencing the loss. When end-to-end signaling is limited by the refresh interval, the delay incurred in the establishment or the change of a reservation may be beyond the range of what is acceptable for some applications.

One way to address the refresh volume problem is to increase the refresh period, "R" as defined in Section 3.7 of [RFC2205]. Increasing the value of R provides linear improvement on transmission overhead, but at the cost of increasing the time it takes to synchronize state.

One way to address the reliability and latency of RSVP Signaling is to decrease the refresh period R. Decreasing the value of R increases the probability that state will be installed in the face of message loss, but at the cost of increasing refresh message rate and associated processing requirements.

An additional issue is the time to deallocate resources after a tear message is lost. RSVP does not retransmit ResvTear or PathTear messages. If the sole tear message transmitted is lost, then resources will only be deallocated once the "cleanup timer" interval has passed. This may result in resources being allocated for an unnecessary period of time. Note that even when the refresh period is adjusted, the "cleanup timer" must still expire since tear messages are not retransmitted.

The extensions defined in this document address both the refresh volume and the reliability issues with mechanisms other than adjusting refresh rate. The extensions are collectively referred to as the "Refresh Overhead Reduction" or the "Refresh Reduction" extensions. A Bundle message is defined to reduce overall message handling load. A MESSAGE_ID object is defined to reduce refresh message processing by allowing the receiver to more readily identify an unchanged message. A MESSAGE_ACK object is defined which can be used to detect message loss and support reliable RSVP message delivery on a per hop basis. A summary refresh message is defined to enable refreshing state without the transmission of whole refresh messages, while maintaining RSVP's ability to indicate when state is lost and to adjust to changes in routing.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.1. Trigger and Refresh Messages

This document categorizes RSVP messages into two types: trigger and refresh messages. Trigger messages are those RSVP messages that advertise state or any other information not previously transmitted. Trigger messages include messages advertising new state, a route change that alters a reservation path, or a modification to an existing RSVP session or reservation. Trigger messages also include those messages that include changes in non-RSVP processed objects, such as changes in the Policy or ADSPEC objects.

Refresh messages represent previously advertised state and contain exactly the same objects and same information as a previously transmitted message, and are sent over the same path. Only Path and Resv messages can be refresh messages. Refresh messages are identical to the corresponding previously transmitted message, with some possible exceptions. Specifically, the checksum field, the flags field and the INTEGRITY object may differ in refresh messages.

2. Refresh-Reduction-Capable Bit

To indicate support for the refresh overhead reduction extensions, an additional capability bit is added to the common RSVP header, which is defined in [RFC2205].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vers | Flags |   Msg Type    |         RSVP Checksum         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Send_TTL    |  (Reserved)  |         RSVP Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Flags: 4 bits

0x01: Refresh (overhead) reduction capable

When set, indicates that this node is willing and capable of receiving all the messages and objects described in this document. This includes the Bundle message described in Section 3, the MESSAGE_ID objects and Ack messages described in Section 4, and the MESSAGE_ID LIST objects and Srefresh message described in Section 5. This bit is meaningful only between RSVP neighbors.

Nodes supporting the refresh overhead reduction extensions must also take care to recognize when a next hop stops sending RSVP messages with the Refresh-Reduction-Capable bit set. To cover this case, nodes supporting the refresh overhead reduction extensions MUST examine the flags field of each received RSVP message. If the flag changes from indicating support to indicating non-support then, unless configured otherwise, Srefresh messages (described in Section 5) MUST NOT be used for subsequent state refreshes to that neighbor and Bundle messages (Section 3) MUST NOT be sent to that neighbor. Note, a node that supports reliable RSVP message delivery (Section 4) but not Bundle and Srefresh messages, will not set the Refresh-Reduction-Capable bit.

3. RSVP Bundle Message

An RSVP Bundle message consists of a bundle header followed by a body consisting of a variable number of standard RSVP messages. A Bundle message is used to aggregate multiple RSVP messages within a single PDU. The term "bundling" is used to avoid confusion with RSVP reservation aggregation. The following subsections define the formats of the bundle header and the rules for including standard RSVP messages as part of the message.

3.1. Bundle Header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vers  | Flags |   Msg type    |         RSVP checksum         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Send_TTL    | (Reserved)    |         RSVP length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The format of the bundle header is identical to the format of the RSVP common header [RFC2205]. The fields in the header are as follows:

Vers: 4 bits

Protocol version number. This is version 1.

Flags: 4 bits

0x01: Refresh (overhead) reduction capable

See Section 2.

0x02-0x08: Reserved

Msg type: 8 bits

12 = Bundle

RSVP checksum: 16 bits

The one's complement of the one's complement sum of the entire 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. Because individual sub-messages may carry their own checksum as well as the INTEGRITY object for authentication, this field MAY be set to zero. Note that when the checksum is not computed, the header of the bundle message will not be covered by any checksum. If the checksum is computed, individual sub-messages MAY set their own checksum to zero.

Send_TTL: 8 bits

The IP TTL value with which the message was sent. This is used by RSVP to detect a non-RSVP hop by comparing the Send_TTL with the IP TTL in a received message.

RSVP length: 16 bits

The total length of this RSVP Bundle message in bytes, including the bundle header and the sub-messages that follow.

3.2. Message Formats

An RSVP Bundle message must contain at least one sub-message. A sub-message MAY be any message type except for another Bundle message.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | Vers  | Flags |       12      |         RSVP checksum         |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |   Send_TTL    |  (Reserved)   |         RSVP length           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    //                   First sub-message                         //
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    //                   More sub-messages..                       //
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3. Sending RSVP Bundle Messages

Support for RSVP Bundle messages is optional. While message bundling helps in scaling RSVP, by reducing processing overhead and bandwidth consumption, a node is not required to transmit every standard RSVP message in a Bundle message. A node MUST always be ready to receive standard RSVP messages.

RSVP Bundle messages can only be sent to RSVP neighbors that support bundling. Methods for discovering such information include: (1) manual configuration and (2) observing the Refresh-Reduction-Capable bit (see Section 2) in the received RSVP messages. RSVP Bundle messages MUST NOT be used if the RSVP neighbor does not support RSVP Bundle messages.

RSVP Bundle messages are sent hop by hop between RSVP-capable nodes as "raw" IP datagrams with protocol number 46. The IP source address is an address local to the system that originated the Bundle message. The IP destination address is the RSVP neighbor for which the sub-messages are intended.

RSVP Bundle messages SHOULD NOT be sent with the Router Alert IP option in their IP headers. This is because Bundle messages are addressed directly to RSVP neighbors.

Each RSVP Bundle message MUST occupy exactly one IP datagram, which is approximately 64K bytes. If it exceeds the MTU, the datagram is fragmented by IP and reassembled at the recipient node. Implementations may choose to limit each RSVP Bundle message to the MTU size of the outgoing link, e.g., 1500 bytes. Implementations SHOULD also limit the amount of time that a message is delayed in order to be bundled. Different limits may be used for trigger and standard refresh messages. Trigger messages SHOULD be delayed a minimal amount of time. Refresh messages may be delayed up to their refresh interval. Note that messages related to the same Resv or Path state should not be delayed at different intervals in order to preserve ordering.

If the RSVP neighbor is not known or changes in next hops cannot be identified via routing, Bundle messages MUST NOT be used. Note that when the routing next hop is not RSVP capable it will typically not be possible to identify changes in next hop.

Any message that will be handled by the RSVP neighbor indicated in a Bundle Message's destination address may be included in the same message. This includes all RSVP messages that would be sent out a point-to-point link. It includes any message, such as a Resv, addressed to the same destination address. It also includes Path and PathTear messages when the next hop is known to be the destination and changes in next hops can be detected. Path and PathTear messages for multicast sessions MUST NOT be sent in Bundle messages when the outgoing link is not a point-to-point link or when the next hop does not support the refresh overhead reduction extensions.

3.4. Receiving RSVP Bundle Messages

If the local system does not recognize or does not wish to accept a Bundle message, the received messages shall be discarded without further analysis.

The receiver next compares the Send_TTL with which a Bundle message is sent to the IP TTL with which it is received. If a non-RSVP hop is detected, the number of non-RSVP hops is recorded. It is used later in processing of sub-messages.

Next, the receiver verifies the version number and checksum of the RSVP Bundle message and discards the message if any mismatch is found.

The receiver then starts decapsulating individual sub-messages. Each sub-message has its own complete message length and authentication information. With the exception of using the Send_TTL from the header of the Bundle message, each sub-message is processed as if it was received individually.

4. MESSAGE_ID Extension

Three new objects are defined as part of the MESSAGE_ID extension. The objects are the MESSAGE_ID object, the MESSAGE_ID_ACK object, and the MESSAGE_ID_NACK objects. The first two objects are used to support acknowledgments and reliable RSVP message delivery. The last object is used to support the summary refresh extension described in Section 5. The MESSAGE_ID object can also be used to simply provide a shorthand indication of when the message carrying the object is a refresh message. Such information can be used by the receiving node to reduce refresh processing requirements.

Message identification and acknowledgment is done on a per hop basis. All types of MESSAGE_ID objects contain a message identifier. The identifier MUST be unique on a per object generator's IP address basis. No more than one MESSAGE_ID object may be included in an RSVP message. Each message containing a MESSAGE_ID object may be acknowledged via a MESSAGE_ID_ACK object, when so indicated. MESSAGE_ID_ACK and MESSAGE_ID_NACK objects may be sent piggy-backed in unrelated RSVP messages or in RSVP Ack messages. RSVP messages carrying any of the three object types may be included in a bundle message. When included, each object is treated as if it were contained in a standard, non-bundled, RSVP message.

4.1. Modification of Standard Message Formats

The MESSAGE_ID, MESSAGE_ID_ACK and MESSAGE_ID_NACK objects may be included in the standard RSVP messages, as defined in [RFC2205]. When included, one or more MESSAGE_ID_ACK or MESSAGE_ID_NACK objects MUST immediately follow the INTEGRITY object. When no INTEGRITY object is present, the MESSAGE_ID_ACK or MESSAGE_ID_NACK objects MUST immediately follow the message or sub-message header. Only one MESSAGE_ID object MAY be included in a message or sub-message and it MUST follow any present MESSAGE_ID_ACK or MESSAGE_ID_NACK objects. When no MESSAGE_ID_ACK or MESSAGE_ID_NACK objects are present, the MESSAGE_ID object MUST immediately follow the INTEGRITY object. When no INTEGRITY object is present, the MESSAGE_ID object MUST immediately follow the message or sub-message header.

The ordering of the ACK objects for all standard RSVP messages is:

```
<Common Header>  [ <INTEGRITY> ]
                 [ [<MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>] ... ]
                 [ <MESSAGE_ID> ]
```

4.2. MESSAGE_ID Objects

MESSAGE_ID Class = 23

MESSAGE_ID object

Class = MESSAGE_ID Class, C_Type = 1

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                    Epoch                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Message_Identifier                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Flags: 8 bits

0x01 = ACK_Desired flag

Indicates that the sender requests the receiver to send an acknowledgment for the message.

Epoch: 24 bits

A value that indicates when the Message_Identifier sequence has reset. SHOULD be randomly generated each time a node reboots or the RSVP agent is restarted. The value SHOULD NOT be the same as was used when the node was last operational. This value MUST NOT be changed during normal operation.

Message_Identifier: 32 bits

When combined with the message generator's IP address, the Message_Identifier field uniquely identifies a message. The values placed in this field change incrementally and only decrease when the Epoch changes or when the value wraps.

4.3. MESSAGE_ID_ACK and MESSAGE_ID_NACK Objects

MESSAGE_ID_ACK Class = 24

MESSAGE_ID_ACK object

Class = MESSAGE_ID_ACK Class, C_Type = 1

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                    Epoch                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Message_Identifier                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Flags: 8 bits

No flags are currently defined. This field MUST be zero on transmission and ignored on receipt.

Epoch: 24 bits

The Epoch field copied from the message being acknowledged.

Message_Identifier: 32 bits

The Message_Identifier field copied from the message being acknowledged.

MESSAGE_ID_NACK object

Class = MESSAGE_ID_ACK Class, C_Type = 2

Definition is the same as the MESSAGE_ID_ACK object.

4.4. Ack Message Format

Ack messages carry one or more MESSAGE_ID_ACK or MESSAGE_ID_NACK objects. They MUST NOT contain any MESSAGE_ID objects. Ack messages are sent between neighboring RSVP nodes. The IP destination address of an Ack message is the unicast address of the node that generated the message(s) being acknowledged. For messages with RSVP_HOP objects, such as Path and Resv messages, the address is found in the RSVP_HOP object. For other messages, such as ResvConf, the associated IP address is the source address in the IP header. The IP source address is an address of the node that sends the Ack message.

The Ack message format is as follows:

```
<ACK Message> ::= <Common Header> [ <INTEGRITY> ]
                  <MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>
                  [ [<MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>] ... ]
```

For Ack messages, the Msg Type field of the Common Header MUST be set to 13.

Section 4.6 provides guidance on when an Ack message should be used and when MESSAGE_ID objects should be sent piggy-backed in other RSVP messages.

