RSVP Refresh Overhead Reduction Extensions

Status of this Memo

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

Abstract

   This 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.

Berger, et al.   Standards Track   [Page 2]

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             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
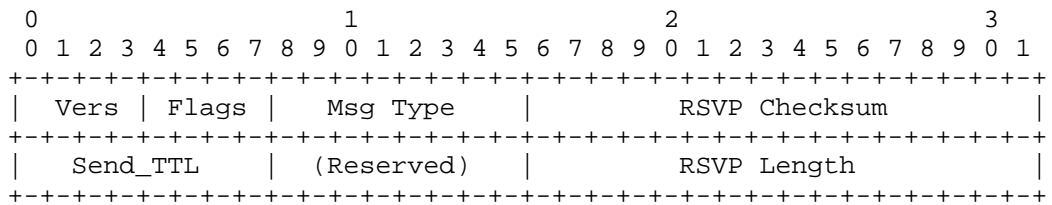
Flags: 4 bits

   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
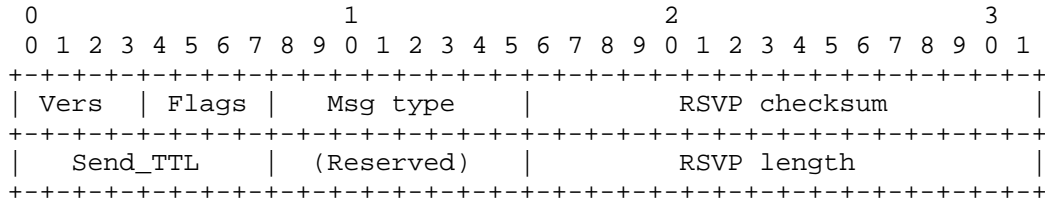with the Refresh-Reduction-Capable bit set. To cover this case,
nodes supporting the refresh overhead reduction extensions MUST
examine the flags field of each received RSVP message. If the flag
changes from indicating support to indicating non-support then,
unless configured otherwise, Srefresh messages (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..                     //
     |                                                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
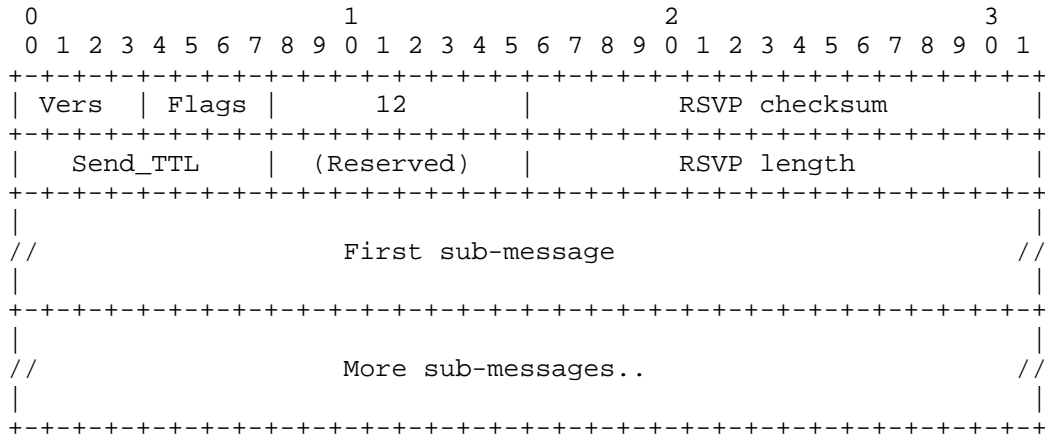
3.3. Sending RSVP Bundle Messages

   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                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
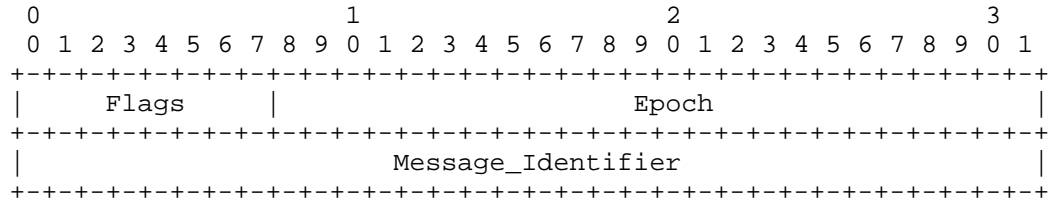
      Flags: 8 bits

         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.

