Network Working Group
Internet Draft
Expiration Date: May 1999

Daniel O. Awduche
UUNET Worldcom, Inc.

Lou Berger
FORE Systems, Inc.

Der-Hwa Gan
Juniper Networks, Inc.

Tony Li
Juniper Networks, Inc.

George Swallow
Cisco Systems, Inc.

Vijay Srinivasan
Torrent Networks, Inc.

November 1998

Extensions to RSVP for LSP Tunnels

draft-ietf-mpls-rsvp-lsp-tunnel-00.txt

Status of this Memo

Abstract

This document describes the use of RSVP, including all the necessary extensions, to establish label-switched paths (LSPs) in MPLS. Since

the flow along an LSP is completely identified by the label applied at the ingress node of the path, these paths may be treated as tunnels. A key application of LSP tunnels is traffic engineering with MPLS as specified in [3].

We propose several additional objects that extend RSVP, allowing the establishment of explicitly routed label switched paths using RSVP as a signaling protocol. The result is the instantiation of label-switched tunnels which can be automatically routed away from network failures, congestion, and bottlenecks.

Finally, we propose a number of mechanisms to reduce the refresh overhead of RSVP. The extensions can be used to reduce processing requirements of refresh messages, eliminate the state synchronization latency incurred when an RSVP message is lost and, when desired, eliminate the generation of refresh messages. An extension to support detection of when an RSVP neighbor resets its state is also presented. These extension present no backwards compatibility issues.

Contents

1. Introduction and Background

1.1. Introduction

This document is a specification of extensions to RSVP for establishing label switched paths (LSPs) in Multi-protocol Label Switching (MPLS) networks. Several of the new features described in this document were motivated by the requirements for traffic engineering over MPLS (see [3]). In particular, the extended RSVP protocol supports the instantiation of explicitly routed LSPs, with or without resource reservations. It also supports smooth rerouting of LSPs, preemption, loop detection, and a fast reroute option to allow expedited service restoration under fault conditions.

Since the traffic that flows along a label-switched path is defined by the label applied at the ingress node of the LSP, these paths can be treated as tunnels. When an LSP is used in this way we refer to it as an LSP tunnel.

LSP tunnels allow the implementation of a variety of policies related to network performance optimization. For example, LSP tunnels can be automatically or manually routed away from network failures, congestion, and bottlenecks. Furthermore, multiple parallel LSP tunnels can be established between two nodes, and traffic between the two nodes can be mapped onto the LSP tunnels according to local policy. Although traffic engineering (that is, performance optimization of operational networks) is expected to be an important application of this specification, the extended RSVP protocol can be used in a much wider context.

The purpose of this document is to describe the use of RSVP to establish LSP tunnels. The intent is to fully describe all the objects, packet formats, and procedures required to realize interoperable implementations.

All objects described in this specification are optional with respect to RSVP. This document discusses what happens when an object described here is not supported by a node.

Resilience and scalability are very important considerations in this specification. When an LSP tunnel fails, a significant amount of data can be lost. As a result, failure notification and service restoration should be fast and reliable. Accordingly, a number of features are provided to facilitate smooth reroute of LSP tunnels, fast reroute of LSP tunnels through intermediate detour paths under faults, and fast and reliable LSP tunnel teardown. A few new objects are also defined that enhance management and diagnostics of LSP tunnels.

Several new RSVP objects and messages are used to reduced processing requirements related to RSVP refresh messages and address the latency and reliability of RSVP Signaling. First, an aggregate message is proposed to reduce the message handing load. Second tokens are added as a short hand method of identifying state. Third, procedures to suppress refreshes are defined. Last a Hello protocol is defined to detect loss of a neighbor's state.

These extensions may be used in part in combination. They may be useful in other RSVP environments and may be supported independent of other MPLS related RSVP extensions.

Throughout this document, the discussion will be restricted to unicast label switched paths. Multicast LSPs are left for further study.

1.2. Background

Hosts and routers that support both RSVP [1] and Multi-Protocol Label Switching [2] can associate labels with RSVP flows. When MPLS and RSVP are combined, the definition of a flow can be made more flexible. Once a label switched path (LSP) is established, the traffic through the path is defined by the label applied at the ingress node of the LSP. The mapping of label to traffic can be accomplished using a number of different criteria. The set of packets that are assigned the same label value by a specific node are said to belong to the same forwarding equivalence class (FEC) (see [2]), and effectively define the "RSVP flow." When traffic is mapped onto a label-switched path in this way, we call the LSP an "LSP Tunnel".

When labels are associated with traffic flows, it becomes possible for a router to identify the appropriate reservation state for a packet based on the packet's label value. This approach greatly simplifies packet classification and improves network performance because a single label lookup identifies both packet forwarding information and packet reservation state.

The signaling protocol model uses downstream-on-demand label distribution. A request to bind labels to a specific LSP tunnel is initiated by an ingress node through the RSVP Path message. For this purpose, the RSVP Path message is augmented with a LABEL_REQUEST object. Labels are allocated downstream and distributed (propagated upstream) by means of the RSVP Resv message. For this purpose, the RSVP Resv message is extended with a special LABEL object. Label stacking is also supported. The procedures for label allocation, distribution, binding, and stacking are described in subsequent

sections of this document.

The signaling protocol model also supports explicit routing capability. This is accomplished by incorporating a simple EXPLICIT_ROUTE object into RSVP Path messages. The EXPLICIT_ROUTE object encapsulates a concatenation of hops which constitutes the explicitly routed path. Using this object, the paths taken by label-switched RSVP-MPLS flows can be pre-determined, independent of conventional IP routing. The explicitly routed path can be administratively specified, or automatically computed by a suitable entity based on QoS and policy requirements, taking into consideration the prevailing network state. In general, path computation can be control-driven or data-driven. The mechanisms, processes, and algorithms used to compute explicitly routed paths are beyond the scope of this specification.

One useful application of explicit routing is traffic engineering. Using explicitly routed LSPs, a node at the ingress edge of an MPLS domain can control the path through which traffic traverses from itself, through the MPLS network, to an egress node. Explicit routing can be used to optimize the utilization of network resources and enhance traffic oriented performance characteristics.

The concept of explicitly routed label switched paths can be generalized through the notion of abstract nodes. An abstract node is a group of nodes whose internal topology is opaque to the ingress node of the LSP. An abstract node is said to be trivial if it is a singleton, that is if it contains only one physical node. Using this concept of abstraction, an explicitly routed LSP can be specified as a sequence of IP prefixes with subnet masks or a sequence of Autonomous Systems.

The signaling protocol model supports the specification of an explicit path as a sequence of strict and loose routes. The combination of abstract nodes, and strict and loose routes significantly enhances the flexibility of path definitions.

An advantage of using RSVP to establish LSP tunnels is that it enables the allocation of resources along the path. For example, bandwidth can be allocated to an LSP tunnel using standard RSVP reservations and Integrated Services service classes [4].

While resource reservations are useful, they are not mandatory. Indeed, an LSP can be instantiated without any resource reservations whatsoever. Such LSPs without resource reservations can be used, for example, to carry best effort traffic. They can also be used in many other contexts, including implementation of fall-back and recovery policies under fault conditions, and so forth.

2. Overview

2.1. LSP Tunnels

According to [1], "RSVP defines a 'session' to be a data flow with a particular destination and transport-layer protocol." However, when RSVP and MPLS are combined, a flow or session can be defined with greater flexibility and generality. The ingress node of an LSP can use a variety of means to determine which packets are assigned a particular label. Once a label is assigned to a set of packets, the label effectively defines the "flow" through the LSP. We refer to such an LSP as an "LSP tunnel" because the traffic through it is opaque to intermediate nodes along the label switched path.

A new RSVP SESSION object, called LSP_TUNNEL_IPv4, has been defined to support the LSP tunnel feature. The semantics of this object, from the perspective of a node along the label switched path, is that traffic belonging to the LSP tunnel is identified solely on the basis of packets arriving from the PHOP or "previous hop" (see [1]) with the particular label value(s) assigned by this node to upstream senders to the session. In fact, the IPv4 that appears in the object name only denotes that the destination address is an IPv4 address.

2.2. Operation of LSP Tunnels

This section summarizes some of the features supported by RSVP as extended by this document related to the operation of LSP tunnels. These include: (1) the capability to establish LSP tunnels with or without QoS requirements, (2) the capability to dynamically reroute an established LSP tunnel, (3) the capability to observe the actual route traversed by an established LSP tunnel, (4) the capability to identify and diagnose LSP tunnels, (5) the capability to preempt an established LSP tunnel under administrative policy control, and (6) the capability to perform downstream-on-demand label allocation, distribution, and binding. In the following paragraphs, these features are briefly described. More detailed descriptions can be found in subsequent sections of this document.

To create an LSP tunnel, the first MPLS node on the path -- that is, the sender node with respect to the path -- creates an RSVP Path message with a session type of LSP_Tunnel_IPv4 and inserts a LABEL_REQUEST object into the Path message. The LABEL_REQUEST object indicates that a label binding for this path is requested and also provides an indication of the network layer protocol that is to be carried over this path. The reason for this is that the network layer protocol sent down an LSP cannot be assumed to be IPv4 and cannot be deduced from the L2 header, which simply identifies the higher layer

protocol as MPLS.

