Network Working Group   Daniel O. Awduche
Internet Draft   UUNET (MCI Worldcom), Inc.
Expiration Date: February 2001

                                                        Lou Berger
                                                LabN Consulting, LLC

                                                        Der-Hwa Gan
                                             Juniper Networks, Inc.

                                                           Tony Li
                                              Procket Networks, Inc.

                                                   Vijay Srinivasan
                                          Cosine Communications, Inc.

                                                   George Swallow
                                                Cisco Systems, Inc.

                                                        August 2000

RSVP-TE: Extensions to RSVP for LSP Tunnels

draft-ietf-mpls-rsvp-lsp-tunnel-07.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**

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.

The LSPs created with RSVP can be used to carry the "Traffic Trunks"
described in [3]. The LSP which carries a traffic trunk and a
traffic trunk are distinct though closely related concepts. For
example, two LSPs between the same source and destination could be
load shared to carry a single traffic trunk. Conversely several
traffic trunks could be carried in the same LSP if, for instance, the
LSP were capable of carrying several service classes. The
applicability of these extensions is discussed further in [10].

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, tunneling below normal IP routing and
filtering mechanisms. 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.

The document also describes a means of rapid node failure detection
via a new HELLO message.

All objects and messages 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.

1.1. 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. 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 simple 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.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in RFC2119 [6].

The reader is assumed to be familiar with the terminology in [1], [2]
and [3].

Abstract Node

A group of nodes whose internal topology is opaque to the ingress node of the LSP. An abstract node is said to be simple if it contains only one physical node.

Explicitly Routed LSP

An LSP whose path is established by a means other than normal IP routing.

Label Switched Path

The path created by the concatenation of one or more label switched hops, allowing a packet to be forwarded by swapping labels from an MPLS node to another MPLS node. For a more precise definition see [2].

LSP
A Label Switched Path

LSP Tunnel

An LSP which is used to tunnel below normal IP routing and/or filtering mechanisms.

Traffic Engineered Tunnel (TE Tunnel)

A set of one or more LSP Tunnels which carries a traffic trunk.

Traffic Trunk

A set of flows aggregated by their service class and then placed on an LSP or set of LSPs called a traffic engineered tunnel. For further discussion see [3].

2. **Overview**

2.1. **LSP Tunnels and Traffic Engineered 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.

   New RSVP SESSION, SENDER_TEMPLATE, and FILTER_SPEC objects, called
   LSP_TUNNEL_IPv4 and LSP_TUNNEL_IPv6 have been defined to support the
   LSP tunnel feature. The semantics of these objects, 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(v6) that appears in the object
   name only denotes that the destination address is an IPv4(v6)
   address. When we refer to these objects generically, we use the
   qualifier LSP_TUNNEL.

   In some applications it is useful to associate sets of LSP tunnels.
   This can be useful during reroute operations or to spread a traffic
   trunk over multiple paths. In the traffic engineering application
   such sets are called traffic engineered tunnels (TE tunnels). To
   enable the identification and association of such LSP tunnels, two
   identifiers are carried. A tunnel ID is part of the SESSION object.
   The SESSION object uniquely defines a traffic engineered tunnel. The
   SENDER_TEMPLATE and FILTER_SPEC objects carry an LSP ID. The
   SENDER_TEMPLATE (or FILTER_SPEC) object together with the SESSION
   object uniquely identifies an LSP tunnel

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 or LSP_TUNNEL_IPv6 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 IP 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, 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 setup and hold priorities, resource affinities
(see [3]), and local-protection, are also included in this object.

Routers along the path may use the setup and hold priorities along
with SENDER_TSPEC and any POLICY_DATA objects contained in Path
messages as input to policy control. For instance, in the traffic
engineering application, it is very useful to use the Path message as

a means of verifying that bandwidth exists at a particular priority
along an entire path before pre-empting any lower priority
reservations. If a Path message is allowed to progress when there
are insufficient resources, the there is a danger that lower priority
reservations downstream of this point will unnecessarily be pre-
empted in a futile attempt to service this request.

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
in addition to the received ERO.