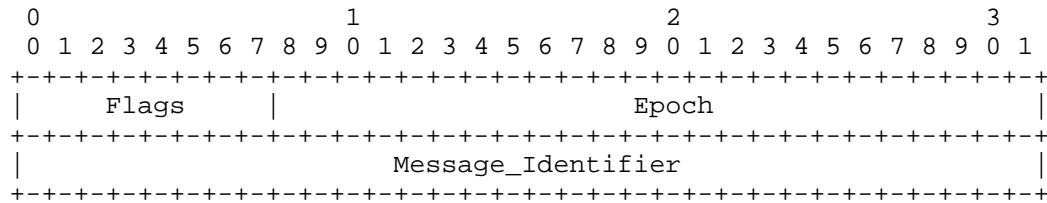4.3. MESSAGE_ID_ACK and MESSAGE_ID_NACK Objects

   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
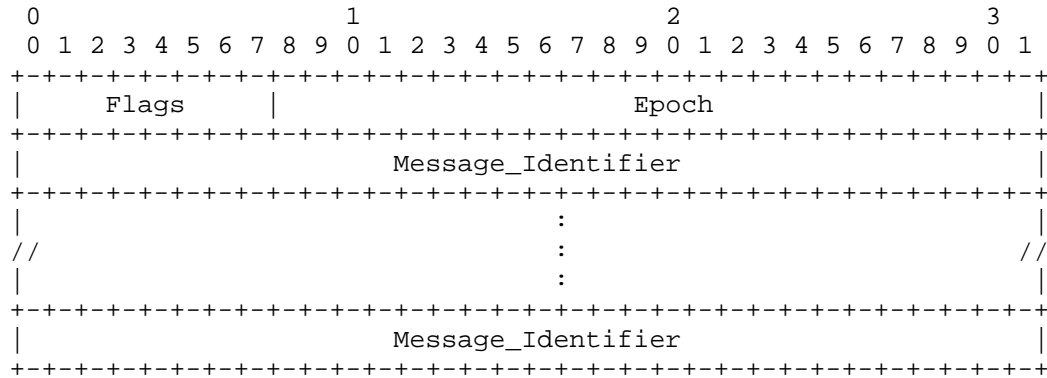
```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |    Flags      |                   Epoch                       |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                      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                   |
    |                                                               |
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
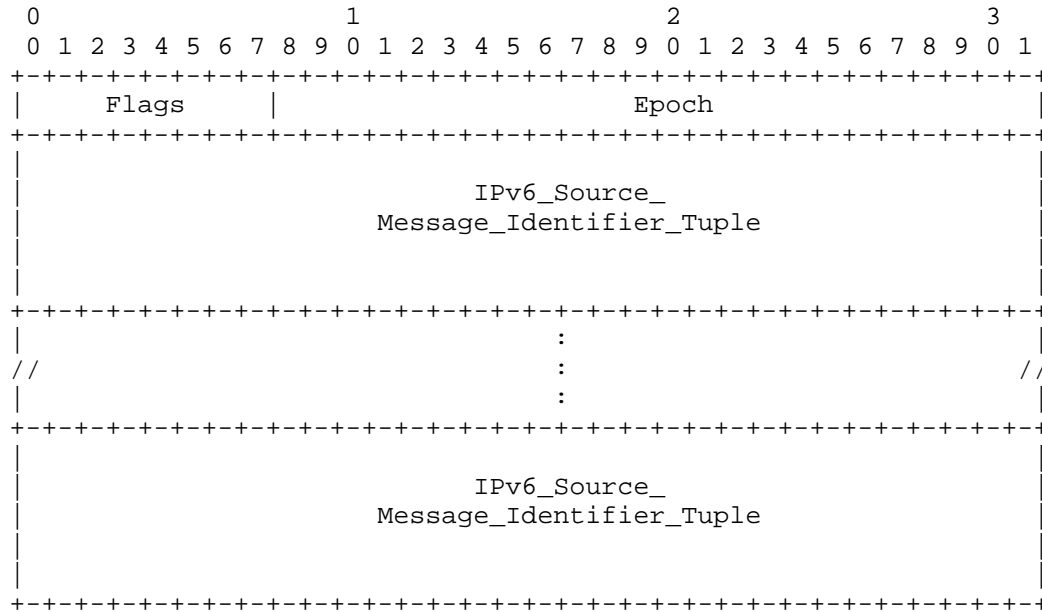
   Where a IPv6 Source_Message_Identifier_Tuple consists of:
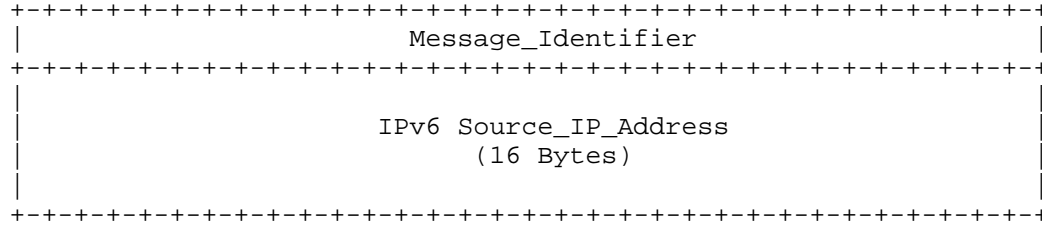
```
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                       Message_Identifier                      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    |                     IPv6 Source_IP_Address                    |
    |                          (16 Bytes)                           |
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Flags: 8 bits