4.5. MESSAGE_ID Object Usage

The MESSAGE_ID object may be included in any RSVP message other than the Ack and Bundle messages. The MESSAGE_ID object is always generated and processed over a single hop between RSVP neighbors. The IP address of the object generator, i.e., the node that creates the object, is represented in a per RSVP message type specific fashion. For messages with RSVP_HOP objects, such as Path and Resv messages, the generator's IP address is found in the RSVP_HOP object. For other messages, such as ResvConf message, the generator's IP address is the source address in the IP header. Note that MESSAGE_ID objects can only be used in a Bundle sub-messages, but not in a Bundle message. As is always the case with the Bundle message, each sub-message is processed as if it was received individually. This includes processing of MESSAGE_ID objects.

The Epoch field contains a generator selected value. The value is used to indicate when the sender resets the values used in the Message_Identifier field. On startup, a node SHOULD randomly select a value to be used in the Epoch field. The node SHOULD ensure that the selected value is not the same as was used when the node was last operational. The value MUST NOT be changed unless the node or the RSVP agent is restarted.

The Message_Identifier field contains a generator selected value. This value, when combined with the generator's IP address, identifies a particular RSVP message and the specific state information it represents. The combination of Message_Identifier and Epoch can also be used to detect out of order messages. When a node is sending a refresh message with a MESSAGE_ID object, it SHOULD use the same Message_Identifier value that was used in the RSVP message that first advertised the state being refreshed. When a node is sending a trigger message, the Message_Identifier value MUST have a value that is greater than any other value previously used with the same Epoch field value. A value is considered to have been used when it has been sent in any message using the associated IP address with the same Epoch field value.

The ACK_Desired flag is set when the MESSAGE_ID object generator wants a MESSAGE_ID_ACK object sent in response to the message. Such information can be used to ensure reliable delivery of RSVP messages in the face of network loss. Nodes setting the ACK_Desired flag SHOULD retransmit unacknowledged messages at a more rapid interval than the standard refresh period until the message is acknowledged or until a "rapid" retry limit is reached. Rapid retransmission rate MUST be based on the exponential exponential back-off procedures defined in section 6. The ACK_Desired flag will typically be set only in trigger messages. The ACK_Desired flag MAY be set in refresh messages. Issues relate to multicast sessions are covered in a later section.

Nodes processing incoming MESSAGE_ID objects SHOULD check to see if a newly received message is out of order and can be ignored. Out of order messages SHOULD be ignored, i.e., silently dropped. Out of order messages can be identified by examining the values in the Epoch and Message_Identifier fields. To determine ordering, the received Epoch value must match the value previously received from the message sender. If the values differ then the receiver MUST NOT treat the message as out of order. When the Epoch values match and the Message_Identifier value is less than the largest value previously received from the sender, then the receiver SHOULD check the value previously received for the state associated with the message. This check should be performed for any message that installs or changes state. (Includes at least: Path, Resv, PathTear, ResvTear, PathErr and ResvErr.) If no local state information can be associated with the message, the receiver MUST NOT treat the message as out of order. If local state can be associated with the message and the received Message_Identifier value is less than the most recently received value associated with the state, the message SHOULD be treated as being out of order.

Note that the 32-bit Message_Identifier value MAY wrap. To cover the wrap case, the following expression may be used to test if a newly received Message_Identifier value is less than a previously received value:

```
if ((int) old_id - (int) new_id > 0) {
   new value is less than old value;
}
```

MESSAGE_ID objects of messages that are not out of order SHOULD be used to aid in determining if the message represents new state or a state refresh. Note that state is only refreshed in Path and Resv messages. If the received Epoch values differs from the value previously received from the message sender, the message is a trigger message and the receiver MUST fully process the message. If a Path or Resv message contains the same Message_Identifier value that was used in the most recently received message for the same session and, for Path messages, SENDER_TEMPLATE then the receiver SHOULD treat the message as a state refresh. If the Message_Identifier value is greater than the most recently received value, the receiver MUST fully processes the message. When fully processing a Path or Resv message, the receiver MUST store the received Message_Identifier value as part of the local Path or Resv state for future reference.

Nodes receiving a non-out of order message containing a MESSAGE_ID object with the ACK_Desired flag set, SHOULD respond with a MESSAGE_ID_ACK object. Note that MESSAGE_ID objects received in messages containing errors, i.e., are not syntactically valid, MUST NOT be acknowledged. PathErr and ResvErr messages SHOULD be treated as implicit acknowledgments.

4.6. MESSAGE_ID_ACK Object and MESSAGE_ID_NACK Object Usage

The MESSAGE_ID_ACK object is used to acknowledge receipt of messages containing MESSAGE_ID objects that were sent with the ACK_Desired flag set. A MESSAGE_ID_ACK object MUST NOT be generated in response to a received MESSAGE_ID object when the ACK_Desired flag is not set.

The MESSAGE_ID_NACK object is used as part of the summary refresh extension. The generation and processing of MESSAGE_ID_NACK objects is described in further detail in Section 5.4.

MESSAGE_ID_ACK and MESSAGE_ID_NACK objects MAY be sent in any RSVP message that has an IP destination address matching the generator of the associated MESSAGE_ID object. This means that the objects will not typically be included in the non hop-by-hop Path, PathTear and ResvConf messages. When no appropriate message is available, one or more objects SHOULD be sent in an Ack message. Implementations SHOULD include MESSAGE_ID_ACK and MESSAGE_ID_NACK objects in standard RSVP messages when possible.

Implementations SHOULD limit the amount of time that an object is delayed in order to be piggy-backed or sent in an Ack message. Different limits may be used for MESSAGE_ID_ACK and MESSAGE_ID_NACK objects. MESSAGE_ID_ACK objects are used to detect link transmission losses. If an ACK object is delayed too long, the corresponding message will be retransmitted. To avoid such retransmission, ACK objects SHOULD be delayed a minimal amount of time. A delay time equal to the link transit time MAY be used. MESSAGE_ID_NACK objects may be delayed an independent and longer time, although additional delay increases the amount of time a desired reservation is not installed.

4.7. Multicast Considerations

Path and PathTear messages may be sent to IP multicast destination addresses. When the destination is a multicast address, it is possible that a single message containing a single MESSAGE_ID object will be received by multiple RSVP next hops. When the ACK_Desired flag is set in this case, acknowledgment processing is more complex.

There are a number of issues to be addressed including ACK implosion, number of acknowledgments to be expected and handling of new receivers.

ACK implosion occurs when each receiver responds to the MESSAGE_ID object at approximately the same time. This can lead to a potentially large number of MESSAGE_ID_ACK objects being simultaneously delivered to the message generator. To address this case, the receiver MUST wait a random interval prior to acknowledging a MESSAGE_ID object received in a message destined to a multicast address. The random interval SHOULD be between zero (0) and a configured maximum time. The configured maximum SHOULD be set in proportion to the refresh and "rapid" retransmission interval, i.e, such that the maximum time before sending an acknowledgment does not result in retransmission. It should be noted that ACK implosion is being addressed by spreading acknowledgments out in time, not by ACK suppression.

A more fundamental issue is the number of acknowledgments that the upstream node, i.e., the message generator, should expect. The number of acknowledgments that should be expected is the same as the number of RSVP next hops. In the router-to-router case, the number of next hops can often be obtained from routing. When hosts are either the upstream node or the next hops, the number of next hops will typically not be readily available. Another case where the number of RSVP next hops will typically not be known is when there are non-RSVP routers between the message generator and the RSVP next hops.

When the number of next hops is not known, the message generator SHOULD only expect a single response. The result of this behavior will be special retransmission handling until the message is delivered to at least one next hop, then followed by standard RSVP refreshes. Refresh messages will synchronize state with any next hops that don't receive the original message.

4.7.1. Reference RSVP/Routing Interface

When using the MESSAGE_ID extension with multicast sessions it is preferable for RSVP to obtain the number of next hops from routing and to be notified when that number changes. The interface between routing and RSVP is purely an implementation issue. Since RSVP [RFC2205] describes a reference routing interface, a version of the RSVP/routing interface updated to provide number of next hop information is presented. See [RFC2205] for previously defined parameters and function description.

```
o    Route Query
     Mcast_Route_Query( [ SrcAddress, ] DestAddress,
                         Notify_flag )
                         -> [ IncInterface, ] OutInterface_list,
                         NHops_list

o    Route Change Notification
     Mcast_Route_Change( ) -> [ SrcAddress, ] DestAddress,
                        [ IncInterface, ] OutInterface_list,
                        NHops_list
```

NHops_list provides the number of multicast group members reachable via each OutInterface_list entry.

4.8. Compatibility

All nodes sending messages with the Refresh-Reduction-Capable bit set will support the MESSAGE_ID Extension. There are no backward compatibility issues raised by the MESSAGE_ID Class with nodes that do not set the Refresh-Reduction-Capable bit. The MESSAGE_ID Class has an assigned value whose form is 0bbbbbbb. Per RSVP [RFC2205], classes with values of this form must be rejected with an "Unknown Object Class" error by nodes not supporting the class. When the receiver of a MESSAGE_ID object does not support the class, a corresponding error message will be generated. The generator of the MESSAGE_ID object will see the error and then MUST re-send the original message without the MESSAGE_ID object. In this case, the message generator MAY still choose to retransmit messages at the "rapid" retransmission interval. Lastly, since the MESSAGE_ID_ACK class can only be issued in response to the MESSAGE_ID object, there are no possible issues with this class or Ack messages. A node MAY support the MESSAGE_ID Extension without supporting the other refresh overhead reduction extensions.

5. Summary Refresh Extension

The summary refresh extension enables the refreshing of RSVP state without the transmission of standard Path or Resv messages. The benefits of the described extension are that it reduces the amount of information that must be transmitted and processed in order to maintain RSVP state synchronization. Importantly, the described extension preserves RSVP's ability to handle non-RSVP next hops and to adjust to changes in routing. This extension cannot be used with Path or Resv messages that contain any change from previously transmitted messages, i.e., are trigger messages.

The summary refresh extension builds on the previously defined MESSAGE_ID extension. Only state that was previously advertised in Path and Resv messages containing MESSAGE_ID objects can be refreshed via the summary refresh extension.

The summary refresh extension uses the objects and the ACK message previously defined as part of the MESSAGE_ID extension, and a new Srefresh message. The new message carries a list of Message_Identifier fields corresponding to the Path and Resv trigger messages that established the state. The Message_Identifier fields are carried in one of three Srefresh related objects. The three objects are the MESSAGE_ID LIST object, the MESSAGE_ID SRC_LIST object, and the MESSAGE_ID MCAST_LIST object.

The MESSAGE_ID LIST object is used to refresh all Resv state, and Path state of unicast sessions. It is made up of a list of Message_Identifier fields that were originally advertised in MESSAGE_ID objects. The other two objects are used to refresh Path state of multicast sessions. A node receiving a summary refresh for multicast path state will at times need source and group information. These two objects provide this information. The objects differ in the information they contain and how they are sent. Both carry Message_Identifier fields and corresponding source IP addresses. The MESSAGE_ID SRC_LIST is sent in messages addressed to the session's multicast IP address. The MESSAGE_ID MCAST_LIST object adds the group address and is sent in messages addressed to the RSVP next hop. The MESSAGE_ID MCAST_LIST is normally used on point-to-point links.

An RSVP node receiving an Srefresh message, matches each listed Message_Identifier field with installed Path or Resv state. All matching state is updated as if a normal RSVP refresh message has been received. If matching state cannot be found, then the Srefresh message sender is notified via a refresh NACK.

A refresh NACK is sent via the MESSAGE_ID_NACK object. As described in the previous section, the rules for sending a MESSAGE_ID_NACK object are the same as for sending a MESSAGE_ID_ACK object. This includes sending MESSAGE_ID_NACK object both piggy-backed in unrelated RSVP messages or in RSVP ACK messages.

5.1. MESSAGE_ID LIST, SRC_LIST and MCAST_LIST Objects

MESSAGE_ID LIST object

MESSAGE_ID_LIST Class = 25

Class = MESSAGE_ID_LIST Class, C_Type = 1

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                     Epoch                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Message_Identifier                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                :                              |
//                               :                             //
|                                :                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Message_Identifier                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Flags: 8 bits

No flags are currently defined. This field MUST be zero on transmission and ignored on receipt.

Epoch: 24 bits

The Epoch field from the MESSAGE_ID object corresponding to the trigger message that advertised the state being refreshed.

Message_Identifier: 32 bits

The Message_Identifier field from the MESSAGE_ID object corresponding to the trigger message that advertised the state being refreshed. One or more Message_Identifiers may be included.

IPv4/MESSAGE_ID SRC_LIST object

Class = MESSAGE_ID_LIST Class, C_Type = 2

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                    Epoch                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Source_                            |
|                  Message_Identifier_Tuple                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                  :                            |
//                                 :                           //
|                                  :                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Source_                            |
|                  Message_Identifier_Tuple                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where a Source_Message_Identifier_Tuple consists of:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Message_Identifier                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Source_IP_Address (4 bytes)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv6/MESSAGE_ID SRC_LIST object

Class = MESSAGE_ID_LIST Class, C_Type = 3

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                     Epoch                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                          IPv6_Source_                         |
|                    Message_Identifier_Tuple                   |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                :                              |
//                               :                             //
|                                :                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                          IPv6_Source_                         |
|                    Message_Identifier_Tuple                   |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where a IPv6 Source_Message_Identifier_Tuple consists of:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Message_Identifier                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                    IPv6 Source_IP_Address                     |
|                          (16 Bytes)                           |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Flags: 8 bits

No flags are currently defined. This field MUST be zero on transmission and ignored on receipt.

