MPLS Working Group   Daniel O. Awduche
Internet Draft   UUNET (MCI Worldcom), Inc.
Expiration Date: March 2000

                                                       Lou Berger
                                                    LabN Consulting

                                                      Der-Hwa Gan
                                               Juniper Networks, Inc.

                                                          Tony Li
                                                Procket Networks, Inc.

                                                   George Swallow
                                                 Cisco Systems, Inc.

                                                  Vijay Srinivasan
                                                Torrent Networks, Inc.

                                                   September 1999

Extensions to RSVP for LSP Tunnels

draft-ietf-mpls-rsvp-lsp-tunnel-03.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.

Contents

1. Introduction and Background

1.1. Introduction

Section 2.9 of the MPLS architecture [2] defines a label distribution
protocol as a set of procedures by which one Label Switched Router
(LSR) informs another of the meaning of labels used to forward
traffic between and through them. The MPLS architecture does not
assume a single label distribution protocol. 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, and loop
detection.

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. A few new objects are also defined
that enhance management and diagnostics of LSP tunnels.

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.

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

[1.2](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.

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 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 local-protection, 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] and the Class-of-Service service, see
   Section 4.8.

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, all senders send simultaneously, then there is no means of
getting the proper reservations made. Either the reserved bandwidth
on links close to the destination will be less than what is required
or then the reserved bandwidth on links close to some senders will 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, different
labels may be assigned to different senders, thereby creating
separate LSPs.

SE style reservations can be provided using multipoint-to-point
label-switched-path or LSP per sender. Multipoint-to-point LSPs may
be used when path messages do not carry the EXPLICIT_ROUTE object, or
when Path messages have identical EXPLICIT_ROUTE objects. In either
of these cases a common label may be assigned.

Path messages from different senders can each carry their own ERO,
and the paths taken by the senders can converge and diverge at any
point in the network topology. When Path messages have differing
EXPLICIT_ROUTE objects, separate LSPs for each EXPLICIT_ROUTE object
must be established.

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 (an address of the node which is the
   egress of the tunnel), a Tunnel ID, and the tunnel ingress node's IP
   address, which is placed in the Extended Tunnel ID field.

   During the reroute operation, the tunnel ingress needs to appear as
   two different senders to the RSVP session. 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 ingress node picks a new LSP ID and forms a
new SENDER_TEMPLATE. The ingress 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 ingress
node receives a Resv message for the new LSP, it can transition
traffic to it and tear down the old LSP.

3. **LSP Tunnel related Message Formats**

   Five new objects are defined in this section:

       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,
   FILTER_SPEC, FLOWSPEC 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>

     <flow descriptor list> ::= <FF flow descriptor list>
                               | <SE flow descriptor>

     <FF flow descriptor list> ::= <FF flow descriptor>
                               | <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. LSP Tunnel related 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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   //                    (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 is not addressed in this 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 is expected 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.

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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Reserved           |             L3PID             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

      Reserved

         This field is reserved. It must be set to zero on transmis-
         sion 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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Reserved           |             L3PID             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 | Res  |    Minimum VPI         |       Minimum VCI             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 | Res  |    Maximum VPI         |       Maximum VCI             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
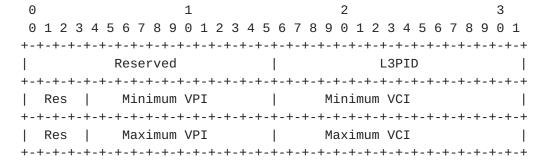
      Reserved (Res)

This field is reserved. It must be set to zero on transmis-
sion 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 ori-
ginating 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 ori-
ginating 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Reserved           |             L3PID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Reserved      |DLI|                 Minimum DLCI              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Reserved      |                   Maximum DLCI                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Reserved

This field is reserved. It must be set to zero on transmis-
sion 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 error
   value "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 error value
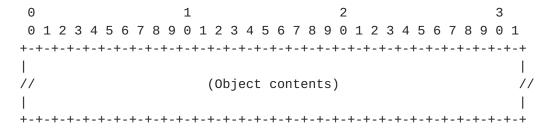   "Unsupported L3PID." This causes the RSVP session to fail.

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 senders and receivers. However, obviously, non-RSVP routers
   cannot convey labels via RSVP. This means that if a router has a
   neighbor that is known to not be RSVP capable, the router must not
   advertise the LABEL_REQUEST object 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 error
   value "MPLS being negotiated, but a non-RSVP capable router stands in
   the path." This same message should be sent, if a router receives a
   LABEL_REQUEST object in a message from a non-RSVP See [1] for a
   description of how a downstream router 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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   //                      (Object contents)                     //
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

      Class-Num

         The Class-Num for an EXPLICIT_ROUTE object is 20 (need to get
         an official one from the IANA with the form 0bbbbbbb to ensure
         that non-supporting implementations reject the message.)