If the sender node has knowledge of a route that has high likelihood of meeting the tunnel's QoS requirements, or that makes efficient use of network resources, or that satisfies some policy criteria, the node can decide to use the route for some or all of its sessions. To do this, the sender node adds an EXPLICIT_ROUTE object to the RSVP Path message. The EXPLICIT_ROUTE object specifies the route as a sequence of abstract nodes.

If, after a session has been successfully established and the sender node discovers a better route, the sender can dynamically reroute the session by simply changing the EXPLICIT_ROUTE object. If problems are encountered with an EXPLICIT_ROUTE object, either because it causes a routing loop or because some intermediate routers do not support it, the sender node is notified.

By adding a RECORD_ROUTE object to the Path message, the sender node can receive information about the actual route that the LSP tunnel traverses. The sender node can also use this object to request notification from the network concerning changes to the routing path. The RECORD_ROUTE object is analogous to a path vector, and hence can be used for loop detection.

Finally, a SESSION_ATTRIBUTE object can be added to Path messages to aid in session identification and diagnostics. Additional control information, such as preemption, priority, and fast-reroute, are also included in this object.

When the EXPLICIT_ROUTE object (ERO) is present, the Path message is forwarded towards its destination along a path specified by the ERO. Each node along the path records the ERO in its path state block. Nodes may also modify the ERO before forwarding the Path message. In this case the modified ERO should be stored in the path state block.

The LABEL_REQUEST object requests intermediate routers and receiver nodes to provide a label binding for the session. If a node is incapable of providing a label binding, it sends a PathErr message with an "unknown object class" error. If the LABEL_REQUEST object is not supported end to end, the sender node will be notified by the first node which does not provide this support.

The destination node of a label-switched path responds to a LABEL_REQUEST by including a LABEL object in its response RSVP Resv message. The LABEL object is inserted in the filter spec list immediately following the filter spec to which it pertains.

The Resv message is sent back upstream towards the sender, in a

direction opposite to that followed by the Path message. Each node that receives a Resv message containing a LABEL object uses that label for outgoing traffic associated with this LSP tunnel. If the node is not the sender, it allocates a new label and places that label in the corresponding LABEL object of the Resv message which it sends upstream to the PHOP. The label sent upstream in the LABEL object is the label which this node will use to identify incoming traffic associated with this LSP tunnel. This label also serves as shorthand for the Filter Spec. The node can now update its "Incoming Label Map" (ILM), which is used to map incoming labeled packets to a "Next Hop Label Forwarding Entry" (NHLFE), see [2].

When the Resv message propagates upstream to the sender node, a label-switched path is effectively established.

2.3. Service Classes

This document does not restrict the type of Integrated Service requests for reservations. However, an implementation should support the Controlled-Load service [4].

An LSP may not need bandwidth reservations or QoS guarantees. Such LSPs can be used to deliver best-effort traffic, even if RSVP is used for setting up LSPs. When resources do not have to be allocated to the LSP, the Sender_TSpec in the Path message can specify a token bucket rate of zero and a token bucket size of zero. The corresponding FLOWSPEC (in the Resv message) should carry a zero rate and size as well. LSPs with no bandwidth reservation are not subject to Admission Control and do not require traffic policing.

2.4. Reservation Styles

The receiver node can select from among a set of possible reservation styles for each session, and each RSVP session must have a particular style. Senders have no influence on the choice of reservation style. The receiver can choose different reservation styles for different LSPs.

An RSVP session can result in one or more LSPs, depending on the reservation style chosen.

Some reservation styles, such as FF, dedicate a particular reservation to an individual sender node. Other reservation styles, such as WF and SE, can share a reservation among several sender nodes. The following sections discuss the different reservation styles and their advantages and disadvantages. A more detailed

discussion of reservation styles can be found in [1].

2.4.1. Fixed Filter (FF) Style

The Fixed Filter (FF) reservation style creates a distinct reservation for traffic from each sender that is not shared by other senders. This style is common for applications in which traffic from each sender is likely to be concurrent and independent. The total amount of reserved bandwidth on a link for sessions using FF is the sum of the reservations for the individual senders.

Because each sender has its own reservation, a unique label and a separate label-switched-path can be assigned to each sender. This can result in a point-to-point LSP between every sender/receiver pair.

2.4.2. Wildcard Filter (WF) Style

With the Wildcard Filter (WF) reservation style, a single shared reservation is used for all senders to a session. The total reservation on a link remains the same regardless of the number of senders.

A single multipoint-to-point label-switched-path is created for all senders to the session. On links that senders to the session share, a single label value is allocated to the session. If there is only one sender, the LSP looks like a normal point-to-point connection. When multiple senders are present, a multipoint-to-point LSP (a reversed tree) is created.

This style is useful for applications in which not all senders send traffic at the same time. A phone conference, for example, is an application where not all speakers talk at the same time. If, however, the reservation requested is greater than a single sender's requirements, then the reserved bandwidth on links close to the some senders may be greater than what is required. This restricts the applicability of WF for traffic engineering purposes.

Furthermore, because of the merging rules of WF, EXPLICIT_ROUTE objects cannot be used with WF reservations. As a result of this issue and the lack of applicability to traffic engineering, use of WF is not considered in this document.

2.4.3. Shared Explicit (SE) Style

The Shared Explicit (SE) style allows a receiver to explicitly specify the senders to be included in a reservation. There is a single reservation on a link for all the senders listed.

Because each sender is explicitly listed in the Resv message, separate labels may be assigned to each sender, thereby creating separate LSPs for each sender.

Having separate LSPs for each sender ensures compatibility with the EXPLICIT_ROUTE object. Path messages from different senders can carry their own ERO, and the paths taken by the senders can converge and diverge at any point in the network topology.

2.5. Rerouting LSP Tunnels

One of the requirements for Traffic Engineering is the capability to reroute an established LSP tunnel under a number of conditions, based on administrative policy. For example, in some contexts, an administrative policy may dictate that a given LSP tunnel is to be rerouted when a more "optimal" route becomes available. Another important context when LSP tunnel reroute is usually required is upon failure of a resource along the tunnel's established path. Under some policies, it may also be necessary to return the LSP tunnel to its original path when the failed resource becomes re-activated.

In general, it is highly desirable not to disrupt traffic, or adversely impact network operations while LSP tunnel rerouting is in progress. This adaptive and smooth rerouting requirement necessitates establishing a new LSP tunnel and transferring traffic from the old LSP tunnel onto it before tearing down the old LSP tunnel. This concept is called "make-before-break." A problem can arise because the old and new LSP tunnels might compete with other for resources on network segments which they have in common. Depending on availability of resources, this competition can cause Admission Control to prevent the new tunnel from being established. An advantage of using RSVP to establish LSP tunnels is that it solves this problem very elegantly.

To support make-before-break in a smooth fashion, it is necessary that on links that are common to the old and new LSPs, resources used by the old LSP tunnel should not be released before traffic is transitioned to the new LSP tunnel, and reservations should not be counted twice because this might cause Admission Control to reject the new LSP tunnel.

The combination of the LSP_TUNNEL_IPv4 SESSION object and the SE reservation style naturally achieves smooth transitions. The basic idea is that the old and new LSP tunnels share resources along links which they have in common. The LSP_TUNNEL_IPv4 SESSION object is used to narrow the scope of the RSVP session to the particular tunnel in question. To uniquely identify a tunnel, we use the combination of the destination IP address, a Tunnel ID, and the sender's IP address, which is placed in the Extended Tunnel ID field.

During the reroute operation, the source needs to appear as two different sources to RSVP. This is achieved by the inclusion of an "LSP ID", which is carried in the SENDER_TEMPLATE and FILTER_SPEC objects. Since the semantics of these objects are changed, a new C-Type is assigned.

To effect a reroute, the source node picks a new LSP ID and forms a new SENDER_TEMPLATE. The source node then creates a new ERO to define the new path. Thereafter the node sends a new Path Message using the original SESSION object and the new SENDER_TEMPLATE and ERO. It continues to use the old LSP and refresh the old Path message. On links that are not held in common, the new Path message is treated as a conventional new LSP tunnel setup. On links held in common, the shared SESSION object and SE style allow the LSP to be established sharing resources with the old LSP. Once the sender receives a Resv message for the new LSP, it can transition traffic to it and tear down the old LSP.

3. RSVP Message Formats

Five new objects are defined in this document:

Object name	Applicable RSVP messages
LABEL_REQUEST	Path
LABEL	Resv
EXPLICIT_ROUTE	Path
RECORD_ROUTE	Path, Resv
SESSION_ATTRIBUTE	Path

New C-Types are also assigned for the SESSION, SENDER_TEMPLATE, and FILTER_SPEC objects.

Detailed descriptions of the new objects are given in later sections. All new objects are optional with respect to RSVP. An implementation can choose to support a subset of objects. However, the LABEL_REQUEST and LABEL objects are mandatory with respect to this

specification.

The LABEL and RECORD_ROUTE objects, are sender specific. They must immediately follow either the SENDER_TEMPLATE in Path messages, or the FILTER_SPEC in Resv messages.