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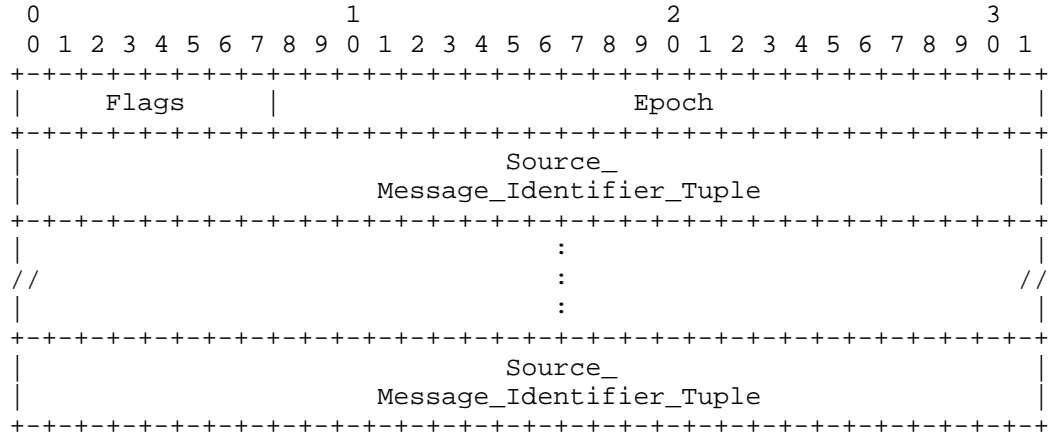   transmission and ignored on receipt.

   Epoch: 24 bits

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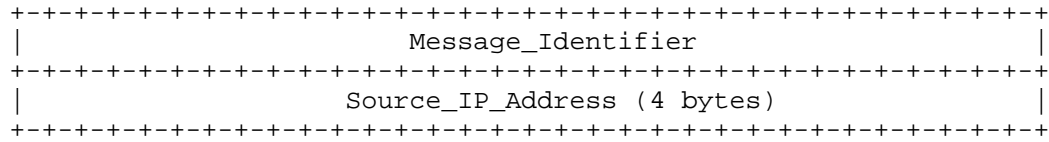   trigger message that advertised the state being refreshed.
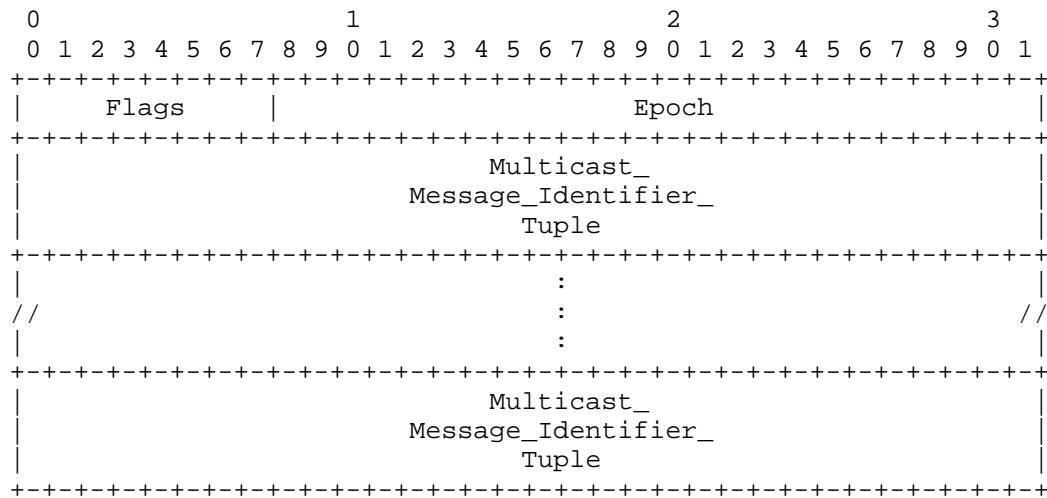
Message_Identifier