Epoch: 24 bits

The Epoch field from the MESSAGE_ID object corresponding to the trigger message that advertised the state being refreshed.

Message_Identifier

The Message_Identifier field from the MESSAGE_ID object corresponding to the trigger message that advertised the Path state being refreshed. One or more Message_Identifiers may be included. Each Message_Identifier MUST be followed by the source IP address corresponding to the sender described in the Path state being refreshed.

Source_IP_Address

The IP address corresponding to the sender of the Path state being refreshed.

IPv4/MESSAGE_ID MCAST_LIST object

Class = MESSAGE_ID_LIST Class, C_Type = 4

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                     Epoch                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Multicast_                           |
|                     Message_Identifier_                       |
|                            Tuple                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               :                               |
//                              :                              //
|                               :                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Multicast_                           |
|                     Message_Identifier_                       |
|                            Tuple                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where a Multicast_Message_Identifier_Tuple consists of:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Message_Identifier                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source_IP_Address (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Destination_IP_Address (4 bytes)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv6/MESSAGE_ID MCAST_LIST object

Class = MESSAGE_ID_LIST Class, C_Type = 5

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                     Epoch                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                                                               |
|                          IPv6 Multicast_                      |
|                       Message_Identifier_                     |
|                              Tuple                            |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               :                               |
//                              :                              //
|                               :                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                                                               |
|                          IPv6 Multicast_                      |
|                       Message_Identifier_                     |
|                              Tuple                            |
|                                                               |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where a IPv6 Multicast_Message_Identifier_Tuple consists of:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Message_Identifier                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                      IPv6 Source_IP_Address                   |
|                            (16 Bytes)                         |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                    IPv6 Destination_IP_Address                |
|                            (16 Bytes)                         |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Flags: 8 bits

No flags are currently defined. This field MUST be zero on transmission and ignored on receipt.

Epoch: 24 bits

The Epoch field from the MESSAGE_ID object corresponding to the trigger message that advertised the state being refreshed.

Message_Identifier: 32 bits

The Message_Identifier field from the MESSAGE_ID object corresponding to the trigger message that advertised the Path state being refreshed. One or more Message_Identifiers may be included. Each Message_Identifier MUST be followed by the source IP address corresponding to the sender of the Path state being refreshed, and the destination IP address of the session.

Source_IP_Address

The IP address corresponding to the sender of the Path state being refreshed.

Destination_IP_Address

The destination IP address corresponding to the session of the Path state being refreshed.

5.2. Srefresh Message Format

Srefresh messages carry one or more MESSAGE_ID LIST, MESSAGE_ID SRC_LIST, and MESSAGE_ID MCAST_LIST objects. MESSAGE_ID LIST and MESSAGE_ID MCAST_LIST objects MAY be carried in the same Srefresh message. MESSAGE_ID SRC_LIST can not be combined in Srefresh messages with the other objects. A single Srefresh message MAY refresh both Path and Resv state.

Srefresh messages carrying Message_Identifier fields corresponding to Path state are normally sent with a destination IP address equal to the address carried in the corresponding SESSION objects. The destination IP address MAY be set to the RSVP next hop when the next hop is known to be RSVP capable and either (a) the session is unicast or (b) the outgoing interface is a point-to-point link. Srefresh messages carrying Message_Identifier fields corresponding to Resv state MUST be sent with a destination IP address set to the Resv state's previous hop.

Srefresh messages sent to a multicast session's destination IP address, MUST contain MESSAGE_ID SRC_LIST objects and MUST NOT include any MESSAGE_ID LIST or MESSAGE_ID MCAST_LIST objects. Srefresh messages sent to the RSVP next hop MAY contain either or both MESSAGE_ID LIST and MESSAGE_ID MCAST_LIST objects, but MUST NOT include any MESSAGE_ID SRC_LIST objects.

The source IP address of an Srefresh message is an address of the node that generates the message. The source IP address MUST match the address associate with the MESSAGE_ID objects when they were included in a standard RSVP message. As previously mentioned, the source address associated with a MESSAGE_ID object is represented in a per RSVP message type specific fashion. For messages with RSVP_HOP objects, such as Path and Resv messages, the address is found in the RSVP_HOP object. For other messages, such as ResvConf message, the associated IP address is the source address in the IP header.

Srefresh messages that are addressed to a session's destination IP address MUST be sent with the Router Alert IP option in their IP headers. Srefresh messages addressed directly to RSVP neighbors SHOULD NOT be sent with the Router Alert IP option in their IP headers.

Each Srefresh message MUST occupy exactly one IP datagram. If it exceeds the MTU, the datagram is fragmented by IP and reassembled at the recipient node. Srefresh messages MAY be sent within an RSVP Bundle messages. Although this is not expected since Srefresh messages can carry a list of Message_Identifier fields within a single object. Implementations may choose to limit each Srefresh message to the MTU size of the outgoing link, e.g., 1500 bytes.

The Srefresh message format is:

```
<Srefresh Message> ::= <Common Header> [ <INTEGRITY> ]
                       [ [<MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>] ... ]
                       [ <MESSAGE_ID> ]
                       <srefresh list> | <source srefresh list>

<srefresh list> ::= <MESSAGE_ID LIST> | <MESSAGE_ID MCAST_LIST>
                       [ <srefresh list> ]

<source srefresh list> ::= <MESSAGE_ID SRC_LIST>
                              [ <source srefresh list> ]
```

For Srefresh messages, the Msg Type field of the Common Header MUST be set to 15.

5.3. Srefresh Message Usage

An Srefresh message may be generated to refresh Resv and Path state. If an Srefresh message is used to refresh some particular state, then the generation of a standard refresh message for that particular state SHOULD be suppressed. A state's refresh interval is not affected by the use of Srefresh message based refreshes.

When generating an Srefresh message, a node SHOULD refresh as much Path and Resv state as is possible by including the information from as many MESSAGE_ID objects in the same Srefresh message. Only the information from MESSAGE_ID objects that meet the source and destination IP address restrictions, as described in Sections 5.2, may be included in the same Srefresh message. Identifying Resv state that can be refreshed using the same Srefresh message is fairly straightforward. Identifying which Path state may be included is a little more complex.

Only state that was previously advertised in Path and Resv messages containing MESSAGE_ID objects can be refreshed via an Srefresh message. Srefresh message based refreshes must preserve the state synchronization properties of Path or Resv message based refreshes. Specifically, the use of Srefresh messages MUST NOT result in state being timed-out at the RSVP next hop. The period at which state is refreshed when using Srefresh messages MAY be shorter than the period that would be used when using Path or Resv message based refreshes, but it MUST NOT be longer.

The particular approach used to trigger Srefresh message based refreshes is implementation specific. Some possibilities are triggering Srefresh message generation based on each state's refresh period or, on a per interface basis, periodically generating Srefresh messages to refresh all state that has not been refreshed within the state's refresh interval. Other approaches are also possible. A default Srefresh message generation interval of 30 seconds is suggested for nodes that do not dynamically calculate a generation interval.

When generating an Srefresh message, there are two methods for identifying which Path state may be refreshed in a specific message. In both cases, the previously mentioned refresh interval and source IP address restrictions must be followed. The primary method is to include only those sessions that share the same destination IP address in the same Srefresh message.

The secondary method for identifying which Path state may be refreshed within a single Srefresh message is an optimization. This method MAY be used when the next hop is known to support RSVP and when either (a) the session is unicast or (b) the outgoing interface is a point-to-point link. This method MUST NOT be used when the next hop is not known to support RSVP or when the outgoing interface is to a multi-access network and the session is to a multicast address. The use of this method MAY be administratively configured. When using this method, the destination address in the IP header of the Srefresh message is usually the next hop's address. When the use of this method is administratively configured, the destination address should be the well known group address 224.0.0.14. When the outgoing interface is a point-to-point link, all Path state associated with sessions advertised out the interface SHOULD be included in the same Srefresh message. When the outgoing interface is not a point-to-point link, all unicast session Path state SHOULD be included in the same Srefresh message.

Identifying which Resv state may be refreshed within a single Srefresh message is based simply on the source and destination IP addresses. Any state that was previously advertised in Resv messages with the same IP addresses as an Srefresh message MAY be included.

After identifying the Path and Resv state that can be included in a particular Srefresh message, the message generator adds to the message MESSAGE_ID information matching each identified state's previously used object. For all Resv state and for Path state of unicast sessions, the information is added to the message in a MESSAGE_ID LIST object that has a matching Epoch value. (Note only one Epoch value will be in use during normal operation.) If no matching object exists, then a new MESSAGE_ID LIST object is created.

Path state of multicast sessions may be added to the same message when the destination address of the Srefresh message is the RSVP next hop and the outgoing interface is a point-to-point link. In this case the information is added to the message in a MESSAGE_ID MCAST_LIST object that has a matching Epoch value. If no matching object exists, then a new MESSAGE_ID MCAST_LIST object is created. When the destination address of the message is a multicast address, then identified information is added to the message in a MESSAGE_ID SRC_LIST object that has a matching Epoch value. If no matching object exists, then a new MESSAGE_ID SRC_LIST object is created. Once the Srefresh message is composed, the message generator transmits the message out the proper interface.

Upon receiving an Srefresh message, the node MUST attempt to identify matching installed Path or Resv state. Matching is done based on the source address in the IP header of the Srefresh message, the object type and each Message_Identifier field. If matching state can be found, then the receiving node MUST update the matching state information as if a standard refresh message had been received. If matching state cannot be identified, then an Srefresh NACK MUST be generated corresponding to the unmatched Message_Identifier field. Message_Identifier fields received in MESSAGE_ID LIST objects may correspond to any Resv state or to Path state of unicast sessions. Message_Identifier fields received in MESSAGE_ID SRC_LIST or MCAST_LIST objects correspond to Path state of multicast sessions.

An additional check must be performed to determine if a NACK should be generated for unmatched Message_Identifier fields associated with Path state of multicast sessions, i.e., fields that were carried in MESSAGE_ID SRC_LIST or MCAST_LIST objects. The receiving node must check to see if the node would forward data packets originated from the source corresponding to the unmatched field. This check, commonly known as an RPF check, is performed based on the source and group information carried in the MESSAGE_ID SRC_LIST and MCAST_LIST objects. In both objects the IP address of the source is listed immediately after the corresponding Message_Identifier field. The group address is listed immediately after the source IP address in MESSAGE_ID MCAST_LIST objects. The group address is the message's destination IP address when MESSAGE_ID SRC_LIST objects are used. The receiving node only generates an Srefresh NACK when the node would forward packets to the identified group from the listed sender. If the node would forward multicast data packets from a listed sender and there is a corresponding unmatched Message_Identifier field, then an appropriate Srefresh NACK MUST be generated. If the node would not forward packets to the identified group from a listed sender, a corresponding unmatched Message_Identifier field is silently ignored.

5.4. Srefresh NACK

Srefresh NACKs are used to indicate that a received Message_Identifier field carried in MESSAGE_ID LIST, SRC_LIST, or MCAST_LIST object does not match any installed state. This may occur for a number of reasons including, for example, a route change. An Srefresh NACK is encoded in a MESSAGE_ID_NACK object. When generating an Srefresh NACK, the epoch and Message_Identifier fields of the MESSAGE_ID_NACK object MUST have the same value as was received. MESSAGE_ID_NACK objects are transmitted as described in Section 4.6.

Received MESSAGE_ID_NACK objects indicate that the object generator does not have any installed state matching the object. Upon receiving a MESSAGE_ID_NACK object, the receiver performs an installed Path or Resv state lookup based on the Epoch and Message_Identifier values contained in the object. If matching state is found, then the receiver MUST transmit the matching state via a standard Path or Resv message. If the receiver cannot identify any installed state, then no action is required.

5.5. Preserving RSVP Soft State

As discussed in [RFC2205], RSVP uses soft state to address a large class of potential errors. RSVP does this by periodically sending a full representation of installed state in Resv and Path messages. Srefresh messages are used in place of the periodic sending of standard Path and Resv refresh messages. While this provides scaling benefits and protects against common network events such as packet loss or routing change, it does not provide exactly the same error recovery properties. An example error that could potentially be recovered from via standard messages but not with Srefresh messages is internal corruption of state. This section recommends two methods that can be used to better preserve RSVP's soft state error recovery mechanism. Both mechanisms are supported using existing protocol messages.

The first mechanism uses a checksum or other algorithm to detect a previously unnoticed change in internal state. This mechanism does not protect against internal state corruption. It just covers the case where a trigger message should have been sent, but was not. When sending a Path or Resv trigger message, a node should run a checksum or other algorithm, such as [MD5], over the internal state and store the result. The choice of algorithm is an administrative decision. Periodically the node should rerun the algorithm and compare the new result with the stored result. If the values differ, then a corresponding standard Path or Resv refresh message should be sent and the new value should be stored. The recomputation period should be set based on the computation resources of the node and the reliability requirements of the network.

The second mechanism is simply to periodically send standard Path and Resv refresh messages. Since this mechanism uses standard refresh messages, it can recover from the same set of errors as standard RSVP. When using this mechanism, the period that standard refresh messages are sent must be longer than the interval that Srefresh messages are generated in order to gain the benefits of using the summary refresh extension. When a standard refresh message is sent, a corresponding summary refresh SHOULD NOT be sent during the same refresh period. When a node supports the periodic generation of standard refresh messages while Srefreshes are being used, the frequency of generation of standard refresh messages relative to the generation of summary refreshes SHOULD be configurable by the network administrator.