The placement of EXPLICIT_ROUTE, LABEL_REQUEST, and SESSION_ATTRIBUTE objects is simply a recommendation. The ordering of these objects is not important, so an implementation must be prepared to accept objects in any order.

3.1. Path Message

The format of the Path message is as follows:

```
<Path Message> ::=       <Common Header> [ <INTEGRITY> ]
                         <SESSION> <RSVP_HOP>
                         <TIME_VALUES>
                         [ <EXPLICIT_ROUTE> ]
                         <LABEL_REQUEST>
                         [ <SESSION_ATTRIBUTE> ]
                         [ <POLICY_DATA> ... ]
                         [ <sender descriptor> ]

<sender descriptor> ::=  <SENDER_TEMPLATE> [ <SENDER_TSPEC> ]
                         [ <ADSPEC> ]
                         [ <RECORD_ROUTE> ]
```

3.2. Resv Message

The format of the Resv message is as follows:

```
<Resv Message> ::=       <Common Header> [ <INTEGRITY> ]
                         <SESSION>  <RSVP_HOP>
                         <TIME_VALUES>
                         [ <RESV_CONFIRM> ]  [ <SCOPE> ]
                         [ <POLICY_DATA> ... ]
                         <STYLE> <flow descriptor list>

<FF flow descriptor list> ::= <FLOWSPEC> <FILTER_SPEC> <LABEL>
                         [ <RECORD_ROUTE> ]
                         | <FF flow descriptor list> <FF flow descriptor>

<FF flow descriptor> ::= [ <FLOWSPEC> ] <FILTER_SPEC> <LABEL>
                         [ <RECORD_ROUTE> ]
```

```
<SE flow descriptor> ::= <FLOWSPEC> <SE filter spec list>

<SE filter spec list> ::= <SE filter spec>
                         | <SE filter spec list> <SE filter spec>

<SE filter spec> ::=     <FILTER_SPEC> <LABEL> [ <RECORD_ROUTE> ]

Note:  LABEL and RECORD_ROUTE (if present), are bound to the
       preceding FILTER_SPEC.  No more than one LABEL and/or
       RECORD_ROUTE may follow each FILTER_SPEC.
```

4. Objects

4.1. Label Object

Labels may be carried in Resv messages. For the FF and SE styles, a label is associated with each sender. The label for a sender must immediately follow the FILTER_SPEC for that sender in the Resv message.

The LABEL object has the following format:

LABEL class = 16, 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                      (Object contents)                      //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       (top label)                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The contents of a LABEL object are a stack of labels, where each label is encoded right aligned in 4 octets. The top of the stack is in the right 4 octets of the object contents. A LABEL object that contains no labels is illegal.

Each label is an unsigned integer in the range 0 through 1048575.

The decision concerning whether to create a label stack with more than one label, when to push a new label, and when to pop the label stack are to be specified in a separate document. For

implementations that do not support a label stack, only the top label is examined. The rest of the label stack should be passed through unchanged. Such implementations are required to generate a label stack of depth 1 when initiating the first LABEL.

4.1.1. Handling Label Objects in Resv messages

A router uses the top label carried in the LABEL object as the outgoing label associated with the sender. The router allocates a new label and binds it to the incoming interface of this session/sender. This is the same interface that the router uses to forward Resv messages to the previous hops.

In MPLS a node may support multiple label spaces, perhaps associating a unique space with each incoming interface. For the purposes of the following discussion, the term "same label" means the identical label value drawn from the identical label space. Further, the following applies only to unicast sessions.

If a node receives a Resv message that has assigned the same label value to multiple senders, then that node may also assign the same value to those same senders or to any subset of those senders. Note that if a node intends to police individual senders to a session, it must assign unique labels to those senders.

Labels received in Resv messages on different interfaces are always considered to be different even if the label value is the same.

To construct a new LABEL object, the router replaces the top label (from the received Resv message) with the locally allocated new label. The router then sends the new LABEL object as part of the Resv message to the previous hop. The LABEL object should be kept in the Reservation State Block. It is then used in the next Resv refresh event for formatting the Resv message.

A router can decide to send a Resv message before its refresh timers expire if the contents of the LABEL object change.

4.1.2. Non-support of the Label Object

Under normal circumstances, a node should never receive a LABEL object in a Resv message unless it had included a LABEL_REQUEST object in the corresponding Path message. However, an RSVP router that does not recognize the LABEL object sends a ResvErr with the error code "Unknown object class" toward the receiver. This causes the reservation to fail.

RSVP is designed to cope gracefully with non-RSVP routers anywhere between senders and receivers. However, non-RSVP routers cannot receive label-switched packets conveyed in PATH or RESV messages. This means that if a router has a neighbor that is not RSVP capable, the router must not advertise the LABEL object when sending messages that pass through the non-RSVP router. The RSVP specification [1] describes how routers can determine the presence of non-RSVP routers.

4.2. Label Request Object

The LABEL_REQUEST object formats are shown below. Currently there three possible C_Types. Type 1 is a Label Request without label range. Type 2 is a label request with an ATM label range. Type 3 is a label request with a Frame Relay label range.

Label Request without Label Range

```
   class = 19, C_Type = 1  (need to get an official class num from
                            the IANA)

    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |         Length (bytes)        |   Class-Num   |    C-Type     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Reserved            |             L3PID             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Reserved

This field is reserved. It must be set to zero on transmission and must be ignored on receipt.

L3PID

an identifier of the layer 3 protocol using this path. Standard Ethertype values are used.

Label Request with ATM Label Range

class = 19, C_Type = 2 (need to get an official class num from the IANA)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Reserved            |             L3PID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Res  |    Minimum VPI        |      Minimum VCI              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Res  |    Maximum VPI        |      Maximum VCI              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Reserved (Res)

This field is reserved. It must be set to zero on transmission and must be ignored on receipt.

L3PID

an identifier of the layer 3 protocol using this path. Standard Ethertype values are used.

Minimum VPI (12 bits)

This 12 bit field specifies the lower bound of a block of Virtual Path Identifiers that is supported on the originating switch. If the VPI is less than 12-bits it should be right justified in this field and preceding bits should be set to zero.

Minimum VCI (16 bits)

This 16 bit field specifies the lower bound of a block of Virtual Connection Identifiers that is supported on the originating switch. If the VCI is less than 16-bits it should be right justified in this field and preceding bits should be set to zero.

Maximum VPI (12 bits)

This 12 bit field specifies the upper bound of a block of Virtual Path Identifiers that is supported on the originating switch. If the VPI is less than 12-bits it should be right justified in this field and preceding bits should be set to zero.

Maximum VCI (16 bits)

This 16 bit field specifies the upper bound of a block of Virtual Connection Identifiers that is supported on the originating switch. If the VCI is less than 16-bits it should be right justified in this field and preceding bits should be set to zero.

Label Request with Frame Relay Label Range
class = 19, C_Type = 3 (need to get an official class num from the IANA)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Reserved            |             L3PID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Reserved    |DLI|                     Minimum DLCI            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Reserved        |                     Maximum DLCI            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Reserved

This field is reserved. It must be set to zero on transmission and ignored on receipt.

L3PID

an identifier of the layer 3 protocol using this path. Standard Ethertype values are used.

DLI

DLCI Length Indicator. The number of bits in the DLCI. The following values are supported:

Len	DLCI bits
0	10
1	17
2	23

Minimum DLCI

This 23-bit field specifies the lower bound of a block of Data Link Connection Identifiers (DLCIs) that is supported on the originating switch. The DLCI should be right justified in this field and unused bits should be set to 0.

Maximum DLCI

This 23-bit field specifies the upper bound of a block of Data Link Connection Identifiers (DLCIs) that is supported on the originating switch. The DLCI should be right justified in this field and unused bits should be set to 0.

4.2.1. Handling of LABEL_REQUEST

To establish an LSP tunnel the sender creates a Path message with a LABEL_REQUEST object. The LABEL_REQUEST object indicates that a label binding for this path is requested and provides an indication of the network layer protocol that is to be carried over this path. This permits non-IP network layer protocols to be sent down an LSP. This information can also be useful in actual label allocation, because some reserved labels are protocol specific, see [5].

The LABEL_REQUEST should be stored in the Path State Block, so that Path refresh messages will also contain the LABEL_REQUEST object. When the Path message reaches the receiver, the presence of the LABEL_REQUEST object triggers the receiver to allocate a label and to place the label in the LABEL object for the corresponding Resv message. If a label range was specified, the label must be allocated from that range. A receiver that accepts a LABEL_REQUEST object MUST include a LABEL object in Resv messages pertaining to that Path message. If a LABEL_REQUEST object was not present in the Path message, a node MUST NOT include a LABEL object in a Resv message for that Path message's session and PHOP.

A node that sends a LABEL_REQUEST object must be ready to accept and correctly process a LABEL object in the corresponding Resv messages.

A node that recognizes a LABEL_REQUEST object, but that is unable to support it (possibly because of a failure to allocate labels) should send a PathErr with the error code "Routing problem" and the subcode "MPLS label allocation failure." This includes the case where a label range has been specified and a label cannot be allocated from that range.

If the receiver cannot support the protocol L3PID, it should send a PathErr with the error code "Routing problem" and the subcode "Unsupported L3PID." This causes the RSVP session to fail.

4.2.2. Non-support of the Label Request Object

An RSVP router that does not recognize the LABEL_REQUEST object sends a PathErr with the error code "Unknown object class" toward the sender. An RSVP router that recognizes the LABEL_REQUEST object but does not recognize the C_Type send a PathErr with the error code "Unknown object C_Type" toward the sender. This causes the path setup to fail. The sender should notify management that a LSP cannot be established and possibly take action to continue the reservation without the LABEL_REQUEST.

RSVP is designed to cope gracefully with non-RSVP routers anywhere between the sender and the receiver. However, non-RSVP routers cannot receive label-switched packets. This means that if a router has a neighbor that is not RSVP capable, the router must not advertise LABEL_REQUEST objects when sending messages that pass through the non-RSVP routers. The router should send a PathErr back to the sender, with the error code "Routing problem" and the subcode "MPLS being negotiated, but a non-RSVP capable router stands in the path." See [1] for a description of how routers can determine the presence of non-RSVP routers.

4.3. Explicit Route Object

As stated earlier, explicit routes are to be specified through a new EXPLICIT_ROUTE object (ERO) in RSVP Path messages. The EXPLICIT_ROUTE object has the following format:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                      (Object contents)                      //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class-Num

The Class-Num for an EXPLICIT_ROUTE object is 18 (need to get an official one from the IANA with the high order two bits set to 11)

C-Type

The C-Type for an EXPLICIT_ROUTE object is 2 (need to get an official one from the IANA)

If a Path message contains multiple EXPLICIT_ROUTE objects, only the first object is meaningful. Subsequent EXPLICIT_ROUTE objects may be ignored and should not be propagated.

4.3.1. Applicability

The EXPLICIT_ROUTE object is intended to be used only for unicast situations. Applications of explicit routing to multicast are a topic for further research.

The EXPLICIT_ROUTE object is to be used only when all routers along the explicit route support RSVP and the EXPLICIT_ROUTE object. The mechanisms for determining, a priori, that such support is present are beyond the scope of this document.

4.3.2. Semantics of the Explicit Route Object

An explicit route is a particular path in the network topology. Typically, the explicit route is determined by a node, with the intent of directing traffic along that path.

An explicit route is described as a list of groups of nodes along the explicit route. Certain operations to be performed along the path can also be encoded in the EXPLICIT_ROUTE object.

In addition to the ability to identify specific nodes along the path, an explicit route can identify a group of nodes that must be traversed along the path. This capability allows the routing system a significant amount of local flexibility in fulfilling a request for an explicit route. This capability allows the generator of the explicit route to have imperfect information about the details of the path.

The explicit route is encoded as a series of subobjects contained in an EXPLICIT_ROUTE object. Each subobject may identify a group of nodes in the explicit route or may specify an operation to be performed along the path. An explicit route then becomes a specification of groups of nodes to be traversed and a set of operations to be performed along the path.

To formalize the discussion, we call each group of nodes an abstract node. Thus, we say that an explicit route is a specification of a set of abstract nodes to be traversed and a set operations to be performed along that path. If an abstract node consists of only one node, we refer to it as a simple abstract node.

As an example of the concept of abstract nodes, consider an explicit route that consists solely of Autonomous System number subobjects. Each subobject corresponds to an Autonomous System in the global topology. In this case, each Autonomous System is an abstract node, and the explicit route is a path that includes each of the specified Autonomous Systems. There may be multiple hops within each Autonomous System, but these are opaque to the source node for the explicit route.

4.3.3. Subobjects

The contents of an EXPLICIT_ROUTE object are a series of variable-length data items called subobjects. Each subobject has the form:

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-------------//----------------+
|L|    Type     |     Length    | (Subobject contents)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-------------//----------------+
```