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, following
the path state created by the Path message, in reverse order. Note
that if the path state was created by use of an ERO, then the Resv
message will follow the reverse path of the ERO.

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 is
   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](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 Traffic Engineered Tunnels

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

   A similar situation can arise when one wants to increase the
   bandwidth of a TE tunnel. The new reservation will be for the full
   amount needed, but the actual allocation needed is only the delta
   between the new and old bandwidth. If policy is being applied to
   PATH messages by intermediate nodes, then a PATH message requesting
   too much bandwidth will be rejected. In this situation simply
   increasing the bandwidth request without changing the
   SENDER_TEMPLATE, could result in a tunnel being torn down, depending
   upon local policy.

   The combination of the LSP_TUNNEL SESSION object and the SE
   reservation style naturally accommodates smooth transitions in
   bandwidth and routing. The idea is that the old and new LSP tunnels
   share resources along links which they have in common. The LSP_TUNNEL
   SESSION object is used to narrow the scope of the RSVP session to the
   particular TE tunnel in question. To uniquely identify a TE 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 or bandwidth-increase operation, the tunnel
ingress needs to appear as two different senders to the RSVP session.
This is achieved by the inclusion of the "LSP ID", which is carried
in the SENDER_TEMPLATE and FILTER_SPEC objects. Since the semantics
of these objects are changed, a new C-Types are 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.

To effect a bandwidth-increase, a new Path Message with a new LSP_ID
can be used to attempt a larger bandwidth reservation while the
current LSP_ID continues to be refreshed to ensure that the
reservation is not lost if the larger reservation fails.

2.6. Path MTU

Standard RSVP [1] and Int-Serv [11] provide the RSVP sender with the
minimum MTU available between the sender and the receiver. This path
MTU identification capability is also provided for LSPs established
via RSVP.

Path MTU information is carried, depending on which is present, in
the Integrated Services or Class-of-Service objects. When using
Integrated Services objects, path MTU is provided based on the
procedures defined in [11]. Path MTU identification when using
Class-of-Service objects is defined in Section 4.8.

With standard RSVP, the path MTU information is used by the sender to
check which IP packets exceed the path MTU. For packets that exceed
the path MTU, the sender either fragments the packets or, when the IP
datagram has the "Don't Fragment" bit set, issues an ICMP destination
unreachable message. This path MTU related handling is also required
for LSPs established via RSVP.

The following algorithm applies to all unlabeled IP datagrams and to
any labeled packets which the node knows to be IP datagrams, to which
labels need to be added before forwarding. For labeled packets the
bottom of stack is found, the IP header examined.

Using the terminology defined in [5], an LSR MUST execute the
following algorithm:

   1. Let N be the number of bytes in the label stack (i.e, 4 times
      the number of label stack entries) including labels to be added
      by this node.

   2. Let M be the smaller of the "Maximum Initially Labeled IP
      Datagram Size" or of (Path MTU - N).

   When the size of the datagram (without labels) exceeds the value
     of M,

    If the DF bit is not set in the IP header, then

      (a) the datagram MUST be broken into fragments, each of whose
          size is no greater than M, and

      (b) each fragment MUST be labeled and then forwarded.

    If the DF bit is set in the IP header, then

      (a) the datagram MUST NOT be forwarded

      (b) Create an ICMP Destination Unreachable Message:
            i. set its Code field [12] to "Fragmentation Required
               and DF Set",
           ii. set its Next-Hop MTU field [13] to M

      (c) If possible, transmit the ICMP Destination Unreachable
          Message to the source of the of the discarded datagram.

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. In Resv messages they MUST appear after the associated FILTER_SPEC and prior to any subsequent FILTER_SPEC.

The relative 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> ::= <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. 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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                           (top label)                         |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The contents of a LABEL is a single label, encoded in 4 octets. Each
generic MPLS label is an unsigned integer in the range 0 through
1048575. Generic MPLS labels and FR labels are encoded right aligned
in 4 octets. ATM labels are encoded with the VPI right justified in
bits 0-15 and the VCI right justified in bits 16-31.

[4.1.1](4.1.1). Handling Label Objects in Resv messages

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.