5.6. Compatibility

Nodes supporting the summary refresh extension advertise their support via the Refresh-Reduction-Capable bit in the RSVP message header. This enables nodes supporting the extension to detect each other. When it is not known if a next hop supports the extension, standard Path and Resv message based refreshes MUST be used. Note that when the routing next hop does not support RSVP, it will not always be possible to detect if the RSVP next hop supports the summary refresh extension. Therefore, when the routing next hop is not RSVP capable the Srefresh message based refresh SHOULD NOT be used. A node MAY be administratively configured to use Srefresh messages in all cases when all RSVP nodes in a network are known to support the summary refresh extension. This is useful since when operating in this mode, the extension properly adjusts to the case of non-RSVP next hops and changes in routing.

Per section 2, nodes supporting the summary refresh extension must also take care to recognize when a next hop stops sending RSVP messages with the Refresh-Reduction-Capable bit set.

6. Exponential Back-Off Procedures

This section is based on [Pan] and provides procedures to implement exponential back-off for retransmission of messages awaiting acknowledgment, see Section 4.5. Implementations MUST use the described procedures or their equivalent.

6.1. Outline of Operation

The following is one possible mechanism for exponential back-off retransmission of an unacknowledged RSVP message: When sending such a message, a node inserts a MESSAGE_ID object with the ACK_Desired flag set. The sending node will retransmit the message until a message acknowledgment is received or the message has been transmitted a maximum number of times. Upon reception, a receiving node acknowledges the arrival of the message by sending back a message acknowledgment (that is, a corresponding MESSAGE_ID_ACK object.) When the sending node receives the acknowledgment retransmission of the message is stopped. The interval between retransmissions is governed by a rapid retransmission timer. The rapid retransmission timer starts at a small interval and increases exponentially until it reaches a threshold.

6.2. Time Parameters

The described procedures make use of the following time parameters. All parameters are per interface.

Rapid retransmission interval Rf:

> Rf is the initial retransmission interval for unacknowledged messages. After sending the message for the first time, the sending node will schedule a retransmission after Rf seconds. The value of Rf could be as small as the round trip time (RTT) between a sending and a receiving node, if known.

Rapid retry limit Rl:

> Rl is the maximum number of times a message will be transmitted without being acknowledged.

Increment value Delta:

> Delta governs the speed with which the sender increases the retransmission interval. The ratio of two successive retransmission intervals is (1 + Delta).

Suggested default values are an initial retransmission timeout (Rf) of 500ms, a power of 2 exponential back-off (Delta = 1) and a retry limit (Rl) of 3.

6.3. Retransmission Algorithm

After a sending node transmits a message containing a MESSAGE_ID object with the ACK_Desired flag set, it should immediately schedule a retransmission after Rf seconds. If a corresponding MESSAGE_ID_ACK object is received earlier than Rf seconds, then retransmission SHOULD be canceled. Otherwise, it will retransmit the message after (1 + Delta)*Rf seconds. The staged retransmission will continue until either an appropriate MESSAGE_ID_ACK object is received, or the rapid retry limit, Rl, has been reached.

A sending node can use the following algorithm when transmitting a message containing a MESSAGE_ID object with the ACK_Desired flag set:

```
Prior to initial transmission initialize: Rk = Rf and Rn = 0

while (Rn++ < Rl)  {
    transmit the message;
    wake up after Rk seconds;
    Rk = Rk * (1 + Delta);
}
/* acknowledged or no reply from receiver for too long: */ do any
needed clean up; exit;
```

Asynchronously, when a sending node receives a corresponding MESSAGE_ID_ACK object, it will change the retry count, Rn, to Rl.

Note that the transmitting node does not advertise the use of the described exponential back-off procedures via the TIME_VALUE object.

6.4. Performance Considerations

The use of exponential back-off retransmission is a new and significant addition to RSVP. It will be important to review related operations and performance experience before this document advances to Draft Standard. It will be particularly important to review experience with multicast, and any ACK implosion problems actually encountered.

7. Acknowledgments

This document represents ideas and comments from the MPLS-TE design team and participants in the RSVP Working Group’s interim meeting. Thanks to Bob Braden, Lixia Zhang, Fred Baker, Adrian Farrel, Roch Guerin, Kireeti Kompella, David Mankins, Henning Schulzrinne, Andreas Terzis, Lan Wang and Masanobu Yuhara for specific feedback on the various versions of the document.

Portions of this work are based on work done by Masanobu Yuhara and Mayumi Tomikawa [Yuhara].

8. Security Considerations

No new security issues are raised in this document. See [RFC2205] for a general discussion on RSVP security issues.

9. References

[Pan] Pan, P., Schulzrinne, H., "Staged Refresh Timers for RSVP," Global Internet'97, Phoenix, AZ, November 1997. http://www.cs.columbia.edu/~pingpan/papers/timergi.pdf

[MD5] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2205] Braden, R., Ed., Zhang, L., Berson, S., Herzog, S. and S. Jamin , "Resource ReserVation Protocol -- Version 1 Functional Specification", RFC 2205, September 1997.

[Yuhara] Yuhara, M., and M Tomikawa, "RSVP Extensions for ID-based Refreshes", Work in Progress.

10. Authors' Addresses

Lou Berger
LabN Consulting, LLC

Phone: +1 301 468 9228
EMail: lberger@labn.net

Der-Hwa Gan
Juniper Networks, Inc.
1194 N. Mathilda Avenue,
Sunnyvale, CA 94089

Voice: +1 408 745 2074
Email: dhg@juniper.net

George Swallow
Cisco Systems, Inc.
250 Apollo Drive
Chelmsford, MA 01824

Phone: +1 978 244 8143
EMail: swallow@cisco.com

Ping Pan
Juniper Networks, Inc.
1194 N. Mathilda Avenue,
Sunnyvale, CA 94089

Voice: +1 408 745 3704
Email: pingpan@juniper.net

Franco Tommasi
University of Lecce, Fac. Ingegneria
Via Monteroni 73100 Lecce, ITALY

EMail: franco.tommasi@unile.it

Simone Molendini
University of Lecce, Fac. Ingegneria
Via Monteroni 73100 Lecce, ITALY

EMail: molendini@ultra5.unile.it

11. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

Network Working Group F. Baker
Request for Comments: 3175 C. Iturralde
Category: Standards Track F. Le Faucheur
B. Davie
Cisco Systems
September 2001

Aggregation of RSVP for IPv4 and IPv6 Reservations

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.

Copyright Notice

Abstract

This document describes the use of a single RSVP (Resource ReSerVation Protocol) reservation to aggregate other RSVP reservations across a transit routing region, in a manner conceptually similar to the use of Virtual Paths in an ATM (Asynchronous Transfer Mode) network. It proposes a way to dynamically create the aggregate reservation, classify the traffic for which the aggregate reservation applies, determine how much bandwidth is needed to achieve the requirement, and recover the bandwidth when the sub-reservations are no longer required. It also contains recommendations concerning algorithms and policies for predictive reservations.

1. Introduction

A key problem in the design of RSVP version 1 [RSVP] is, as noted in its applicability statement, that it lacks facilities for aggregation of individual reserved sessions into a common class. The use of such aggregation is recommended in [CSZ], and required for scalability.

The problem of aggregation may be addressed in a variety of ways. For example, it may sometimes be sufficient simply to mark reserved traffic with a suitable DSCP (e.g., EF), thus enabling aggregation of scheduling and classification state. It may also be desirable to install one or more aggregate reservations from ingress to egress of an "aggregation region" (defined below) where each aggregate reservation carries similarly marked packets from a large number of flows. This is to provide high levels of assurance that the end-to-end requirements of reserved flows will be met, while at the same time enabling reservation state to be aggregated.

Throughout, we will talk about "Aggregator" and "Deaggregator", referring to the routers at the ingress and egress edges of an aggregation region. Exactly how a router determines whether it should perform the role of aggregator or deaggregator is described below.

We will refer to the individual reserved sessions (the sessions we are attempting to aggregate) as "end-to-end" reservations ("E2E" for short), and to their respective Path/Resv messages as E2E Path/Resv messages. We refer to the the larger reservation (that which represents many E2E reservations) as an "aggregate" reservation, and its respective Path/Resv messages as "aggregate Path/Resv messages".

1.1. Problem Statement: Aggregation Of E2E Reservations

The problem of many small reservations has been extensively discussed, and may be summarized in the observation that each reservation requires a non-trivial amount of message exchange, computation, and memory resources in each router along the way. It would be nice to reduce this to a more manageable level where the load is heaviest and aggregation is possible.

Aggregation, however, brings its own challenges. In particular, it reduces the level of isolation between individual flows, implying that one flow may suffer delay from the bursts of another. Synchronization of bursts from different flows may occur. However, there is evidence [CSZ] to suggest that aggregation of flows has no negative effect on the mean delay of the flows, and actually leads to a reduction of delay in the "tail" of the delay distribution (e.g., 99% percentile delay) for the flows. These benefits of aggregation to some extent offset the loss of strict isolation.

1.2. Proposed Solution

The solution we propose involves the aggregation of several E2E reservations that cross an "aggregation region" and share common ingress and egress routers into one larger reservation from ingress to egress. We define an "aggregation region" as a contiguous set of systems capable of performing RSVP aggregation (as defined following) along any possible route through this contiguous set.

Communication interfaces fall into two categories with respect to an aggregation region; they are "exterior" to an aggregation region, or they are "interior" to it. Routers that have at least one interface in the region fall into one of three categories with respect to a given RSVP session; they aggregate, they deaggregate, or they are between an aggregator and a deaggregator.

Aggregation depends on being able to hide E2E RSVP messages from RSVP-capable routers inside the aggregation region. To achieve this end, the IP Protocol Number in the E2E reservation's Path, PathTear, and ResvConf messages is changed from RSVP (46) to RSVP-E2E-IGNORE (134) upon entering the aggregation region, and restored to RSVP at the deaggregator point. These messages are ignored (no state is stored and the message is forwarded as a normal IP datagram) by each router within the aggregation region whenever they are forwarded to an interior interface. Since the deaggregating router perceives the previous RSVP hop on such messages to be the aggregating router, Resv and other messages do not require this modification; they are unicast from RSVP hop to RSVP hop anyway.

The token buckets (SENDER_TSPECs and FLOWSPECS) of E2E reservations are summed into the corresponding information elements in aggregate Path and Resv messages. Aggregate Path messages are sent from the aggregator to the deaggregator(s) using RSVP's normal IP Protocol Number. Aggregate Resv messages are sent back from the deaggregator to the aggregator, thus establishing an aggregate reservation on behalf of the set of E2E flows that use this aggregator and deaggregator.

Such establishment of a smaller number of aggregate reservations on behalf of a larger number of E2E reservations yields the corresponding reduction in the amount of state to be stored and amount of signalling messages exchanged in the aggregation region.

By using Differentiated Services mechanisms for classification and scheduling of traffic supported by aggregate reservations (rather than performing per aggregate reservation classification and scheduling), the amount of classification and scheduling state in the aggregation region is even further reduced. It is not only independent of the number of E2E reservations, it is also independent of the number of aggregate reservations in the aggregation region. One or more Diff-Serv DSCPs are used to identify traffic covered by aggregate reservations and one or more Diff-Serv PHBs are used to offer the required forwarding treatment to this traffic. There may be more than one aggregate reservation between the same pair of routers, each representing different classes of traffic and each using a different DSCP and a different PHB.

1.3. Definitions

We define an "aggregation region" as a set of RSVP-capable routers for which E2E RSVP messages arriving on an exterior interface of one router in the set would traverse one or more interior interfaces (of this and possibly of other routers in the set) before finally traversing an exterior interface.

Such an E2E RSVP message is said to have crossed the aggregation region.

We define the "aggregating" router for this E2E flow as the first router that processes the E2E Path message as it enters the aggregation region (i.e., the one which forwards the message from an exterior interface to an interior interface).

We define the "deaggregating" router for this E2E flow as the last router to process the E2E Path as it leaves the aggregation region (i.e., the one which forwards the message from an interior interface to an exterior interface).

We define an "interior" router for this E2E flow as any router in the aggregation region which receives this message on an interior interface and forwards it to another interior interface. Interior routers perform neither aggregation nor deaggregation for this flow.

Note that by these definitions a single router with a mix of interior and exterior interfaces may have the capability to act as an aggregator on some E2E flows, a deaggregator on other E2E flows, and an interior router on yet other flows.

1.4. Detailed Aspects of Proposed Solution

A number of issues jump to mind in considering this model.

1.4.1. Traffic Classification Within The Aggregation Region

One of the reasons that RSVP Version 1 did not identify a way to aggregate sessions was that there was not a clear way to classify the aggregate. With the development of the Differentiated Services architecture, this is at least partially resolved; traffic of a particular class can be marked with a given DSCP and so classified. We presume this model.

We presume that on each link en route, a queue, WDM color, or similar management component is set aside for all aggregated traffic of the same class, and that sufficient bandwidth is made available to carry the traffic that has been assigned to it. This bandwidth may be adjusted based on the total amount of aggregated reservation traffic assigned to the same class.