L

The L bit is an attribute of the subobject. The L bit is set if the subobject represents a loose hop in the explicit route. If the bit is not set, the subobject represents a strict hop in the explicit route.

Type

The Type indicates the type of contents of the subobject. Currently defined values are:

```
 0   Reserved
 1   IPv4 prefix
 2   IPv6 prefix
32   Autonomous system number
64   MPLS label switched path termination
```

Length

The Length contains the total length of the subobject in bytes, including the L, Type and Length fields. The Length must be at least 4, and must be a multiple of 4.

4.3.3.1. Strict and Loose Subobjects

The L bit in the subobject is a one-bit attribute. If the L bit is set, then the value of the attribute is 'loose.' Otherwise, the value of the attribute is 'strict.' For brevity, we say that if the value of the subobject attribute is 'loose' then it is a 'loose subobject.' Otherwise, it's a 'strict subobject.' Further, we say that the abstract node of a strict or loose subobject is a strict or a loose node, respectively. Loose and strict nodes are always interpreted relative to their prior abstract nodes.

The path between a strict node and its preceding node MUST include only network nodes from the strict node and its preceding abstract node.

The path between a loose node and its preceding node MAY include other network nodes that are not part of the strict node or its preceding abstract node.

The L bit has no meaning in operation subobjects.

4.3.3.2. **Subobject 1:** IPv4 prefix

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type     |     Length    | IPv4 address (4 bytes)        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv4 address (continued)      |     Mask      |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

0x81 IPv4 address

IPv4 address

An IPv4 address. This address is treated as a prefix based on the mask value below. Bits beyond the mask are ignored and should be set to zero.

Length

The Length contains the total length of the subobject in bytes, including the Type and Length fields. The Length is always 8.

Mask

Length in bits of the IPv4 prefix

Padding

Zero on transmission. Ignored on receipt.

The contents of an IPv4 prefix subobject are a 4-octet IPv4 address, a 1-octet prefix length, and a 1-octet pad. The abstract node represented by this subobject is the set of nodes that have an IP

address which lies within this prefix. Note that a prefix length of 32 indicates a single IPv4 node.

4.3.3.3. Subobject 2: IPv6 Prefix

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type     |     Length    | IPv6 address (16 bytes)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv6 address (continued)                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv6 address (continued)                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv6 address (continued)                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv6 address (continued)      |    Mask       |  Padding      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

0x82 IPv6 address

Length

The Length contains the total length of the subobject in bytes, including the Type and Length fields. The Length is always 20.

IPv6 address

An IPv6 address. This address is treated as a prefix based on the mask value below. Bits beyond the mask are ignored and should be set to zero.

Mask

Length in bits of the IPv6 prefix.

Padding

Zero on transmission. Ignored on receipt.

The contents of an IPv6 prefix subobject are a 16-octet IPv6 address, a 1-octet prefix length, and a 1-octet pad. The abstract node represented by this subobject is the set of nodes that have an IP

address which lies within this prefix. Note that a prefix length of 128 indicates a single IPv6 node.

4.3.3.4. **Subobject 32:** Autonomous System Number

The contents of an Autonomous System (AS) number subobject are a 2-octet AS number. The abstract node represented by this subobject is the set of nodes belonging to the autonomous system.

The length of the AS number subobject is 4 octets.

4.3.3.5. **Subobject 64:** MPLS Label Switched Path Termination

The contents of an MPLS label switched path termination subobject are 2 octets of padding. This subobject is an operation subobject. This object is only meaningful if there is a LABEL_REQUEST object in the Path message.

If a LABEL_REQUEST object is present in the Path message, this Path message is being used to establish a label-switched path. In this case, this subobject indicates that the prior abstract node should remove one level of label from all packets following this label-switched path.

The length of the MPLS label termination subobject is 4 octets.

4.3.4. Processing of the Explicit Route Object

4.3.4.1. Selection of the Next Hop

A node receiving a Path message containing an EXPLICIT_ROUTE object must determine the next hop for this path. This is necessary because the next abstract node along the explicit route might be an IP subnet or an Autonomous System. Therefore, selection of this next hop may involve a decision from a set of feasible alternatives. The criteria used to make a selection from feasible alternatives is implementation dependent and can also be impacted by local policy, and is beyond the scope of this specification. However, it is assumed that each node will make a best effort attempt to determine a loop-free path. Note that paths so determined can be overridden by local policy.

To determine the next hop for the path, a node performs the following steps:

1) The node receiving the RSVP message must first evaluate the first

subobject. If the node is not part of the abstract node described by the first subobject, it has received the message in error and should return a "Bad initial subobject" error. If the first subobject is an operation subobject, the message is in error and the system should return a "Bad EXPLICIT_ROUTE object" error. If there is no first subobject, the message is also in error and the system should return a "Bad EXPLICIT_ROUTE object" error.

2) If there is no second subobject, this indicates the end of the explicit route. The EXPLICIT_ROUTE object should be removed from the Path message. This node may or may not be the end of the path. Processing continues with section 4.3.4.2, where a new EXPLICIT_ROUTE object may be added to the Path message.

3) Next, the node evaluates the second subobject. If the subobject is an operation subobject, the node records the subobject, deletes it from the EXPLICIT_ROUTE object and continues processing with step 2, above. Note that this changes the third subobject into the second subobject in subsequent processing. The precise operations to be performed by this node must be defined by the operation subobject.

4) If the node is also a part of the abstract node described by the second subobject, then the node deletes the first subobject and continues processing with step 2, above. Note that this makes the second subobject into the first subobject of the next iteration.