   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
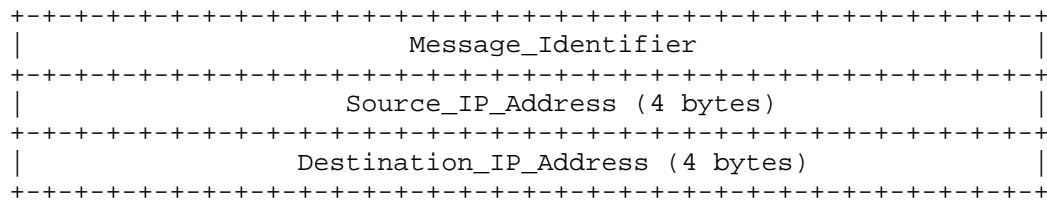   Path state being refreshed.

Source_IP_Address

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

IPv4/MESSAGE_ID MCAST_LIST object
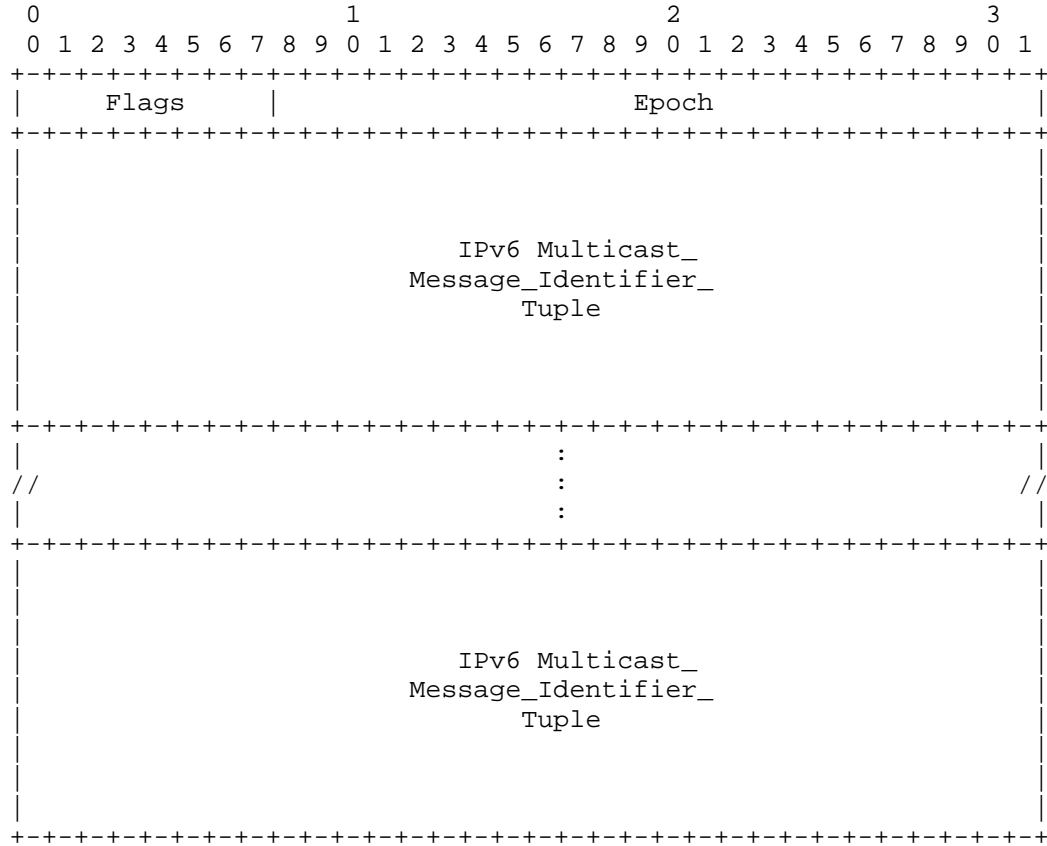
Class = MESSAGE_ID_LIST Class, C_Type = 4

```
 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