There are numerous options for exactly which Diff-serv PHBs might be used for different classes of traffic as it crosses the aggregation region. This is the "service mapping" problem described in [RFC2998], and is applicable to situations broader than those described in this document. Arguments can be made for using either EF or one or more AF PHBs for aggregated traffic. For example, since controlled load requires non-TSpec-conformant (policed) traffic to be forwarded as best effort traffic rather than dropped, it may be appropriate to use an AF class for controlled load, using the higher drop preference for non-conformant packets.

In conventional (unaggregated) RSVP operation, a session is identified by a destination address and optionally a protocol port. Since data belonging to an aggregated reservation is identified by a DSCP, the session is defined by the destination address and DSCP. For those cases where two DSCPs are used (for conformant and non-conformant packets, as noted above), the session is identified by the DSCP of conformant packets. In general we will talk about mapping aggregated traffic onto a DSCP (even if a second DSCP may be used for non-conformant traffic).

Whichever PHB or PHBs are used to carry aggregated reservations, care needs to be take in an environment where provisioned Diff-Serv and aggregated RSVP are used in the same network, to ensure that the total admitted load for a single PHB does not exceed the link capacity allocated to that PHB. One solution to this is to reserve one PHB (or more) strictly for the aggregated reservation traffic (e.g., AF1 Class) while using other PHBs for provisioned Diff-Serv (e.g., AF2, AF3 and AF4 Classes).

Inside the aggregation region, some RSVP reservation state is maintained per aggregate reservation, while classification and scheduling state (e.g., DSCPs used for classifying traffic) is maintained on a per aggregate reservation class basis (rather than per aggregate reservation). For example, if Guaranteed Service reservations are mapped to the EF DSCP throughout the aggregation region, there may be a reservation for each aggregator/deaggregator pair in each router, but only the EF DSCP needs to be inspected at each interior interface, and only a single queue is used for all EF traffic.

1.4.2. Deaggregator Determination

The first question is "How do we determine the Aggregator/Deaggregator pair that are responsible for aggregating a particular E2E flow through the aggregation region?"

Determination of the aggregator is trivial: we know that an E2E flow has arrived at an aggregator when its Path message arrives at a router on an exterior interface and must be forwarded on an interior interface.

Determination of the deaggregator is more involved. If an SPF routing protocol, such as OSPF or IS-IS, is in use, and if it has been extended to advertise information on Deaggregation roles, it can tell us the set of routers from which the deaggregator will be chosen. In principle, if the aggregator and deaggregator are in the same area, then the identity of the deaggregator could be determined from the link state database. However, this approach would not work in multi-area environments or for distance vector protocols.

One method for Deaggregator determination is manual configuration. With this method the network operator would configure the Aggregator and the Deaggregator with the necessary information.

Another method allows automatic Deaggregator determination and corresponding Aggregator notification. When the E2E RSVP Path message transits from an interior interface to an exterior interface, the deaggregating router must advise the aggregating router of the correlation between itself and the flow. This has the nice attribute of not being specific to the routing protocol. It also has the property of automatically adjusting to route changes. For instance, if because of a topology change, another Deaggregator is now on the shortest path, this method will automatically identify the new Deaggregator and swap to it.

1.4.3. Mapping E2E Reservations Onto Aggregate Reservations

As discussed above, there may be multiple Aggregate Reservations between the same Aggregator/Deaggregator pair. The rules for mapping E2E reservations onto aggregate reservations are policy decisions which depend on the network environment and network administrator's objectives. Such a policy is outside the scope of this specification and we simply assume that such a policy is defined by the network administrator. We also assume that such a policy is somehow accessible to the Aggregators/Deaggregators but the details of how this policy is made accessible to Aggregators/Deaggregators (Local Configuration, COPS, LDAP, etc.) is outside the scope of this specification.

An example of very simple policy would be that all the E2E reservations are mapped onto a single Aggregate Reservation (i.e., single DSCP) between a given pair of Aggregator/Deaggregator.

Another example of policy, which takes into account the Int-Serv service type requested by the receiver (and signalled in the E2E Resv), would be where Guaranteed Service E2E reservations are mapped onto one DSCP in the aggregation region and where Controlled Load E2E reservations are mapped onto another DSCP.

A third example of policy would be one where the mapping of E2E reservations onto Aggregate Reservations take into account Policy Objects (such as information authenticating the end user) which may be included by the sender in the E2E path and/or by the receiver in the E2E Resv.

Regardless of the actual policy, a range of options are conceivable for where the decision to map an E2E reservation onto an aggregate reservation is taken and how this decision is communicated between Aggregator and Deaggregator. Both Aggregator and Deaggregator could be assumed to make such a decision independently. However, this would either require definition of additional procedures to solve inconsistent mapping decisions (i.e., Aggregator and Deaggregator decide to map a given E2E reservation onto different Aggregate Reservations) or would result in possible undetected misbehavior in the case of inconsistent decisions.

For simplicity and reliability, we assign the responsibility of the mapping decision entirely to the Deaggregator. The Aggregator is notified of the selected mapping by the Deaggregator and follows this decision. The Deaggregator was chosen rather than the Aggregator because the Deaggregator is the first to have access to all the information required to make such a decision (in particular receipt of the E2E Resv which indicates the requested Int-Serv service type and includes information signalled by the receiver). This allows faster operations such as set-up or size adjustment of an Aggregate Reservation in a number of situations resulting in faster E2E reservation establishment.

1.4.4. Size of Aggregate Reservations

A range of options exist for determining the size of the aggregate reservation, presenting a tradeoff between simplicity and scalability. Simplistically, the size of the aggregate reservation needs to be greater than or equal to the sum of the bandwidth of the E2E reservations it aggregates, and its burst capacity must be greater than or equal to the sum of their burst capacities. However, if followed religiously, this leads us to change the bandwidth of the aggregate reservation each time an underlying E2E reservation changes, which loses one of the key benefits of aggregation, the reduction of message processing cost in the aggregation region.

We assume, therefore, that there is some policy, not defined in this specification (although sample policies are suggested which have the necessary characteristics). This policy maintains the amount of bandwidth required on a given aggregate reservation by taking account of the sum of the bandwidths of its underlying E2E reservations, while endeavoring to change it infrequently. This may require some level of trend analysis. If there is a significant probability that in the next interval of time the current aggregate reservation will be exhausted, the router must predict the necessary bandwidth and request it. If the router has a significant amount of bandwidth reserved but has very little probability of using it, the policy may be to predict the amount of bandwidth required and release the excess.

This policy is likely to benefit from introduction of some hysteresis (i.e., ensure that the trigger condition for aggregate reservation size increase is sufficiently different from the trigger condition for aggregate reservation size decrease) to avoid oscillation in stable conditions.

Clearly, the definition and operation of such policies are as much business issues as they are technical, and are out of the scope of this document.

1.4.5. E2E Path ADSPEC update

As described above, E2E RSVP messages are hidden from the Interior routers inside the aggregation region. Consequently, the ADSPECs of E2E Path messages are not updated as they travel through the aggregation region. Therefore, the Deaggregator for a flow is responsible for updating the ADSPEC in the corresponding E2E Path to reflect the impact of the aggregation region on the QoS that may be achieved end-to-end. The Deaggregator should update the ADSPEC of the E2E Path as accurately as possible.

Since Aggregate Path messages are processed inside the aggregation region, their ADSPEC is updated by Interior routers to reflect the impact of the aggregation region on the QoS that may be achieved within the interior region. Consequently, the Deaggregator should make use of the information included in the ADSPEC from an Aggregate Path where available. The Deaggregator may elect to wait until such information is available before forwarding the E2E Path in order to accurately update its ADSPEC.

To maximize the information made available to the Deaggregator, whenever the Aggregator signals an Aggregate Path, the Aggregator should include an ADSPEC with fragments for all service types supported in the aggregation region (even if the Aggregate Path corresponds to an Aggregate Reservation that only supports a subset of those service types). Providing this information to the Deaggregator for every possible service type facilitates accurate and timely update of the E2E ADSPEC by the Deaggregator.

Depending on the environment and on the policy for mapping E2E reservations onto Aggregate Reservations, to accurately update the E2E Path ADSPEC, the Deaggregator may for example:

- update all the E2E Path ADSPEC segments (Default General Parameters Fragment, Guaranteed Service Fragment, Controlled-Load Service Fragment) based on the ADSPEC of a single Aggregate Path, or

- update the E2E Path ADSPEC by taking into account the ADSPEC from multiple Aggregate Path messages (e.g.,. update the Default General Parameters Fragment using the "worst" value for each parameter across all the Aggregate Paths' ADSPECs, update the Guaranteed Service Fragment using the Guaranteed Service Fragment from the ADSPEC of the Aggregate Path for the reservation used for Guaranteed Services).

By taking into account the information contained in the ADSPEC of Aggregate Path(s) as mentioned above, the Deaggregator should be able to accurately update the E2E Path ADSPEC in most situations.

However, we note that there may be particular situations where the E2E Path ADSPEC update cannot be made entirely accurately by the Deaggregator. This is most likely to happen when the path taken across the aggregation region depends on the service requested in the E2E Resv, which is yet to arrive. Such a situation could arise if, for example:

- The service mapping policy for the aggregation region is such that E2E reservations requesting Guaranteed Service are mapped to a different PHB that those requesting Controlled Load service.

- Diff-Serv aware routing is used in the aggregation region, so that packets with different DSCPs follow different paths (sending them over different MPLS label switched paths, for example).

As a result, the ADSPEC for the aggregate reservation that supports guaranteed service may differ from the ADSPEC for the aggregate reservation that supports controlled load.

Assume that the sender sends an E2E Path with an ADSPEC containing segments for both Guaranteed Services and Controlled Load. Then, at the time of updating the E2E ADSPEC, the Deaggregator does not know which service type will actually be requested by the receiver and therefore cannot know which PHB will be used to transport this E2E flow and, in turn, cannot pick the right parameter values to factor in when updating the Default General Parameters Fragment. As mentioned above, in this particular case, a conservative approach would be to always take into account the worst value for every parameter. Regardless of whether this conservative approach is followed or some simpler approach such as taking into account one of the two Aggregate Path ADSPEC, the E2E Path ADSPEC will be inaccurate (over-optimistic or over-pessimistic) for at least one service type actually requested by the destination.

Recognizing that entirely accurate update of E2E Path ADSPEC may not be possible in all situations, we recommend that a conservative approach be taken in such situations (over-pessimistic rather than over-optimistic) and that the E2E Path ADSPEC be corrected as soon as possible. In the example described above, this would mean that as soon as the Deaggregator receives the E2E Resv from the receiver, the Deaggregator should generate another E2E Path with an accurately updated ADSPEC based on the knowledge of which aggregate reservation will actually carry the E2E flow.

1.4.6. Intra-domain Routes

RSVP directly handles route changes, in that reservations follow the routes that their data follow. This follows from the property that Path messages contain the same IP source and destination address as the data flow for which a reservation is to be established. However, since we are now making aggregate reservations by sending a Path message from an aggregating to a deaggregating router, the reserved (E2E) data packets no longer carry the same IP addresses as the relevant (aggregate) Path message. The issue becomes one of making sure that data packets for reserved flows follow the same path as the Path message that established Path state for the aggregate reservation. Several approaches are viable.

First, the data may be tunneled from aggregator to deaggregator, using technologies such as IP-in-IP tunnels, GRE tunnels, MPLS label-switched paths, and so on. These each have particular advantages, especially MPLS, which allows traffic engineering. They each also have some cost in link overhead and configuration complexity.

If data is not tunneled, then we are depending on a characteristic of IP best metric routing , which is that if the route from A to Z includes the path from H to L, and the best metric route was chosen all along the way, then the best metric route was chosen from H to L. Therefore, an aggregate path message which crosses a given aggregator and deaggregator will of necessity use the best path between them.

If this is a single path, the problem is solved. If it is a multi-path route, and the paths are of equal cost, then we are forced to determine, perhaps by measurement, what proportion of the traffic for a given E2E reservation is passing along each of the paths, and assure ourselves of sufficient bandwidth for the present use. A simple, though wasteful, way of doing this is to reserve the total capacity of the aggregate route down each path.

For this reason, we believe it is advantageous to use one of the above-mentioned tunneling mechanisms in cases where multiple equal-cost paths may exist.

1.4.7. Inter-domain Routes

The case of inter-domain routes differs somewhat from the intra-domain case just described. Specifically, best-path considerations do not apply, as routing is by a combination of routing policy and shortest AS path rather than simple best metric.

In the case of inter-domain routes, data traffic belonging to different E2E sessions (but the same aggregate session) may not enter an aggregation region via the same aggregator interface, and/or may not leave via the same deaggregator interface. It is possible that we could identify this occurrence in some central system which sees the reservation information for both of the apparent sessions, but it is not clear that we could determine a priori how much traffic went one way or the other apart from measurement.

We simply note that this problem can occur and needs to be allowed for in the implementation. We recommend that each such E2E reservation be summed into its appropriate aggregate reservation, even though this involves over-reservation.

1.4.8. Reservations for Multicast Sessions

Aggregating reservations for multicast sessions is significantly more complex than for unicast sessions. The first challenge is to construct a multicast tree for distribution of the aggregate Path messages which follows the same path as will be followed by the data packets for which the aggregate reservation is to be made. This is complicated by the fact that the path taken by a data packet may depend on many factors such as its source address, the choice of shared trees or source-specific trees, and the location of a rendezvous point for the tree.