5) The node determines whether it is topologically adjacent to the abstract node described by the second subobject. If so, the node selects a particular next hop which is a member of the abstract node. The node then deletes the first subobject and continues processing with section 4.3.4.2.

6) Otherwise, the node selects a next hop within the abstract node of the first subobject that is along the path to the abstract node of the second subobject. If no such path exists then there are two cases:

6a) If the second subobject is a strict subobject, there is an error and the node should return a "Bad strict node" error.

6b) Otherwise, if the second subobject is a loose subobject, the node selects any next hop that is along the path to the next abstract node. If no path exists, there is an error, and the node should return a "Bad loose node" error.

7) Finally, the node replaces the first subobject with any subobject that denotes an abstract node containing the next hop. This is necessary so that when the explicit route is received by the next

hop, it will be accepted.

4.3.4.2. Adding subobjects to the Explicit Route Object

After selecting a next hop, the node may alter the explicit route in the following ways.

If, as part of executing the algorithm in section 4.3.4.1, the EXPLICIT_ROUTE object is removed, the node may add a new EXPLICIT_ROUTE object.

Otherwise, if the node is a member of the abstract node for the first subobject, a series of subobjects may be inserted before the first subobject or may replace the first subobject. Each subobject in this series must denote an abstract node that is a subset of the current abstract node.

Alternately, if the first subobject is a loose subobject, an arbitrary series of subobjects may be inserted prior to the first subobject.

4.3.5. Loops

While the EXPLICIT_ROUTE object is of finite length, the existence of loose nodes implies that it is possible to construct forwarding loops during transients in the underlying routing protocol. This can be detected by the originator of the explicit route through the use of another opaque route object called the RECORD_ROUTE object. The RECORD_ROUTE object is used to collect detailed path information and is useful for loop detection and for diagnostics.

4.3.6. Non-support of the Explicit Route Object

An RSVP router that does not recognize the EXPLICIT_ROUTE object sends a PathErr with the error code "Unknown object class" toward the sender. This causes the path setup to fail. The sender should notify management that a LSP cannot be established and possibly take action to continue the reservation without the EXPLICIT_ROUTE or via a different explicit route.

4.4. Record Route Object

The format of the RECORD_ROUTE object (RRO) is as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                        (Subobjects)                         //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class-Num

The Class-Num for a RECORD_ROUTE object is 194 (need to get an official one from the IANA with the high order two bits set to 11)

C-Type

The C-Type for a RECORD_ROUTE object is 1 (need to get an official one from the IANA)

The RRO can be present in both RSVP Path and Resv messages. If a message contains multiple RROs, only the first RRO is meaningful. Subsequent RROs can be ignored and should not be propagated.

4.4.1. Subobjects

The contents of a RECORD_ROUTE object are a series of variable-length data items called subobjects. Each subobject has its own Length field. The length contains the total length of the subobject in bytes, including the Type and Length fields. The length must always be a multiple of 4, and at least 4.

Subobjects are organized as a last-in-first-out stack. The first subobject relative to the beginning of RRO is considered the top. The last subobject is considered the bottom. When a new subobject is added, it is always added to the top.

An empty RRO with no subobjects is considered illegal.

Two kinds of subobjects are currently defined.

4.4.1.1. Subobject 1: IPv4 address

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type     |     Length    | IPv4 address (4 bytes)        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv4 address (continued)      |     Mask      |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

0x81 IPv4 address

IPv4 address

A 32-bit unicast, host address. Any network-reachable interface address is allowed here. Illegal addresses, such as loopback addresses, should not be used.

Length

The Length contains the total length of the subobject in bytes, including the Type and Length fields. The Length is always 8.

Mask

32

Padding

Zero on transmission. Ignored on receipt.

4.4.1.2. Subobject 2: IPv6 address

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type     |     Length    | IPv6 address (16 bytes)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv6 address (continued)                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv6 address (continued)                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv6 address (continued)                                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| IPv6 address (continued)      |   Mask        |  Padding      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

0x82 IPv6 address

Length

The Length contains the total length of the subobject in bytes, including the Type and Length fields. The Length is always 20.

IPv6 address

A 128-bit unicast host address.

Mask

128

Padding

Zero on transmission. Ignored on receipt.

4.4.2. Applicability

Only the procedures for use in unicast sessions are defined here.

There are three possible uses of RRO in RSVP. First, an RRO can function as a loop detection mechanism to discover L3 routing loops, or loops inherent in the explicit route. The exact procedure for doing so is described later in this document.

Second, an RRO collects up-to-date detailed path information hop-by-hop about RSVP sessions, providing valuable information to the sender or receiver. Any path change (due to network topology changes) is quickly reported.

Third, RRO syntax is designed so that, with minor changes, the whole object can be used as input to the EXPLICIT_ROUTE object. This is useful if the sender receives RRO from the receiver in a Resv message, applies it to EXPLICIT_ROUTE object in the next Path message in order to "pin down session path".

4.4.3. Handling RRO

Typically, a node initiates an RSVP session by adding the RRO to the Path message. The initial RRO contains only one subobject - the sender's IP addresses.

When a Path message containing an RRO is received by an intermediate router, the router stores a copy of it in the Path State Block. The RRO is then used in the next Path refresh event for formatting Path messages. When a new Path message is to be sent, the router adds a new subobject to the RRO and appends the resulting RRO to the Path message before transmission.

The newly added subobject must be this router's IP address. The address to be added should be the interface address of the outgoing Path messages. If there are multiple addresses to choose from, the decision is a local matter. However, it is recommended that the same address be chosen consistently. If the newly added subobject causes the RRO to be too big to fit in a Path message, the Path message shall be dropped and a PathErr message should be sent back to the sender.

An RSVP router can decide to send Path messages before its refresh time if the RRO in the next Path message is different from the previous one. This can happen if the contents of the RRO received from the previous hop router changes or if this RRO is newly added to (or deleted from) the Path message.

When the destination node of an RSVP session receives a Path message with an RRO, this indicates that the sender node needs route recording. The destination node initiates the RRO process by adding an RRO to Resv messages. The processing mirrors that of the Path messages. The only difference is that the RRO in a Resv message records the path information in the reverse direction.

Note that each node along the path will now have the complete route

from source to destination. The Path RRO will have the route from the source to this node; the Resv RRO will have the route from this node to the destination. This is useful for network management.

A received Path message without an RRO indicates that the sender node no longer needs route recording. Subsequent Path messages and Resv messages shall not contain an RRO.

4.4.4. Loop Detection

As part of processing an incoming RRO, an intermediate router looks into all subobjects contained within the RRO. If the router determines that it is already in the list, a forwarding loop exists.

An RSVP session is loop-free if downstream nodes receive Path messages or upstream nodes receive Resv messages with no routing loops detected in the contained RRO.

There are two broad classifications of forwarding loops. The first class is the transient loop, which occurs as a normal part of operations as L3 routing tries to converge on a consistent forwarding path for all destinations. The second class of forwarding loop is the permanent loop, which normally results from network mis-configuration.

The action performed by a node on receipt of an RRO depends on the message type in which the RRO is received.

For Path messages containing a forwarding loop, the router builds and sends a "Routing problem" PathErr message, with the subcode "loop detected," and drops the Path message. Until the loop is eliminated, this session is not suitable for forwarding data packets. How the loop eliminated is beyond the scope of this document.

For Resv messages containing a forwarding loop, the router simply drops the message. Resv messages should not loop if Path messages do not loop.

4.4.5. Non-support of RRO

An RSVP router that does not recognize the RRO forwards it unchanged. This has no impact on the reservation. The presence of non-RSVP routers anywhere between senders and receivers has no impact on the object either. The worst result is that the RRO does not reflect the full path information.

4.5. Error Subcodes for ERO and RRO

In the processing described above, certain errors must be reported as part of a "Routing Problem" PathErr message. The value of the "Routing Problem" error code is 24 (TBD).

The following defines the subcodes for the routing problem PathErr message:

```
Value Error:

1      Bad EXPLICIT_ROUTE object

2      Bad strict node

3      Bad loose node

4      Bad initial subobject

5      No route available toward destination

6      RRO syntax error detected

7      RRO indicated routing loops

8      MPLS being negotiated, but a non-RSVP-capable router
       stands in the path

9      MPLS label allocation failure

10     Unsupported L3PID
```

4.6. Session, Sender Template, and Filter Spec Objects

New C-Types are defined for the SESSION, SENDER_TEMPLATE and FILTER_SPEC objects. The LSP_TUNNEL_IPv4 objects have the following format:

4.6.1. Session Object

Class = SESSION, C-Type = LSP_TUNNEL_IPv4 (7)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   IPv4 tunnel end point address               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Must be zero                 |      Tunnel ID                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Extended Tunnel ID                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 tunnel end point address

IPv4 address of the destination node for the tunnel.

Tunnel ID

A 16-bit identifier used in the SESSION that remains constant over the life of the tunnel.

Extended Tunnel ID

A 32-bit identifier used in the SESSION that remains constant over the life of the tunnel. Normally set to all zeros. Source nodes that wish to narrow the scope of a SESSION to the source-destination pair may place their IPv4 address here as a globally unique identifier.