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

[4.1.1.1](4.1.1.1). Downstream

The downstream node selects a label to represent the flow. If a
label range has been specified in the label request, the label MUST
be drawn from that range. If no label is available the node sends a
PathErr message with an error code of "routing problem" and an error
value of "label allocation failure".

If a node receives a Resv message that has assigned the same label
value to multiple senders, then that node MAY also assign a single
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.

In the case of ATM, one further condition applies. Some ATM nodes
are not capable of merging streams. These nodes MAY indicate this by
setting a bit in the label request to zero. The M-bit in the
LABEL_REQUEST object of C-Type 2, label request with ATM label range,
serves this purpose. The M-bit SHOULD be set by nodes which are
merge capable. If for any senders the M-bit is not set, the
downstream node MUST assign unique labels to those senders.

Once a label is allocated, the node formats a new LABEL object. The
node 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 node is expected to send a Resv message before its refresh timers
expire if the contents of the LABEL object change.

4.1.1.2. Upstream

A node uses the 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.

Several circumstance can lead to an unacceptable label.

   1. the node is a merge incapable ATM switch but the downstream node
      has assigned the same label to two senders

   2. The implicit null label was assigned, but the node is not
      capable of doing a penultimate pop for the associated L3PID

   3. The assigned label is outside the requested label range

In any of these events the node send a ResvErr message with an error
code of "routing problem" and an error value of "unacceptable label
value".

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 Class is 19.   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. The LABEL_REQUEST object formats are
shown below.

4.2.1. Label Request without Label Range

Class = 19, 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            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.

4.2.2. Label Request with ATM Label Range

Class = 19, 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Reserved           |             L3PID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M| 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.

M

Setting this bit to one indicates that the node is capable
of merging in the data plane

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 MUST be right
justified in this field and preceding bits MUST 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 MUST be
right justified in this field and preceding bits MUST 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 MUST be right
justified in this field and preceding bits MUST 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 MUST be
right justified in this field and preceding bits MUST be set to
zero.

4.2.3. Label Request with Frame Relay Label Range

```
     Class = 19, C_Type = 3

    0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           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
         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 MUST be right justified in this
   field and unused bits MUST 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 MUST be right justified in this
   field and unused bits MUST be set to 0.

4.2.4. **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.

   A node which receives and forwards a Path message each with a
   LABEL_REQUEST object, MUST copy the L3PID from the received
   LABEL_REQUEST object to the forwarded LABEL_REQUEST object.

   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.

4.2.5. **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 sends 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 capable router. See
   [1] for a description of how a downstream router can determine the
   presence of non-RSVP routers.

4.3. Explicit Route Object

   Explicit routes are specified via the EXPLICIT_ROUTE object (ERO).
   The Explicit Route Class is 20. Currently one C_Type is defined,
   Type 1 Explicit Route. The EXPLICIT_ROUTE object has the following
   format:

      Class = 20, 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
       +-+
       | |
       //   (Subobjects)   //
       | |
       +-+

       Subobjects

          The contents of an EXPLICIT_ROUTE object are a series of
          variable-length data items called subobjects. The subobjects
          are defined in section 4.3.3 below.

   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 respond
   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. 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 identifies a group of nodes
   in the explicit route. An explicit route is thus a specification of
   groups of nodes to be traversed.

   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. 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](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

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](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.

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 |     Resvd     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   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. **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 |    Resvd     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.

4.3.4. 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 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 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 and 3.

4) 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.

5) 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:

5a) If the second subobject is a strict subobject, there is an error
and the node SHOULD return a "Bad strict node" error.

5b) 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.

6) 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. **Forward Compatibility**

   It is anticipated that new subobjects may be defined over time. A
   node which encounters an unrecognized subobject during its normal ERO
   processing sends a PathErr with the error code "Routing Error" and
   error value of "Bad Explicit Route Object" toward the sender. The
   EXPLICIT_ROUTE object is included, truncated (on the left) to the
   offending subobject. The presence of an unrecognized subobject which
   is not encountered in a node's ERO processing SHOULD be ignored. It
   is passed forward along with the rest of the remaining ERO stack.

4.3.7. **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

   Routes can be recorded via   the   RECORD_ROUTE   object   (RRO).
   Optionally, labels may also be recorded. The Record Route Class is
   21. Currently one C_Type is defined, Type 1 Record Route.   The
   RECORD_ROUTE object has the following format:

      Class = 21, 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
      +-+
      | |
      //   (Subobjects)   //
      | |
      +-+

      Subobjects

         The contents of a RECORD_ROUTE object are a series   of
         variable-length data items called subobjects. The subobjects
         are defined in section 4.4.1 below.

   The RRO can be present in both RSVP Path and Resv messages. If a
   Path message contains multiple RROs, only the first RRO is
   meaningful. Subsequent RROs SHOULD be ignored and SHOULD NOT be
   propagated. Similarly, if in a Resv message multiple RROs are
   encountered following a FILTER_SPEC before another FILTER_SPEC is
   encountered, only the first RRO is meaningful. Subsequent RROs
   SHOULD 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.

Three 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 certain 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. This flag can
         only be set if the Local protection flag was set in the
         SESSION_ATTRIBUTE object of the corresponding Path
         message.

   0x02 Local protection in use

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

[4.4.1.2](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. This flag can
            only be set if the Local protection flag was set in the
            SESSION_ATTRIBUTE object of the coooresponding Path
            nessage.

      0x02 Local protection in use

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

[4.4.1.3](4.4.1.3). Subobject 0x03, Label