Once the problem of distributing aggregate Path messages is solved, there are considerable problems in determining the correct amount of resources to reserve at each link along the multicast tree. Because of the amount of heterogeneity that may exist in an aggregate multicast reservation, it appears that it would be necessary to retain information about individual E2E reservations within the aggregation region to allocate resources correctly. Thus, we may end up with a complex set of procedures for forming aggregate reservations that do not actually reduce the amount of stored state significantly for multicast sessions.

As noted above, there are several aspects to RSVP state, and our approach for unicast aggregates all forms of state: classification, scheduling, and reservation state. One possible approach to multicast is to focus only on aggregation of classification and scheduling state, which are arguably the most important because of their impact on the forwarding path. That approach is the one described in the current draft.

1.4.9. Multi-level Aggregation

Ideally, an aggregation scheme should be able to accommodate recursive aggregation, with aggregate reservations being themselves aggregated. Multi-level aggregation can be accomplished using the procedures described here and a simple extension to the protocol number swapping process.

We can consider E2E RSVP reservations to be at aggregation level 0. When we aggregate these reservations, we produce reservations at aggregation level 1. In general, level n reservations may be aggregated to form reservations at level n+1.

When an aggregating router receives an E2E Path, it swaps the protocol number from RSVP to RSVP-E2E-IGNORE. In addition, it should write the aggregation level (1, in this case) in the 2 byte field that is present (and currently unused) in the router alert option. In general, a router which aggregates reservations at level n to create reservations at level n+1 will write the number n+1 in the router alert field. A router which deaggregates level n+1 reservations will examine all messages with IP protocol number RSVP-E2E-IGNORE but will process the message and swap the protocol number back to RSVP only in the case where the router alert field carries the number n+1. For any other value, the message is forwarded unchanged. Interior routers ignore all messages with IP protocol number RSVP-E2E-IGNORE. Note that only a few bits of the 2 byte field in the option would be needed, given the likely number of levels of aggregation.

For IPv6, certain values of the router alert "value" field are reserved. This specification requires IANA assignment of a small number of consecutive values for the purpose of recording the aggregation level.

1.4.10. Reliability Issues

There are a variety of issues that arise in the context of aggregation that would benefit from some form of explicit acknowledgment mechanism for RSVP messages. For example, it is possible to configure a set of routers such that an E2E Path of protocol type RSVP-E2E-IGNORE would be effectively "black-holed", if it never reached a router which was appropriately configured to act as a deaggregator. It could then travel all the way to its destination where it would probably be ignored due to its non-standard protocol number. This situation is not easy to detect. The aggregator can be sure this problem has not occurred if an aggregate PathErr message is received from the deaggregator (as described in detail below). It can also be sure there is no problem if an E2E Resv is received. However, the fact that neither of these events has happened may only mean that no receiver wishes to reserve resources for this session, or that an RSVP message loss occurred, or it may mean that the Path was black-holed. However, if a neighbor-to-neighbor acknowledgment mechanism existed, the aggregator would expect to receive an acknowledgment of the E2E Path from the deaggregator, and would interpret the lack of a response as an indication that a problem of configuration existed. It could then refrain from aggregating this particular session. We note that such a reliability mechanism has been proposed for RSVP in [RFC291] and propose that it be used here.

1.4.11. Message Integrity and Node Authentication

[RSVP] defines a hop-by-hop authentication and integrity check. The present specification allows use of this check on Aggregate RSVP messages and also preserves this check on E2E RSVP messages for E2E RSVP messages.

Outside the Aggregation Region, any E2E RSVP message may contain an INTEGRITY object using a keyed cryptographic digest technique which assumes that RSVP neighbors share a secret. Because E2E RSVP messages are not processed by routers in the Aggregation Region, the Aggregator and Deaggregator appear as logical RSVP neighbors of each other. The Deaggregator is the Aggregator's Next Hop for E2E RSVP messages while the Aggregator is the Deaggregator's Previous Hop. Consequently, INTEGRITY objects which may appear in E2E RSVP messages traversing the Aggregation Region are exchanged directly between the Aggregator and Deaggregator in a manner which is entirely transparent to the Interior routers. Thus, hop-by-hop integrity checking for E2E messages over the Aggregation Region requires that the Aggregator and Deaggregator share a secret. Techniques for establishing that secret are described in [INTEGRITY].

Inside the Aggregation Region, any Aggregate RSVP message may contain an INTEGRITY object which assumes that the corresponding RSVP neighbors inside the Aggregation Region (e.g., Aggregator and Interior Router, two Interior Routers, Interior Router and Deaggregator) share a secret.

1.4.12. Aggregated reservations without E2E reservations

Up to this point we have assumed that the aggregate reservation is established as a result of the establishment of E2E reservations from outside the aggregation region. It should be clear that alternative triggers are possible. As discussed in [RFC2998], an aggregate RSVP reservation can be used to manage bandwidth in a diff-serv cloud even if RSVP is not used end-to-end.

The simplest example of an alternative configuration is the static configuration of an aggregated reservation for a certain amount for traffic from an ingress (aggregator) router to an egress (de-aggregator) router. This would have to be configured in at least the system originating the aggregate PATH message (the aggregator). The deaggregator could detect that the PATH message is directed to it, and could be configured to "turn around" such messages, i.e., it responds with a RESV back to the aggregator. Alternatively, configuration of the aggregate reservation could be performed at both the aggregator and the deaggregator. As before, an aggregate reservation is associated with a DSCP for the traffic that will use the reserved capacity.

In the absence of E2E microflow reservations, the aggregator can use a variety of policies to set the DSCP of packets passing into the aggregation region, thus determining whether they gain access to the resources reserved by the aggregate reservation. These policies are a matter of local configuration, as usual for a device at the edge of a diffserv cloud.

Note that the "aggregator" could even be a device such as a PSTN gateway which makes an aggregate reservation for the set of calls to another PSTN gateway (the deaggregator) across an intervening diff-serv region. In this case the reservation may be established in response to call signalling.

From the perspective of RSVP signalling and the handling of data packets in the aggregation region, these cases are equivalent to the case of aggregating E2E RSVP reservations. The only difference is that E2E RSVP signalling does not take place and cannot therefore be used as a trigger, so some additional knowledge is required in setting up the aggregate reservation.

2. Elements of Procedure

To implement aggregation, we define a number of elements of procedure.

2.1. Receipt of E2E Path Message By Aggregating Router

The very first event is the arrival of the E2E Path message at an exterior interface of an aggregator. Standard RSVP procedures [RSVP] are followed for this, including onto what set of interfaces the message should be forwarded. These interfaces comprise zero or more exterior interfaces and zero or more interior interfaces. (If the number of interior interfaces is zero, the router is not acting as an aggregator for this E2E flow.)

Service on exterior interfaces is handled as defined in [RSVP].

Service on interior interfaces is complicated by the fact that the message needs to be included in some aggregate reservation, but at this point it is not known which one, because the deaggregator is not known. Therefore, the E2E Path message is forwarded on the interior interface(s) using the IP Protocol number RSVP-E2E-IGNORE, but in every other respect identically to the way it would be sent by an RSVP router that was not performing aggregation.

2.2. Handling Of E2E Path Message By Interior Routers

At this point, the E2E Path message traverses zero or more interior routers. Interior routers receive the E2E Path message on an interior interface and forward it on another interior interface. The Router Alert IP Option alerts interior routers to check internally, but they find that the IP Protocol is RSVP-E2E-IGNORE and the next hop interface is interior. As such, they simply forward it as a normal IP datagram.

2.3. Receipt of E2E Path Message By Deaggregating Router

The E2E Path message finally arrives at a deaggregating router, which receives it on an interior interface and forwards it on an exterior interface. Again, the Router Alert IP Option alerts it to intercept the message, but this time the IP Protocol is RSVP-E2E-IGNORE and the next hop interface is an exterior interface.

Before forwarding the E2E Path towards the receiver, the Deaggregator should update its ADSPEC. This update is to reflect the impact of the aggregation region onto the QoS to be achieved E2E by the flow. Such information can be collected by the ADSPEC of Aggregate Path messages travelling from the Aggregator to the Deaggregator. Thus, to enable correct updating of the ADSPEC, a deaggregating router may wait as described below for the arrival of an aggregate Path before forwarding the E2E Path.

When receiving the E2E Path, depending on the policy for mapping E2E reservation onto Aggregate Reservations, the Deaggregator may or may not be in a position to decide which DSCP the E2E flow for the processed E2E Path is going to be mapped onto, as described above. If the Deaggregator is in a position to know the mapping at this point, then the Deaggregator first checks that there is an Aggregate Path in place for the corresponding DSCP. If so, then the Deaggregator uses the ADSPEC of this Aggregate Path to update the ADSPEC of the E2E Path and then forwards the E2E Path towards the receiver. If not, then the Deaggregator requests establishment of the corresponding Aggregate Path by sending an E2E PathErr message with an error code of NEW-AGGREGATE-NEEDED and the desired DSCP encoded in the DCLASS Object. The Deaggregator may also at the same time request establishment of an aggregate reservation for other DSCPs. When receiving the Aggregate Path for the desired DSCP, the Deaggregator then uses the ADSPEC of this Aggregate Path to update the ADSPEC of the E2E Path.

If the Deaggregator is not in a position to know the mapping at this point, then the Deaggregator uses the information contained in the ADSPEC of one Aggregate Path or of multiple Aggregate Paths to update the E2E Path ADSPEC. Similarly, if one or more of the necessary Aggregate Paths is not yet established, the Deaggregator requests establishment of the corresponding Aggregate Path by sending an E2E PathErr message with an error code of NEW-AGGREGATE-NEEDED and the desired DSCP encoded in the respective DCLASS Object. When receiving the Aggregate Path for the desired DSCP, the Deaggregator then uses the ADSPEC of this Aggregate Path to update the ADSPEC of the E2E Path.

Generating a E2E PathErr message with an error code of NEW-AGGREGATE-NEEDED should not result in any Path state being removed, but should result in the aggregating router initiating the necessary aggregate Path message, as described in the following section.

The deaggregating router changes the E2E Path message's IP Protocol from RSVP-E2E-IGNORE to RSVP and forwards the E2E Path message towards its intended destination.

2.4. Initiation of New Aggregate Path Message By Aggregating Router

The aggregating Router is responsible for generating a new Aggregate Path for a DSCP when receiving a E2E PathErr message with the error code NEW-AGGREGATE-NEEDED from the deaggregator. The DSCP value to include in the Aggregate Path Session is found in the DCLASS Object of the received E2E PathErr message. The identity of the deaggregator itself is found in the ERROR SPECIFICATION of the E2E PathErr message. The destination address of the aggregate Path message is the address of the deaggregating router, and the message is sent with IP protocol number RSVP.

Existing RSVP procedures specify that the size of a reservation established for a flow is set to the minimum of the Path SENDER_TSPEC and the Resv FLOW_SPEC. Consequently, the size of an Aggregate Reservation cannot be larger than the SENDER_TSPEC included in the Aggregate Path by the Aggregator. To ensure that Aggregate Reservations can be sized by the Deaggregator without undesired limitations, the Aggregating router should always attempt to include in the Aggregate Path a SENDER_TSPEC which is at least as large as the size that would actually be required as determined by the Deaggregator. One method to achieve this is to use a SENDER_TSPEC which is obviously larger than the highest load of E2E reservations that may be supported onto this network. Another method is for the Aggregator to keep track of which flows are mapped onto a DSCP and always add their E2E Path SENDER_TSPEC into the Aggregate Path SENDER_TSPEC (and possibly also add some additional bandwidth in anticipation of future E2E reservations).

The aggregating router is notified of the mapping from an E2E flow to a DSCP in two ways. First, when the aggregating router receives a E2E PathErr with error code NEW-AGGREGATE-NEEDED, the Aggregator is notified that the corresponding E2E flow is (at least temporarily) mapped onto a given DSCP. Secondly, when the aggregating router receives an E2E Resv containing a DCLASS Object (as described further below), the Aggregating Router is notified that the corresponding E2E flow is mapped onto a given DSCP.

2.5. Handling of E2E Resv Message by Deaggregating Router

Having sent the E2E Path message on toward the destination, the deaggregator must now expect to receive an E2E Resv for the session. On receipt, its responsibility is to ensure that there is sufficient bandwidth reserved within the aggregation region to support the new E2E reservation, and if there is, then to forward the E2E Resv to the aggregating router.

The Deaggregating router first makes the final decision of which Aggregate Reservation (and thus which DSCP) this E2E reservation is to be mapped onto. This decision is made according to the policy selected by the network administrator as described above.

If this final mapping decision is such that the Deaggregator can now make a more accurate update of the E2E Path ADSPEC than done when forwarding the initial E2E Path, the Deaggregator should do so and generate a new E2E Path immediately in order to provide the accurate ADSPEC information to the receiver as soon as possible. Otherwise, normal Refresh procedures should be followed for the E2E Path.

If no Aggregate Reservation currently exists from the corresponding aggregating router with the corresponding DSCP, the Deaggregating router will establish a new Aggregate Reservation as described in the next section.