4.6.2. Sender Template Object

Class = SENDER_TEMPLATE, C-Type = LSP_TUNNEL_IPv4 (7)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   IPv4 tunnel sender address                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Must be zero                 |            LSP ID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 tunnel sender address

IPv4 address for a sender node

LSP ID

A 16-bit identifier used in the SENDER_TEMPLATE and the FILTER_SPEC that can be changed to allow a sender to share resources with itself.

4.6.3. Filter Specification Object

Class = FILTER SPECIFICATION, C-Type = LSP_TUNNEL_IPv4 (7)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   IPv4 tunnel sender address                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Must be zero                 |            LSP ID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv4 tunnel sender address

IPv4 address for a sender node

LSP ID

A 16-bit identifier used in the SENDER_TEMPLATE and the FILTER_SPEC that can be changed to allow a sender to share resources with itself.

4.6.4. Reroute Procedure

This section describes how to setup a tunnel that is capable of maintaining resource reservations (without double counting) while it is being rerouted or while it is attempting to increase its bandwidth. In the initial Path message, the source node forms a SESSION object, assigns a Tunnel_ID, and places its IPv4 address in the Extended_Tunnel_ID. It also forms a SENDER_TEMPLATE and assigns a Tunnel_Path_ID. Tunnel setup then proceeds according to the normal procedure.

On receipt of the Path message, the destination node sends a Resv message with the STYLE Shared Explicit to the source.

[Note: I think we should add a flag to the SESSION_ATTRIBUTE for the source to indicate that it wishes the SE style.]

When a source node with an established path wants to change that path, it forms a new Path message as follows. The existing SESSION object is used. In particular the Tunnel_ID and Extended_Tunnel_ID are unchanged. The source node picks a new Tunnel_Path_ID to form a new SENDER_TEMPLATE. It creates an EXPLICIT_ROUTE object for the new route. The new Path message is sent. The source node refreshes both the old and new path messages

The destination node responds with a Resv message with an SE flow descriptor formatted as:

```
<FLOW_SPEC><old_FILTER_SPEC><old_LABEL_OBJECT><new_FILTER_SPEC>
<new_LABEL_OBJECT>
```

(Note that if the PHOPs are different, then two messages are sent each with the appropriate FILTER_SPEC and LABEL_OBJECT.)

When the Source node receives the Resv Message(s), it may begin using the new route. It should send a PathTear message for the old route.

4.7. Session Attribute Object

The format of the SESSION_ATTRIBUTE object is as follows:

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Length (bytes)        |   Class-Num   |    C-Type     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Setup Prio  | Holding Prio  |     Flags     |  Name Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//          Session Name      (NULL padded display string)      //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Class-Num

The Class-Num indicates that the object is 207. (TBD)

C-Type

The C-Type is 7.

Flags

0x01 = Fast-reroute
This flag permits transit routers to pre-compute and pre-establish detour paths for this session. When a fault is detected on an adjacent downstream link or node, a transit router can reroute traffic onto the detour path for fast service restoration.

0x02 = Merging permitted
This flag permits transit routers to merge this session with other RSVP sessions for the purpose of reducing resource overhead on downstream transit routers, thereby providing better network scalability.

0x04 = Tunnel head may reroute
This flag indicates that the head end of the tunnel may choose to reroute this tunnel without tearing it down. A tunnel tail SHOULD use the SE Style when responding with a Resv message.

Setup Priority
The priority of the session with respect to taking resources, in the range of 0 to 7. The value 0 is the highest priority. The Setup Priority is used in deciding whether this session can preempt another session.

Holding Priority
The priority of the session with respect to holding resources, in the range of 0 to 7. The value 0 is the highest priority. Holding Priority is used in deciding whether this session can be preempted by another session.

Name Length

The length of the display string before padding, in bytes.

Session Name

A null padded string of characters.

The support of setup and holding priorities is optional. A node can

recognize this information but be unable to perform the requested operation. The node should pass the information downstream unchanged.

As noted above, preemption is implemented by two priorities. The Setup Priority is the priority for taking resources. The Holding Priority is the priority for holding a resource. Specifically, the Holding Priority is the priority at which resources assigned to this session will be reserved. The Setup Priority should never be higher than the Holding Priority for a given session.

When a new reservation is considered for admission, the bandwidth requested is compared with the bandwidth available at the priority specified in the Setup Priority. The bandwidth available at a particular Setup Priority is the unused bandwidth plus the bandwidth reserved at all Holding Priorities lower than the Setup Priority.

If the requested bandwidth is not available a PathErr message is returned with an Error Code of 01, Admission Control Failure, and an Error Value of 0x0002. The first 0 in the Error Value indicates a globally defined subcode and is not informational. The 002 indicates "requested bandwidth unavailable".

If the requested bandwidth is less than the unused bandwidth then processing is complete. If the requested bandwidth is available, but is in use by lower priority sessions, then lower priority sessions (beginning with the lowest priority) can be pre-empted to free the necessary bandwidth.

When pre-emption is supported, each pre-empted reservation triggers a TC_Preempt() upcall to local clients, passing a subcode that indicates the reason. A ResvErr and/or PathErr with the code "Policy Control failure" should be sent toward the downstream receivers and upstream senders.

The support of fast-reroute is optional. A node may recognize the fast-reroute Flag but may be unable to perform the requested operation. In this case, the node should pass the information downstream unchanged.

The support of merging is optional. A node may recognize the Merge Flag but may be unable to perform the requested operation. In this case, the node should pass the information downstream unchanged.

If a Path message contains multiple SESSION_ATTRIBUTE objects, only the first SESSION_ATTRIBUTE object is meaningful. Subsequent SESSION_ATTRIBUTE objects can be ignored and need not be forwarded.

The contents of the Session Name field are a string, typically of displayable characters. The Length must always be a multiple of 4 and must be at least 8. For an object length that is not a multiple of 4, the object is padded with trailing NULL characters. The Name Length field contains the actual string length.

All RSVP routers, whether they support the SESSION_ATTRIBUTE object or not, shall forward the object unmodified. The presence of non-RSVP routers anywhere between senders and receivers has no impact on this object.

5. Refresh Related Extensions

The resource requirement (in terms of cpu processing and memory) for running RSVP on a router increases proportionally with the number of sessions. Supporting a large number of sessions can present scaling problems.

This section describes an approach to help alleviate one of the scaling issues. RSVP Path and Resv messages must be periodically refreshed to maintain state. The approach described here simply reduces the volume of messages which must be periodically sent and received.

One way to address the refresh volume problem is to increase the refresh timer R. Increasing the value of R provides linear improvement on transmission overhead, but at the cost of increasing refresh timeout.

An aggregate message is proposed which can reduce R for faster detection of connectivity problems and still reduce overhead by an order of magnitude.

A Message_ID object is defined to reduce refresh message processing by allowing the receiver to immediately identify an unchanged message. A Message_ACK object is defined which used in combination with the Message_ID object may suppress refreshes altogether.

Finally, a hello protocol is defined to allow detection of the loss of a neighbor.

5.1. RSVP Aggregate Message

An RSVP aggregate message consists of an aggregate header followed by a body consisting of a variable number of standard RSVP messages. The following subsections define the formats of the aggregate header and the rules for including standard RSVP messages as part of the message.

5.1.1. Aggregate 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 aggregate header is identical to the format of the RSVP common header [1]. The fields in the header are as follows:

Vers: 4 bits

Protocol version number. This is version 1.

Flags: 4 bits

0x01: Aggregate capable

If set, indicates to RSVP neighbors that this node is willing and capable of receiving aggregate messages. This bit is meaningful only between adjacent RSVP neighbors.

0x02-0x08: Reserved

Msg type: 8 bits

12 = Aggregate

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 submessages carry their own checksum as well as the INTEGRITY object for authentication, this field MAY be set 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 IP TTL that an Aggregate message sent to the TTL in the received message.

RSVP length: 16 bits

The total length of this RSVP aggregate message in bytes, including the aggregate header and the submessages that follow.

5.1.2. Message Formats

An RSVP aggregate message must contain at least one submessage. A submessage is one of the RSVP Path, PathTear, PathErr, Resv, ResvTear, ResvErr, or ResvConf messages.