```
    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     |    Flags      |   C-Type      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Contents of Label Object                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Type

      0x03 Label

   Length

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

   Flags

      0x01 = Global label
        This flag indicates that the label will be understood
        if received on any interface.

C-Type

> The C-Type of the included Label Object. Copied from the
> Label Object.

Contents of Label Object

> The contents of the Label Object. Copied from the Label
> Object

[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](4.4.3). Processing 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. If the node also desires label recording, it
sets the Label_Recording flag in the SESSION_ATTRIBUTE object.

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.

When the Label_Recording flag is set in the SESSION_ATTRIBUTE object,
nodes doing route recording SHOULD include a Label Record subobject.
If the node is using a global label space, then it SHOULD set the
Global Label flag.

The Label Record subobject is pushed onto the RECORD_ROUTE object
prior to pushing on the node's IP address. A node MUST NOT push on a
Label Record subobject without also pushing on an IPv4 or IPv6
subobject.

Note that on receipt of the initial Path message, a node is unlikely
to have a label to include. Once a label is obtained, the node
SHOULD include the label in the RRO in the next Path refresh event.

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. 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".

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 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. **Forward Compatibility**

New subobjects may be defined for the RRO. When processing an RRO,
unrecognized subobjects SHOULD be ignored and passed on. When
processing an RRO for loop detection, a node SHOULD parse over any
unrecognized objects. Loop detection works by detecting subobjects
which were inserted by the node itself on an earlier pass of the

object. This ensures that the subobjects necessary for loop detection are always understood.

4.4.6. 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 respond with an "Unknown Object Class" error.

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; the value of the "Notify" error code is 25.

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      Unacceptable label value

        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 subcodes are defined:

        1   RRO too large for MTU
        2   RRO notification

[4.6](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 objects have the following format:

[4.6.1](4.6.1). Session Object

[4.6.1.1](4.6.1.1). LSP_TUNNEL_IPv4 Session Object

    Class = SESSION, LSP_TUNNEL_IPv4 C-Type = 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.

[4.6.1.2](4.6.1.2). **LSP_TUNNEL_IPv6 Session Object**

   Class = SESSION, LSP_TUNNEL_IPv6 C_Type = 8

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                  IPv6 tunnel end point address                |
+                                                               +
|                          (16 bytes)                           |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   MUST be zero                |           Tunnel ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                       Extended Tunnel ID                      |
+                                                               +
|                          (16 bytes)                           |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IPv6 tunnel end point address

   IPv6 address of the egress node for the tunnel.