If the corresponding Aggregate Reservation exists but has insufficient bandwidth reserved to accommodate the new E2E reservation (in addition to all the existing E2E reservations currently mapped onto it), it should follow the normal RSVP procedures [RSVP] for a reservation being placed with insufficient bandwidth to support the reservation. It may also first attempt to increase the aggregate reservation that is supplying bandwidth by increasing the size of the FLOW_SPEC that it includes in the aggregate Resv that it sends upstream. As discussed in the previous section, the Aggregating Router should ensure that the SENDER_TSPEC it includes in the Aggregate Path is always in excess of the FLOW_SPEC that may be requested in the Aggregate Resv by the Deaggregator, so that the Deaggregator is not unnecessarily prevented from effectively increasing the Aggregate Reservation bandwidth as required.

When sufficient bandwidth is available on the corresponding aggregate reservation, the Deaggregating Router may simply send the E2E Resv message with IP Protocol RSVP to the aggregating router. This message should include the DCLASS object to indicate which DSCP the aggregator must use for this E2E flow. The deaggregator will also add the token bucket from the E2E Resv FLOWSPEC object into its internal understanding of how much of the Aggregate reservation is in use.

As discussed above, in order to minimize the occurrence of situations where insufficient bandwidth is reserved on the corresponding Aggregate Reservation at the time of processing an E2E Resv, and in turn to avoid the delay associated with the increase of this aggregate bandwidth, the Deaggregator MAY anticipate the current demand and increase the Aggregate Reservations size ahead of actual requirements by E2E reservations.

2.6. Initiation of New Aggregate Resv Message By Deaggregating Router

Upon receiving an E2E Resv message on an exterior interface, and having determined the appropriate DSCP for the session according to the mapping policy, the Deaggregator looks for the corresponding path state for a session with the chosen DSCP. If aggregate Path state exists, but no aggregate Resv state exists, the Deaggregator creates a new aggregate Resv.

If no aggregate Path state exists for the appropriate DSCP, this may be because the Deaggregator could not decide earlier the final mapping for this E2E flow and elected to not establish Aggregate Path state for all DSCPs. In that case, the Deaggregator should request establishment of the corresponding Aggregate Path by sending a E2E PathErr with error code of NEW-AGGREGATE-NEEDED and with a DCLASS containing the required DSCP. This will trigger the Aggregator to establish the corresponding Aggregate Path. Once the Deaggregator has determined that the aggregate Path state is established, it creates a new Aggregate Resv.

The FLOW_SPEC of the new Aggregate Resv is set to a value not smaller than the requirement of the E2E reservation it is supporting. The Aggregate Resv is sent toward the aggregator (i.e., to the previous hop), using the AGGREGATED-RSVP session and filter specifications defined below. Since the DSCP is in the SESSION object, no DCLASS object is necessary. The message should be reliably delivered using the mechanisms in [RFC2961] or, alternatively, the CONFIRM object may be used, to assure that the aggregate Resv does indeed arrive and is granted. This enables the deaggregator to determine that the requested bandwidth is available to allocate to the E2E flows it supports.

In order to minimize the occurrence of situations where no corresponding Aggregate Reservation is established at the time of processing an E2E Resv, and in turn to avoid the delay associated with the creation of this aggregate reservation, the Deaggregator MAY anticipate the current demand and create the Aggregate Reservation before receiving E2E Resv messages requiring bandwidth on those aggregate reservations.

2.7. Handling of Aggregate Resv Message by Interior Routers

The aggregate Resv message is handled in essentially the same way as defined in [RSVP]. The Session object contains the address of the deaggregating router (or the group address for the session in the case of multicast) and the DSCP that has been chosen for the session. The Filterspec object identifies the aggregating router. These routers perform admission control and resource allocation as usual and send the aggregate Resv on towards the aggregator.

2.8. Handling of E2E Resv Message by Aggregating Router

The receipt of the E2E Resv message with a DCLASS Object is the final confirmation to the aggregating router of the mapping of the E2E reservation onto an Aggregate Reservation. Under normal circumstances, this is the only way it will be informed of this association. It should now forward the E2E Resv to its previous hop, following normal RSVP processing rules [RSVP].

2.9. Removal of E2E Reservation

E2E reservations are removed in the usual way via PathTear, ResvTear, timeout, or as the result of an error condition. When they are removed, their FLOWSPEC information must also be removed from the allocated portion of the aggregate reservation. This same bandwidth may be re-used for other traffic in the near future. When E2E Path messages are removed, their SENDER_TSPEC information must also be removed from the aggregate Path.

2.10. Removal of Aggregate Reservation

Should an aggregate reservation go away (presumably due to a configuration change, route change, or policy event), the E2E reservations it supports are no longer active. They must be treated accordingly.

2.11. Handling of Data On Reserved E2E Flow by Aggregating Router

Prior to establishment that a given E2E flow is part of a given aggregate, the flow’s data should be treated as traffic without a reservation by whatever policies prevail for such. Generally, this will mean being given the same forwarding behavior as best effort traffic. However, upon establishing that the flow belongs to a given aggregate, the aggregating router is responsible for marking any related traffic with the correct DSCP and forwarding it in the manner appropriate to traffic on that reservation. This may imply forwarding it to a given IP next hop, or piping it down a given link layer circuit, tunnel, or MPLS label switched path.

The aggregator is responsible for performing per-reservation policing on the E2E flows that it is aggregating. The aggregator performs metering of traffic belonging to each reservation to assess compliance to the token bucket for the corresponding E2E reservation. Packets which are assessed in compliance are forwarded as mentioned above. Packets which are assessed out of compliance must be either dropped, reshaped or marked to a different DSCP. The detailed policing behavior is an aspect of the service mapping described in [RFC2998].

2.12. Procedures for Multicast Sessions

Because of the difficulties of aggregating multicast sessions described above, we focus on the aggregation of scheduling and classification state in the multicast case. The main difference between the multicast and unicast cases is that rather than sending an aggregate Path message to the unicast address of a single deaggregating router, in the multicast case we send the "aggregate" Path message to the same group address as the E2E session. This ensures that the aggregate Path message follows the same route as the E2E Path. This difference between unicast and multicast is reflected in the Session objects defined below. A consequence of this approach is that we continue to have reservation state per multicast session inside the aggregation region.

A further challenge arises in multicast sessions with heterogeneous receivers. Consider an interior router which must forward packets for a multicast session on two interfaces, but has only received a reservation request on one of those interfaces. It receives packets marked with the DSCP chosen for the aggregate reservation. When sending them out the interface which has no installed reservation, it has the following options:

a) remark those packets to best effort before sending them out the interface;

b) send the packets out the interface with the DSCP chosen for the aggregate reservation.

The first approach suffers from the drawback that it requires nMF classification at an interior router in order to recognize the flows whose packets must be demoted. The second approach requires over-reservation of resources on the interface on which no reservation was received. In the absence of such over-reservation, the packets sent with the "wrong" DSCP would be able to degrade the service experienced by packets using that DSCP legitimately.

To make MF classification acceptable in an interior router, it may be possible to treat the case of heterogeneous flows as an exception. That is, an interior router only needs to be able to recognize those individual microflows that have heterogeneous resource needs on the outbound interfaces of this router.

3. Protocol Elements

3.1. IP Protocol RSVP-E2E-IGNORE

This specification requires the assignment of a protocol type RSVP-E2E-IGNORE, whose number is at this point 134. This is used only on E2E messages which require a router alert (Path, PathTear, and ResvConf), and signifies that the message must be treated one way when destined to an interior interface, and another way when destined to an exterior interface. The protocol type is swapped by the Aggregator from RSVP to RSVP-E2E-IGNORE in E2E Path, PathTear, and ResvConf messages when they enter the Aggregation Region. The protocol type is swapped back by the Deaggregator from RSVP-E2E-IGNORE to RSVP in such E2E messages when they exit the Aggregation Region.

3.2. Path Error Code

A PathErr code NEW-AGGREGATE-NEEDED is required. This value does not signify that a fatal error has occurred, but that an action is required of the aggregating router to avoid an error condition in the near future.

3.3. SESSION Object

The SESSION object contains two values: the IP Address of the aggregate session destination, and the DSCP that it will use on the E2E data the reservation contains. For unicast sessions, the session destination address is the address of the deaggregating router. For multicast sessions, the session destination is the multicast address of the E2E session (or sessions) being aggregated. The inclusion of the DSCP in the session allows for multiple sessions toward the same address to be distinguished by their DSCP and queued separately. It also provides the means for aggregating scheduling and classification state. In the case where a session uses a pair of PHBs (e.g., AF11 and AF12), the DSCP used should represent the numerically smallest PHB (e.g., AF11). This follows the same naming convention described in [BRIM].

Session types are defined for IPv4 and IPv6 addresses.

o IP4 SESSION object: Class = SESSION,
  C-Type = RSVP-AGGREGATE-IP4

```
+-------------+-------------+-------------+-------------+
|              IPv4 Session Address (4 bytes)           |
+-------------+-------------+-------------+-------------+
| /////////// |    Flags    |  /////////  |     DSCP    |
+-------------+-------------+-------------+-------------+
```

o IP6 SESSION object: Class = SESSION,
  C-Type = RSVP-AGGREGATE-IP6

```
+-------------+-------------+-------------+-------------+
|                                                       |
+                                                       +
|                                                       |
+              IPv6 Session Address (16 bytes)          +
|                                                       |
+                                                       +
|                                                       |
+-------------+-------------+-------------+-------------+
| /////////// |    Flags    |  /////////  |     DSCP    |
+-------------+-------------+-------------+-------------+
```

3.4. SENDER_TEMPLATE Object

The SENDER_TEMPLATE object identifies the aggregating router for the aggregate reservation.

o IP4 SENDER_TEMPLATE object: Class = SENDER_TEMPLATE,
  C-Type = RSVP-AGGREGATE-IP4

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

o IP6 SENDER_TEMPLATE object: Class = SENDER_TEMPLATE,
  C-Type = RSVP-AGGREGATE-IP6

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

3.5. FILTER_SPEC Object

The FILTER_SPEC object identifies the aggregating router for the aggregate reservation, and is syntactically identical to the SENDER_TEMPLATE object.

4. Policies and Algorithms For Predictive Management Of Blocks Of Bandwidth

The exact policies used in determining how much bandwidth should be allocated to an aggregate reservation at any given time are beyond the scope of this document, and may be proprietary to the service provider in question. However, here we explore some of the issues and suggest approaches.

In short, the ideal condition is that the aggregate reservation always has enough resources to allocate to any E2E reservation that requires its support, and never takes too much. Simply stated, but more difficult to achieve. Factors that come into account include significant times in the diurnal cycle: one may find that a large number of people start placing calls at 8:00 AM, even though the hour from 7:00 to 8:00 is dead calm. They also include recent history: if more people have been placing calls recently than have been finishing them, a prediction of the necessary bandwidth a few moments hence may call for more bandwidth than is currently allocated. Likewise, at the end of a busy period, we may find that the trend calls for declining reservation amounts.

We recommend a policy something along this line. At any given time, one should expect that the amount of bandwidth required for the aggregate reservation is the larger of the following:

(a) a requirement known a priori, such as from history of the diurnal cycle at a particular week day and time of day, and

(b) the trend line over recent history, with 90 or 99% statistical confidence.

We further expect that changes to that aggregate reservation would be made no more often than every few minutes, and ideally perhaps on larger granularity such as fifteen minute intervals or hourly. The finer the granularity, the greater the level of signaling required, while the coarser the granularity, the greater the chance for error, and the need to recover from that error.

In general, we expect that the aggregate reservation will not ever add up to exactly the sum of the reservations it supports, but rather will be an integer multiple of some block reservation size, which exceeds that value.

5. Security Considerations

Numerous security issues pertain to this document; for example, the loss of an aggregate reservation to an aggressor causes many calls to operate unreserved, and the reservation of a great excess of bandwidth may result in a denial of service. However, these issues are not confined to this extension: RSVP itself has them. We believe that the security mechanisms in RSVP address these issues as well.

One security issue specific to RSVP aggregation involves the modification of the IP protocol number in RSVP Path messages that traverse an aggregation region. If that field were maliciously modified in a Path message, it would cause the message to be ignored by all subsequent devices on its path, preventing reservations from being made. It could even be possible to correct the value before it reached the receiver, making it difficult to detect the attack. In theory, it might also be possible for a node to modify the IP protocol number for non-RSVP messages as well, thus interfering with the operation of other protocols.

One way to mitigate the risks of malicious modification of the IP protocol number is to use an IPSEC authentication header, which would ensure that malicious modification of the IP header is detected. This is a desirable approach but imposes some administrative burden in the form of key management for authentication purposes.

It is RECOMMENDED that implementations of this specification only support modification of the IP protocol number for RSVP Path, PathTear, and ResvConf messages. That is, a general facility for modification of the IP protocol number SHOULD NOT be made available.

Network operators deploying routers with RSVP aggregation capability should be aware of the risks of inappropriate modification of the IP protocol number and should take appropriate steps (physical security, password protection, etc.) to reduce the risk that a router could be configured by an attacker to perform malicious modification of the protocol number.

6. IANA Considerations

Section 1.2 proposes a new protocol type, RSVP-E2E-IGNORE, which is used to identify a message that routers in the network core will see; further processing of such messages may or may not be required, depending on the egress interface type, as described in Section 1.2. The IANA assigned IP protocol number 134, in accordance with [RFC2780], meeting the Standards Track publication criterion.