Empty RSVP aggregate messages should not be sent. It is illegal to include another RSVP aggregate message as a submessage.

```
 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 submessage                         //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                    More submessage...                       //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.1.3. Sending RSVP Aggregate Messages

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

RSVP Aggregate messages should not be used if the next-hop RSVP

neighbor does not support RSVP Aggregate messages. Methods for discovering such information include: (1) manual configuration and (2) observing the Aggregate-capable bit (see the description that follows) in the received RSVP messages.

Support for RSVP Aggregate messages is optional. While message aggregation might help in scaling RSVP, and in reducing processing overhead and bandwidth consumption, a node is not required to transmit every standard RSVP message in an Aggregate message. A node must always be ready to receive standard RSVP messages.

The IP source address is local to the system that originated the Aggregate message. The IP destination address is the next-hop node for which the submessages are intended. These addresses need not be identical to those used if the submessages were sent as standard RSVP messages.

For example, the IP source address of Path and PathTear messages is the address of the sender it describes, while the IP destination address is the DestAddress for the session. These end-to-end addresses are overridden by hop-by-hop addresses while encapsulated in an Aggregate message. These addresses can easily be restored from the SENDER_TEMPLATE and SESSION objects within Path and PathTear messages. For Path and PathTear messages, the next-hop node can be learned by looking up DestAddress in the forwarding table.

RSVP Aggregate messages do not require the Router Alert IP option [RFC 2113] in their IP headers. This is because Aggregate messages are addressed directly to RSVP neighbors.

Each RSVP Aggregate message must occupy exactly one IP datagram. If it exceeds the MTU, the datagram is fragmented by IP and reassembled at the recipient node. A single RSVP Aggregate message cannot exceed the maximum IP datagram size, which is approximately 64K bytes.

5.1.4. Receiving RSVP Aggregate Messages

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

The receiver next compares the IP TTL with which an Aggregate message is sent to the 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 aggregate 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. Each sub-message is processed according to procedures specified in RFC 2209.

5.1.5. Forwarding RSVP Aggregate Messages

When an RSVP router receives an Aggregate messages which is not addressed to one of it's IP addresses, it SHALL forward the message. Non-RSVP routers should treat RSVP Aggregate messages as any other IP datagram.

When individual submessages are being forwarded, they can be encapsulated in another aggregate message before sending to the next-hop neighbor. The Send_TTL field in the submessages should be decremented properly before transmission.

5.1.6. Aggregate-Capable Bit

To support message aggregation, an additional capability bit is added to the common RSVP header, which is defined in RFC2205 [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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Vers | Flags |   Msg Type    |         RSVP Checksum         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Send_TTL    |  (Reserved)   |         RSVP Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Flags: 4 bits

0x01: Aggregate capable

If set, indicates to RSVP neighbors that this node is willing and capable of receiving aggregate messages. This bit is meaningful only between adjacent RSVP neighbors.

5.2. MESSAGE_ID Extension

Within the MESSAGE_ID Class there are two object types defined. The two object types are the MESSAGE_ID object and the MESSAGE_ID ACK object. The MESSAGE_ID Class is used to support acknowledgments and to indicate when refresh messages are not needed after an acknowledgment. When refreshes are normally generated, the MESSAGE_ID object can also be used to simply provide a shorthand indication of when a message represents new state. Such information can be used on the receiving node to reduce refresh processing requirements.

Message identification and acknowledgment is done on a hop-by-hop basis. Acknowledgment is handled independent of SESSION or message type. Both types of MESSAGE_ID objects contain a message identifier. The identifier MUST be unique on a per source IP address basis across messages sent by an RSVP node and received by a particular node. No more than one MESSAGE_ID object may be included in an RSVP message. Each message containing an MESSAGE_ID object may be acknowledged via a MESSAGE_ID ACK object. MESSAGE_ID ACK objects may be sent piggybacked in unrelated RSVP messages or in RSVP ACK messages

Either type of MESSAGE_ID object contained in an aggregate sub-message. When so included the object is treated as if it were contained in a standard, unaggregated, RSVP message. Only one MESSAGE_ID object MAY be included in a (sub)message and it MUST follow any present MESSAGE_ID ACK objects. When no MESSAGE_ID ACK 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 the (sub)message header.

When present, one or more MESSAGE_ID ACK objects MUST immediately follow the INTEGRITY object. When no INTEGRITY object is present, the MESSAGE_ID ACK objects MUST immediately follow the the (sub)message header. An MESSAGE_ID ACK object may only be included in a message when the message's IP destination address matches the unicast address of the node that generated the message(s) being acknowledged.

5.2.1. MESSAGE_ID Object

MESSAGE_ID Class = TBD. (Value TBD of form 10bbbbbb)

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     |                  Message ID                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Flags: 8 bits

0x08 = ACK_Desired flag

Indicates that the sender is willing to accept a message acknowledgment. Acknowledgments MUST be silently ignored when they are sent in response to messages whose ACK_Desired flag is not set. This flag MUST be set when the Last_Refresh flag is set.

0x04 = Last_Refresh flag

Used in Resv and Path refresh messages to indicate that the sender has received a previously sent ACK and will not be sending further refreshes. When set, the ACK_Desired flag MUST also be set. This flag MUST NOT be set when the message has not been previously acknowledged.

Message ID: 24 bits

a 24-bit identifier. When combined with the message generator's IP address, uniquely identifies a message.

MESSAGE_ID ACK object

Class = MESSAGE_ID 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   ACK Flags   |                  Message ID                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

ACK Flags: 8 bits

0x08 = No_Refresh flag

Indicates that refreshes are not required and SHOULD NOT be sent for the associated message. The associated message is indicated by the Message ID.

Message ID: 24 bits

a 24-bit identifier. When combined with the message generator's IP address, uniquely identifies a message.

5.2.2. Ack Message Format

Ack messages carry one or more MESSAGE_ID ACK objects. They MUST NOT contain any MESSAGE_ID objects. Ack messages are sent hop-by-hop between 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 Path, PathTear, Resv, and RervErr messages this is taken from the RSVP_HOP Object. For PathErr and ResvErr messages this is taken from the message's source address. 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 ACK> ... ]
```

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

5.2.3. MESSAGE_ID Object Usage

The MESSAGE_ID object may be included in any RSVP message other than the Ack message. The MESSAGE_ID object is always generated and processed hop-by-hop. The IP address of the object generator is represented in a per RSVP message type specific fashion. For Path and PathTear messages the generator's IP address is contained in the RSVP_HOP. For Resv, ResvTear, PathErr, ResvErr, ResvConf and Aggregate messages the generator's IP address is the source address in the IP header.

The Message ID 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. When a node is sending a refresh message with a MESSAGE_ID object, it SHOULD use the same Message ID value that was used in the RSVP message that first advertised the state being refreshed. When a node is sending a message that represents new or changed state, the Message ID value MUST have a value that is not otherwise in use. A value is considered to be in use when it has been used in the most recent advertisement or refresh of any state using the associated IP address. Care must also be taken to avoid reuse of a previously used value during times of network loss. At such times, the use of new values may not be noticed by receivers. There is no requirement for Message ID values to be increasing or ordered.

The ACK_Desired flag is set when the MESSAGE_ID object generator is capable of accepting MESSAGE_ID ACK objects. Such information can be used to ensure reliable delivery of error and confirm messages and to support fast refreshes in the face of network loss. Nodes setting the ACK_Desired flag SHOULD retransmit unacknowledged messages at a faster interval than the standard refresh time until the message is acknowledged or a "fast" retry limit is reached.

Nodes receiving messages containing MESSAGE_ID objects SHOULD use the information in the objects to aid in determining if an message represents new state or a state refresh. Note that state is only refreshed in Path and Resv messages. If a Path or Resv message contains the same Message ID 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 ID value differs from the most recently received value, the receiver MUST fully processes the message.

Nodes receiving a message containing a MESSAGE_ID object with the ACK_Desired flag set, SHOULD respond with a MESSAGE_ID ACK object. If a node has ever responded with a MESSAGE_ID ACK object, it MUST

also check the Last_Refresh flag of received Resv and Path messages. If the flag is set, the receiver MUST NOT timeout state associated with associated message. The receiver MUST also be prepared to properly process refresh messages.

5.2.4. MESSAGE_ID ACK 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. The Message ID field of a MESSAGE_ID ACK object MUST have the same value as was received. 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.

A MESSAGE_ID ACK object may be sent in any RSVP message that has an IP destination address matching the generator of the associated MESSAGE_ID object. The MESSAGE_ID ACK object 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 MESSAGE_ID ACK objects SHOULD be sent in an Ack message. Implementations SHOULD include MESSAGE_ID ACK objects in standard RSVP messages when possible.

The No_Refresh flag is set to indicate that the receiver does not desire refreshes for the message being acknowledged. (Note that state is only refreshed in Path and Resv messages.) Receivers SHOULD set this flag when acknowledging receipt of Path or Resv messages and when the receiver has some mechanism to determine when the generator looses it's state, e.g. the mechanism described in Section 5.4. When a receiver sets this flag, the receiver MUST continue to timeout state associated with acknowledged message. The receiver may only stop timing out state after it receives a refresh message with the Last_Refresh flag set, see Section 5.2.3.

Upon receiving a MESSAGE_ID ACK object with the No_Refresh flag set, a refresh message with the Last_Refresh flag set SHOULD be generated. If a refresh message with the Last_Refresh flag set is generate, then normal refresh generation MUST continue until the message containing the Last_Refresh flag is acknowledged. Once an acknowledgment is received, normal refresh generation SHOULD be disabled for the associated state.

When normal refresh generation is suppressed for Path and Resv state, special care must be taken to remove such state. Particularly in the case of possible packet loss. To ensure such state is removed, once a node generates a Path or Resv refresh message containing a MESSAGE_ID object with the Last_Refresh flag set, the node MUST

retransmit until acknowledged all messages removing such state. Messages removing state include PathTear and ResvTear.

5.2.5. Multicast Considerations

Path and PathTear messages may be sent to IP multicast destination addresses. When the destination is multicast, 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, ACK implosion, number acknowledgments to be expected and handling 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 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 "fast" retransmission interval.