C-Type

The C-Type for an EXPLICIT_ROUTE object is 1 (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
EXPLICIT_ROUTE object is assigned a class value of the form 0bbbbbbb.
RSVP routers that do not support the object will therefore response
with an "Unknown Object Class" error.

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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |L|    Type     |     Length    | IPv4 address (4 bytes)        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | IPv4 address (continued)      | Prefix Length |     Flags     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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

      0x01   IPv4 address

   Length

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

   IPv4 address

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

   Prefix length

      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](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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |L|    Type     |     Length    | IPv6 address (16 bytes)       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | IPv6 address (continued)                                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | IPv6 address (continued)                                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | IPv6 address (continued)                                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | IPv6 address (continued)      | Prefix Length |     Flags     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

    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

        0x02   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 prefix length value below. Bits beyond the prefix are
        ignored on receipt and should be set to zero on transmission.

     Prefix Length

         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 pops the subobject from the
   EXPLICIT ROUTE object , records the subobject, and continues
   processing with step 2, above. Note that this changes the third
   subobject into the second subobject (hence "pop") 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 and
   allows the node to identify the next abstract node on the path of the
   message after possible repeated application(s) of steps 2-4.

   5) Abstract Node Border Case: 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) Interior of the Abstract Node Case: Otherwise, the node selects a
   next hop within the abstract node of the first subobject (which the
   node belongs to) that is along the path to the abstract node of the
   second subobject (which is the next abstract node). 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
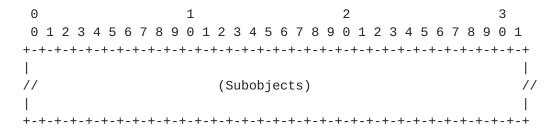subobject.

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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   //                        (Subobjects)                        //
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Class-Num

      The Class-Num for a RECORD_ROUTE object is 21 (need to get an
      official one from the IANA with the form 0bbbbbbb to ensure
      that non-supporting implementations reject the message.)

   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.
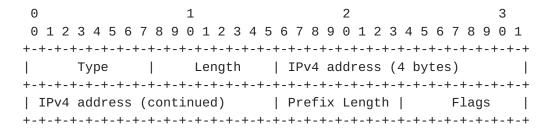Subsequent RROs can be ignored and should not be propagated.

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)      | Prefix Length |     Flags     |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Type

      0x01   IPv4 address

Length

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

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.

Prefix length

      32

Flags

     0x01 Local protection available

         Indicates that the link downstream of this node is
         protected via a local repair mechanism

     0x02 Local protection in use

         Indicates that a local repair mechanism is in use to
         maintain this tunnel (usually in the face a an outage
         of the link it was previously routed over

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)      | Prefix Length |     Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

0x02   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.

Prefix length

128

Flags

0x01   Local protection available

Indicates that the link downstream of this node is
protected via a local repair mechanism

0x02   Local protection in use

Indicates that a local repair mechanism is in use to
maintain this tunnel (usually in the face a an outage
of the link it was previously routed over)

[4.4.2](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) will
be 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 (or Resv) message, the RRO object shall be dropped from the
   message and message processing continues as normal. A PathErr (or
   ResvErr) message should be sent back to the sender (or receiver). An
   error code of "Notify" and an error value of "RRO too large for MTU"
   is used. The RRO object is included in the error message. If the
   receiver receives such a ResvErr, it should send a PathErr message
   with error code of "Notify" and an error value of "RRO notification".
   The RRO object is included in the error message.

   A sender receiving either of these error values should remove the RRO
   from the Path message.

   Nodes should resend the above PathErr or ResvErr message each n
   seconds where n is the greater of 15 and the refresh interval for the
   associated Path or RESV message. The node may apply limits and/or
   back-off timers to limit the number of messages sent.

   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 error value "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](4.4.5). Non-support of RRO

   The RRO object is to be used only when all routers along the path
   support RSVP and the RRO object. The RRO object is assigned a class
   value of the form 0bbbbbbb. RSVP routers that do not support the
   object will therefore response with an "Unknown Object Class" error.

[4.5](4.5). Error Codes for ERO and RRO

   In the processing described above, certain errors must be reported as
   either a "Routing Problem" or "Notify". The value of the "Routing
   Problem" error code is 24 (TBD); the value of the "Notify" error code
   is 25 (TBD).

   The following defines error values for the Routing Problem Error
   Code:

        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

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

        ss00 cccc cccc cccc

   The high order bits are as defined under Error Code 1. (See [1]).