Section 1.4.9 describes the manner in which the Router Alert is used in the context of this specification, which is essentially a simple counter of the depth of nesting of aggregation. The IPv4 Router Alert [RFC2113] has the option simply to ask the router to look at the protocol type of the intercepted datagram and decide what to do with it; the parameter is additional information to that decision. The IPv6 Router Alert [RFC2711] turns the parameter into an option sub-type. As a result, the IPv6 router alert option may not be used algorithmically in the context of the protocol in question. The IANA assigned a block of 32 values (3-35, "Aggregated Reservation Nesting Level") which we may map to nesting depths 0..31, hoping that 32 levels is enough.

Section 3.2 discusses a new, required path error code. The IANA has assigned RSVP Parameters Error Code 26 to NEW-AGGREGATE-NEEDED.

Sections 3.3, 3.4, and 3.5 describe extensions to three object classes: Session, Filter Specification, and Sender Template. The IANA has assigned two new common C-Types to be specified for the aggregator's address. RSVP-AGGREGATE-IP4 is C-Type 9 and RSVP-AGGREGATE-IP6 is C-Type 10. In adding these C-types to IANA RSVP Class Names, Class Numbers and Class Types registry, the same numbering for them is used in all three Classes, as is done for IPv4 and IPv6 address tuples in [RSVP].

7. Acknowledgments

The authors acknowledge that published documents and discussion with several people, notably John Wroclawski, Steve Berson, and Andreas Terzis materially contributed to this document. The design is influenced by the RSVP tunnels document [TERZIS].

APPENDIX 1: Example Signalling Flow For First E2E Flow

This Appendix does not provide additional specification. It only illustrates the specification detailed above through a possible flow of RSVP signalling messages involved in the successful establishment of a unicast E2E reservation which is the first between a given pair of Aggregator/Deaggregator.

```
           Aggregator                                Deaggregator

    E2E Path
   ---------------->
                (1)
                           E2E Path
                     ------------------------------->
                                                   (2)
                      E2E PathErr(New-agg-needed, DCLASS=x)
                     <--------------------------------
                      E2E PathErr(New-agg-needed, DCLASS=y)
                     <--------------------------------
                (3)
                           AggPath(DSCP=x)
                     ------------------------------->
                           AggPath(DSCP=y)
                     ------------------------------->
                                                   (4)
                                                      E2E Path
                                                      ----------->
                                                   (5)
                           AggResv (DSCP=x)
                     <--------------------------------
                           AggResv (DSCP=y)
                     <--------------------------------
               (6)
                           AggResvConfirm (DSCP=x)
                     ------------------------------>
                           AggResvConfirm (DSCP=y)
                     ------------------------------>
                                                   (7)
                                                      E2E Resv
                                                      <----------
                                                   (8)
                           E2E Resv (DCLASS=x)
                     <-----------------------------
               (9)
       E2E Resv
   <---------------
```

(1) Aggregator forwards E2E Path into aggregation region after modifying its IP Protocol Number to RSVP-E2E-IGNORE

(2) Let's assume no Aggregate Path exists. To be able to accurately update the ADSPEC of the E2E Path, the Deaggregator needs the ADSPEC of Aggregate PATH. In this example the Deaggregator elects to instruct the Aggregator to set up Aggregate Path states for the two supported DSCPs by sending a New-Agg-Needed PathErr code for each DSCP.

(3) The Aggregator follows the request from the Deaggregator and signals an Aggregate Path for both DSCPs.

(4) The Deaggregator takes into account the information contained in the ADSPEC from both Aggregate Path and updates the E2E Path ADSPEC accordingly. The Deaggregator also modifies the E2E Path IP Protocol Number to RSVP before forwarding it.

(5) In this example, the Deaggregator elects to immediately proceed with establishment of Aggregate Reservations for both DSCPs. In effect, the Deaggregator can be seen as anticipating the actual demand of E2E reservations so that resources are available on Aggregate Reservations when the E2E Resv requests arrive in order to speed up establishment of E2E reservations. Assume also that the Deaggregator includes the optional Resv Confirm Request in these Aggregate Resv.

(6) The Aggregator merely complies with the received ResvConfirm Request and returns the corresponding Aggregate ResvConfirm.

(7) The Deaggregator has explicit confirmation that both Aggregate Resv are established.

(8) On receipt of the E2E Resv, the Deaggregator applies the mapping policy defined by the network administrator to map the E2E Resv onto an Aggregate Reservation. Let's assume that this policy is such that the E2E reservation is to be mapped onto the Aggregate Reservation with DSCP=x. The Deaggregator knows that an Aggregate Reservation is in place for the corresponding DSCP since (7). The Deaggregator performs admission control of the E2E Resv onto the Aggregate Resv for DSCP=x. Assuming that the Aggregate Resv for DSCP=x had been established with sufficient bandwidth to support the E2E Resv, the Deaggregator adjusts its counter tracking the unused bandwidth on the Aggregate Reservation and forwards the E2E Resv to the Aggregator including a DCLASS object conveying the selected mapping onto DSCP=x.

(9) The Aggregator records the mapping of the E2E Resv onto DSCP=x. The Aggregator removes the DCLASS object and forwards the E2E Resv towards the sender.

APPENDIX 2: Example Signalling Flow For Subsequent E2E Flow Without Reservation Resizing

This Appendix does not provide additional specification. It only illustrates the specification detailed above through a possible flow of RSVP signalling messages involved in the successful establishment of a unicast E2E reservation which follows other E2E reservations between a given pair of Aggregator/Deaggregator. This flow could be imagined as following the flow of messages illustrated in Appendix 1.

```
        Aggregator                                Deaggregator

 E2E Path
---------------->
             (10)
                        E2E Path
                    -------------------------------->
                                                   (11)
                                                      E2E Path
                                                      ----------->
                                                       E2E Resv
                                                      <-----------
                                                   (12)
                        E2E Resv (DCLASS=x)
                  <-----------------------------
              (13)
    E2E Resv
<---------------
```

(10) Aggregator forwards E2E Path into aggregation region after modifying its IP Protocol Number to RSVP-E2E-IGNORE

(11) Because previous E2E reservations have been established, let's assume that Aggregate Path exists for all supported DSCPs. The Deaggregator takes into account the information contained in the ADSPEC from the Aggregate Paths and updates the E2E Path ADSPEC accordingly. The Deaggregator also modifies the E2E Path IP Protocol Number to RSVP before forwarding it.

(12) On receipt of the E2E Resv, the Deaggregator applies the mapping policy defined by the network administrator to map the E2E Resv onto an Aggregate Reservation. Let's assume that this policy is such that the E2E reservation is to be mapped onto the Aggregate Reservation with DSCP=x. Because previous E2E reservations have

been established, let's assume that an Aggregate Reservation is in place for DSCP=x. The Deaggregator performs admission control of the E2E Resv onto the Aggregate Resv for DSCP=x. Assuming that the Aggregate Resv for DSCP=x has sufficient unused bandwidth to support the new E2E Resv, the Deaggregator then adjusts its counter tracking the unused bandwidth on the Aggregate Reservation and forwards the E2E Resv to the Aggregator including a DCLASS object conveying the selected mapping onto DSCP=x.

(13) The Aggregator records the mapping of the E2E Resv onto DSCP=x. The Aggregator removes the DCLASS object and forwards the E2E Resv towards the sender.

APPENDIX 3: Example Signalling Flow For Subsequent E2E Flow With Reservation Resizing

This Appendix does not provide additional specification. It only illustrates the specification detailed above through a possible flow of RSVP signalling messages involved in the successful establishment of a unicast E2E reservation which follows other E2E reservations between a given pair of Aggregator/Deaggregator. This flow could be imagined as following the flow of messages illustrated in Appendix 2.

```
                 Aggregator                        Deaggregator

    E2E Path
   ---------------->
                    (14)
                           E2E Path
                       ------------------------------->
                                                       (15)
                                                           E2E Path
                                                           ----------->

                                                           E2E Resv
                                                           <-----------


                                                       (16)
                        AggResv (DSCP=x, increased Bw)
                       <-------------------------------
                   (17)
                       AggResvConfirm (DSCP=x, increased Bw)
                       ------------------------------>
                                                       (18)
                          E2E Resv (DCLASS=x)
                       <-----------------------------
                   (19)
       E2E Resv
   <---------------
```

(14) Aggregator forwards E2E Path into aggregation region after modifying its IP Protocol Number to RSVP-E2E-IGNORE

(15) Because previous E2E reservations have been established, let's assume that Aggregate Path exists for all supported DSCPs. The Deaggregator takes into account the information contained in the ADSPEC from the Aggregate Paths and updates the E2E Path ADSPEC accordingly. The Deaggregator also modifies the E2E Path IP Protocol Number to RSVP before forwarding it.

(16) On receipt of the E2E Resv, the Deaggregator applies the mapping policy defined by the network administrator to map the E2E Resv onto an Aggregate Reservation. Let's assume that this policy is such that the E2E reservation is to be mapped onto the Aggregate Reservation with DSCP=x. Because previous E2E reservations have been established, let's assume that an Aggregate Reservation is in place for DSCP=x. The Deaggregator performs admission control of the E2E Resv onto the Agg Resv for DSCP=x. Let's assume that the Aggregate Resv for DSCP=x does NOT have sufficient unused bandwidth to support the new E2E Resv. The

Deaggregator then attempts to increase the Aggregate Reservation bandwidth for DSCP=x by sending a new Aggregate Resv with an increased bandwidth sufficient to accommodate all the E2E reservations already mapped onto that Aggregate reservation plus the new E2E reservation plus possibly some additional spare bandwidth in anticipation of additional E2E reservations to come. Assume also that the Deaggregator includes the optional Resv Confirm Request in these Aggregate Resv.

(17) The Aggregator merely complies with the received ResvConfirm Request and returns the corresponding Aggregate ResvConfirm.

(18) The Deaggregator has explicit confirmation that the Aggregate Resv has been successfully increased. The Deaggregator performs again admission control of the E2E Resv onto the increased Aggregate Reservation for DSCP=x. Assuming that the increased Aggregate Reservation for DSCP=x now has sufficient unused bandwidth and resources to support the new E2E Resv, the Deaggregator then adjusts its counter tracking the unused bandwidth on the Aggregate Reservation and forwards the E2E Resv to the Aggregator including a DCLASS object conveying the selected mapping onto DSCP=x.

(19) The Aggregator records the mapping of the E2E Resv onto DSCP=x. The Aggregator removes the DCLASS object and forwards the E2E Resv towards the sender.

References

[CSZ] Clark, D., S. Shenker, and L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," in Proc. SIGCOMM'92, September 1992.

[IP] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[HOSTREQ] Braden, R., "Requirements for Internet hosts - communication layers", STD 3, RFC 1122, October 1989.

[DSFIELD] Nichols, K., Blake, S., Baker, F. and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[PRINCIPLES] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996.

[ASSURED] Heinanen, J, Baker, F., Weiss, W. and J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, June 1999.

[BROKER] Jacobson, V., Nichols K. and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", RFC 2638, June 1999.

[BRIM] Brim, S., Carpenter, B. and F. LeFaucheur, "Per Hop Behavior Identification Codes", RFC 2836, May 2000.

[RSVP] Braden, R., Zhang, L., Berson, S., Herzog, S. and S. Jamin, "Resource Reservation Protocol (RSVP) Version 1 Functional Specification", RFC 2205, September 1997.

[TERZIS] Terzis, A., Krawczyk, J., Wroclawski, J. and L. Zhang, "RSVP Operation Over IP Tunnels", RFC 2746, January 2000.

[DCLASS] Bernet, Y., "Format of the RSVP DCLASS Object", RFC 2996, November 2000.

[INTEGRITY] Baker, F., Lindell, B. and M. Talwar, "RSVP Cryptographic Authentication", RFC 2747, January 2000.

[RFC2998] Bernet Y., Ford, P., Yavatkar, R., Baker, F., Zhang, L., Speer, M., Braden, R., Davie, B., Wroclawski, J. and E. Felstaine, "Integrated Services Operation Over Diffserv Networks", RFC 2998, November 2000.

[RFC2961] Berger, L., Gan, D., Swallow, G., Pan, P. and F. Tommasi, "RSVP Refresh Reduction Extensions", RFC 2961, April 2001.

[RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", RFC 2780, March 2000.

[RFC2711] Partridge, C. and A. Jackson, "IPv6 Router Alert Option", RFC 2711, October 1999.

[RFC2113] Katz, D. "IP Router Alert Option", RFC 2113, February 1997.

Authors' Addresses

Fred Baker
Cisco Systems
1121 Via Del Rey
Santa Barbara, CA, 93117 USA

Phone: (408) 526-4257
EMail: fred@cisco.com

Carol Iturralde
Cisco Systems
250 Apollo Drive
Chelmsford MA, 01824 USA

Phone: 978-244-8532
EMail: cei@cisco.com

Francois Le Faucheur
Cisco Systems
Domaine Green Side
400, Avenue de Roumanille
06410 Biot - Sophia Antipolis
France

Phone: +33.4.97.23.26.19
EMail: flefauch@cisco.com

Bruce Davie
Cisco Systems
250 Apollo Drive
Chelmsford MA,01824 USA

Phone: 978-244-8921
EMail: bdavie@cisco.com

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.