A more fundamental issue is the number of acknowledgments that the upstream node, 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 usually 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. When the number of next-hops is not known, the message generator SHOULD only expect a single response and MUST ignore the No_Refresh flag of MESSAGE_ID Ack objects. 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. Standard refresh messages will synchronize state with any next-hops that don't receive the original message.

Another issue is handling new (host or router) receivers. It is possible that after sending a Path message and handling of expected number of acknowledgments that a new receiver joins the group. In this case a new Path message must be sent to the new receiver. When normal refresh processing is occurring, there is no issue. When normal refresh processing is suppressed, a path message must still be generated. In the router-to-router case, the identification of new next-hops can usually be obtained from routing. When hosts are either the upstream node or the next-hops, the identification of new next-hops will typically not be possible. When identification of new

next-hops is not possible, the message generator SHOULD only expect a single response and MUST ignore the No_Refresh flag of MESSAGE_ID Ack objects. 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. Standard refresh messages will synchronize state with any next-hops that don't receive the original message.

There is one additional minor issue with multiple next-hops. The issue is handling a combination of standard-refresh and non-refresh next-hops. In the case some MESSAGE_ID Ack objects for the same message are received with the No_Refresh flag set and other objects are received with the No_Refresh flag clear. When this case occurs, refreshes MUST be generated per standard RSVP.

5.2.6. Compatibility

There are no backward compatibility issues raised by the MESSAGE_ID Class. The MESSAGE_ID Class has an assigned value whose form is 10bbbbbb. Per RSVP [1], classes with values of this form must be ignored and not forwarded by nodes not supporting the class. When the receiver of a MESSAGE_ID object does not support the class, the object will be silently ignored. The generator of the MESSAGE_ID object will not see any acknowledgments and therefore refresh messages per standard RSVP. Lastly, since the MESSAGE_ID ACK object can only be issued in response to the MESSAGE_ID object, there are no possible issues with this object or Ack messages.

5.3. Hello Extension

The RSVP Hello extension enables RSVP nodes to detect a loss of a neighboring node's state information. In standard RSVP, such detection occurs as a consequence of RSVP's soft state model. When refresh message generation is disabled via the previously discussed No_Refresh flag processing, some other mechanism is needed to address this failure case. In many configurations, it may be possible to leverage existing neighbor-to-neighbor failure detection mechanisms. One example mechanism is routing protocol peering state.

The extension described in this section supports cases where there is no other neighbor-to-neighbor failure detection mechanism available. The extension is specifically designed so that one side can use the mechanism while the other side does not. Neighbor RSVP state tracking may be initiated at any time. This includes when neighbors first learn about each other, or just when neighbors are sharing Resv or Path state. All implementations supporting the MESSAGE_ID ACK

object MUST also support the Hello Extension. Such implementations are not required to initiate Hello processing but they MUST be able to respond to Hello messages.

The Hello extension is composed of a Hello message, a Hello ACK message and a STATE_SET object. The Hello and Hello ACK messages have identical format and only differ in that a Hello ACK message is generate in response to a Hello message. Multiple STATE_SET objects may appear in a Hello or Hello ACK message. These objects are used to indicate what set of state is being refreshed.

For Path State, a set consists of all the state with the same PHOP object. For Reservations State, a consists of all the state for which the associated Path messages have the same PHOP. These PHOP values are the values used in the STATE_SET objects. Thus sending a STATE_SET object with a locally generated PHOP refreshes all Path State sent with that PHOP object. Sending a STATE_SET object with a received PHOP refreshes all Reservation State associated with Path messages sent by a neighbor node with that PHOP.

Hello processing between two neighbors supports independent selection of, typically configured, failure detection intervals. Each neighbor can autonomously issue HELLO messages. Each HELLO messages is answered by an acknowledgment. Hellos also contain enough information so that one neighbor can suppress issuing hello generation and still perform neighbor failure detection.

Neighbor state tracking is accomplished by collecting and storing a state "instance" value per State Set. If a change in value is seen, then the neighbor is presumed to have reset that portion of it's RSVP state. HELLO messages provide a mechanism for polling for and providing one or more RSVP state instance values. A poll request also includes the sender's instance value(s). This allows the receiver of a poll to optionally treat the poll as an implicit poll response. This optional handling is an optimization that can reduce the total number of polls and responses processed by a pair of neighbors. In all cases, when both sides support the optimization the result will be only one set of polls and responses per failure detection interval. Depending on selected intervals, the same benefit can occur even when only one neighbor supports the optimization.

5.3.1. Hello and Hello Ack Message Formats

Hello and Hello Ack Messages are always sent between two RSVP neighbors. The IP source address is the IP address of the sending node. The IP destination address is the IP address of the neighbor node.

The Hello and Hello Ack message formats are as follows:

```
    <Hello Message> ::= <Common Header> [ <INTEGRITY> ]
                        <STATE_SET List>

    <STATE_SET List>      ::= <STATE_SET> [ <STATE_SET List> ]
```

For Hello messages, the Msg Type field of the Common Header MUST be set to <TO_BE_ASSIGNED>.

For Hello Ack messages, the Msg Type field of the Common Header MUST be set to <TO_BE_ASSIGNED>.

5.3.2. STATE_SET Object

Class = TBD, C_Type = 1 (Class of form 0bbbbbbb)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   IPv4 Previous Hop Address                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Logical Interface Handle                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Instance                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Instance: 32 bits

a 32 bit value that represents the sender's RSVP agent's state. This value must change when the agent is reset or the node reboots and otherwise remain the same. This field MUST NOT be set to zero (0).

5.3.3. Hello Message Usage

A Hello message MUST be generated for each neighbor who's state is being tracked. When generating a Hello message, the sender fills in the Instance field with a value representing it's RSVP agent state for each state set. Each instance value MUST NOT change while the agent is maintaining any RSVP state. The generation of a message SHOULD be skipped when a Hello message is received from the destination node within the failure detection interval.

On receipt of a Hello message, the receiver MUST generate a Hello Ack message. The receiver SHOULD also verify that each of the neighbor's state set list has not changed. This is done by comparing the received state set list with the previously received state set list. If any state set values differ or are omitted, than each state set omitted or with a different instance value has reset and all state in that state set MUST be "expired" and cleaned up per standard RSVP processing.

On receipt of a Hello Ack message, the receiver MUST verify that the state set list has not changed. This is done by comparing the received state set list with the previously received state set list. If any state set values differ or are omitted, than each state set omitted or with a different instance value has reset and all state in that state set MUST be "expired" and cleaned up per standard RSVP processing.

5.3.4. Compatibility

The Hello extension is fully backwards compatible. The Hello class is assigned a class value of the form 0bbbbbbb. Depending on the implementation, implementations that don't support the extension will either silently discard Hello messages or will respond with an "Unknown Object Class" error. In either case the sender will fail to see an acknowledgment for the issued Hello. When a Hello sender does not receive an acknowledgment, it MUST NOT send MESSAGE_ID ACK objects with the No_Refresh flag set to the corresponding RSVP neighbor. This restriction will preclude neighbors from getting out of RSVP state synchronization.

6. Acknowledgments

This document contains ideas as well as text that have appeared in previous Internet Drafts. The authors of the current draft wish to thank the authors of those drafts. They are Steven Blake, Bruce Davie, Roch Guerin, Sanjay Kamat, Yakov Rekhter, Eric Rosen, and Arun Viswanathan. We also wish to than Yoram Bernet for his comments on this draft.

7. References

[1] Braden, R. et al. Resource ReSerVation Protocol (RSVP) -- Version 1, Functional Specification, RFC 2205, September 1997.

[2] Rosen, E. et al. A Proposed Architecture for MPLS, Internet Draft, draft-ietf-mpls-arch-02.txt, July 1998.

[3] Awduche, D. et al. Requirements for Traffic Engineering over MPLS, Internet Draft, draft-ietf-mpls-traffic-eng-00.txt, October 1998.

[4] Wroclawski, J. Specification of the Controlled-Load Network Element Service, RFC 2211, September 1997.

[5] Rosen, E. MPLS Label Stack Encoding. Internet Draft, draft-ietf-mpls-label-encaps-03.txt, September 1998.

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

8. Authors' Addresses

Daniel O. Awduche
UUNET Worldcom
3060 Williams Drive
Fairfax, VA 22031
Voice: +1 703 208 5277
Email: awduche@uu.net

Lou Berger
FORE Systems
1595 Spring Hill Road, Suite 500
Vienna, VA 22182
Voice: +1 703 245 4527
Email: lberger@fore.com

Der-Hwa Gan
Juniper Networks, Inc.
385 Ravendale Drive
Mountain View, CA 94043
Voice: +1 650 526
Email: dhg@juniper.net

Tony Li
Juniper Networks, Inc.
385 Ravendale Drive
Mountain View, CA 94043
Voice: +1 650 526 8006
Email: tli@juniper.net

Vijay Srinivasan
Torrent Networking Technologies Corp.
3000 Aerial Center Parkway, Suite 140
Morrisville, NC 27560
Voice: +1 919 468 8466 ext. 236
Email: vijay@torrentnet.com

George Swallow
Cisco Systems, Inc.
250 Apollo Drive
Chelmsford, MA 01824
Voice: +1 978 244 8143
Email: swallow@cisco.com