   When ss = 00, the following subcode is defined:

        1   RRO too large for MTU
        2   RRO notification

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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 IPv4 tunnel end point address                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Must be zero                 |          Tunnel ID            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Extended Tunnel ID                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

     IPv4 tunnel end point address

        IPv4 address of the egress 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.
        Ingress nodes that wish to narrow the scope of a SESSION to the
        ingress-egress pair may place their IPv4 address here as a
        globally unique identifier.

[4.6.2](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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 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](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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 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 ingress 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
LSP_ID. Tunnel setup then proceeds according to the normal procedure.

On receipt of the Path message, the egress node sends a Resv message
with the STYLE Shared Explicit toward the ingress node.

When an ingress 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 ingress node picks a new LSP_ID to form a new
SENDER_TEMPLATE. It creates an EXPLICIT_ROUTE object for the new
route. The new Path message is sent. The ingress node refreshes
both the old and new path messages

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

        <FLOWSPEC><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 ingress 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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  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 = Local protection
         This flag permits transit routers to use a local repair
         mechanism which may result in violation of the explicit
         route object. When a fault is detected on an adjacent
         downstream link or node, a transit router can reroute
         traffic 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 = Ingress node may reroute
         This flag indicates that the tunnel ingress node may
         choose to reroute this tunnel without tearing it down.
         A tunnel egress node 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 local-protection is optional. A node may recognize
   the local-protection 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.

4.8. Tspec and Flowspec Object for Class-of-Service Service

   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 Class-of-Service service should be used.

   The Class-of-Service FLOWSPEC allows indication of a Class of Service
   (CoS) value that should be used when handling data packets associated
   with the request.

   The same format is used both for SENDER_TSPEC object and FLOWSPEC

objects.   The formats are:

       Class-of-Service SENDER_TSPEC object: Class = 12, C-Type = 3 (TBA)

       Class-of-Service FLOWSPEC object: Class = 9, C-Type = 3 (TBA)

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |      CoS      |    Maximum Packet Size [M]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

    Reserved

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

    CoS

       Indicates the Class of Service (CoS) of the data traffic
       associated with the request.   A value of zero (0) indicates
       that associated traffic is "Best-Effort".   Specifically no
       service assurances are being requested from the network. The
       intent is to enable networks to support the IP ToS Octet as
       defined in RFC1349 [7]. It is noted that there is ongoing work
       within the IETF to update the use of the IP ToS Octet.   In
       particular, RFC2474 [8], obsoletes RFC1349. However at this
       time the new uses of this field (now termed the DS byte) are
       still being defined. Non-zero values have local significance.
       The translation from a specific value to an allocation is a
       local administrative decision.

     M
       The maximum packet size parameter [M] should be set to the
       value of the smallest path MTU. This parameter is set in Resv
       messages by the reliever based on information in arriving RSVP
       ADSPEC objects.   This parameter is ignored when the object is
       contained in Path messages.

    There is no Adspec associated with the Class-of-Service SENDER_TSPEC.
    Either the Adspec is omitted or an int-serv adspec with only the
    Default General Characterization Parameters is used.

5. Security Considerations

   In principle these extensions to RSVP pose no security exposures over
   and above RFC 2205[1]. However, there is a slight change in the
   trust model. Traffic sent on a normal RSVP session can be filtered
   according to source and destination addresses as well as port
   numbers. In this specification, filtering occurs only on the basis
   of an incoming label. For this reason an administration may which to
   limit the domain over which LSP tunnels can be established. This can
   be accomplished by setting filters on various ports to deny action on
   a RSVP path message with a SESSION object of type LSP_TUNNEL_IPv4
   (7). In such a case an implementation MAY generate a Path_Err
   message with an error code of 02, "Policy Control failure".

6. Intellectual Property Considerations

   The IETF has been notified of intellectual property rights claimed in
   regard to some or all of the specification contained in this
   document. For more information consult the online list of claimed
   rights.

7. 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 thank Bora Akyol, Yoram Bernet and Alex
   Mondrus for their comments on this draft.

8. References

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

[2] Rosen, E. et al., "Multiprotocol Label Switching Architeture",
    Internet Draft, draft-ietf-mpls-arch-04.txt, February 1999.

[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.

[7] Almquist, P., "Type of Service in the Internet Protocol Suite",
    RFC 1349, July 1992.

[8] Nichols, K. et al., "Definition of the Differentiated Services Field
    (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

9. Authors' Addresses

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

Lou Berger
LabN Consulting
Voice: +1 301 468 9228
Email: lberger@labn.net

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

Tony Li
Procket Networks
3910 Freedom Circle, Ste. 102A
Santa Clara CA 95054
Email: tony1@home.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