Tunnel ID

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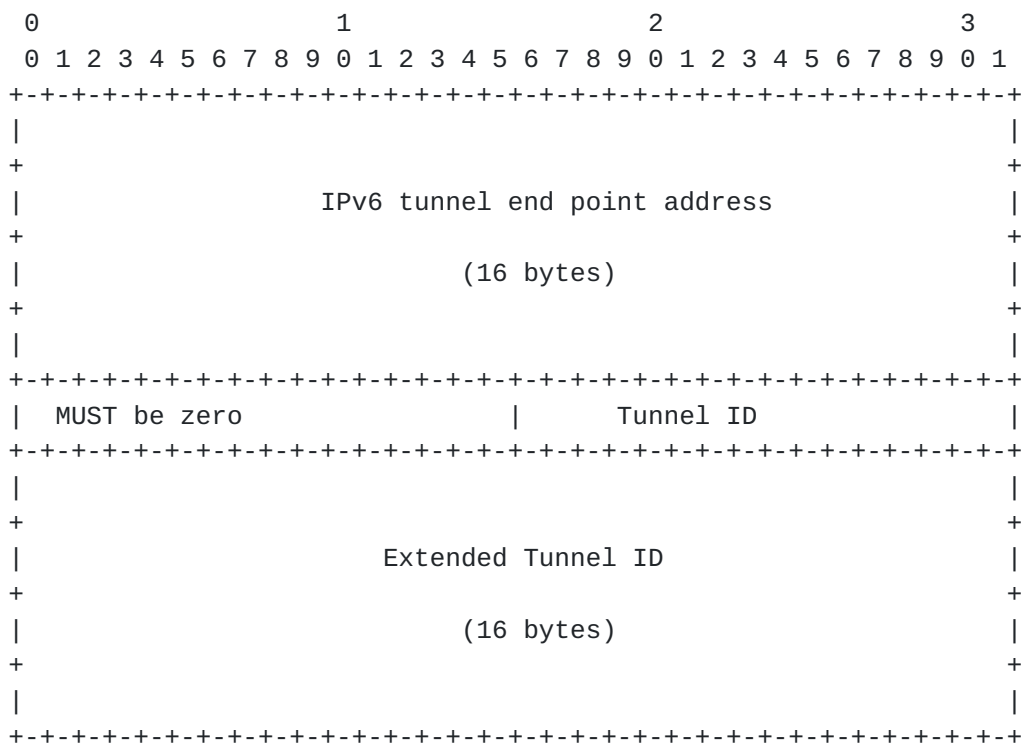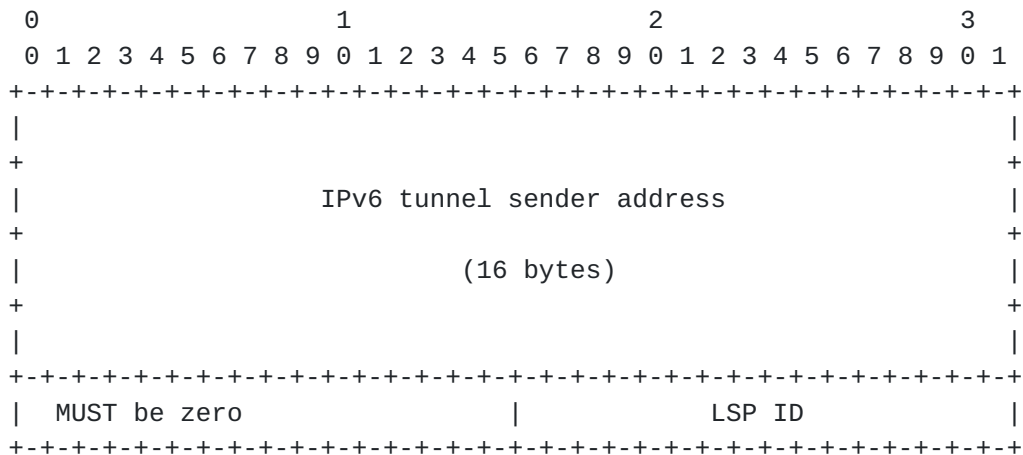   over the life of the tunnel.

Extended Tunnel ID

A 16-byte 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 IPv6 address here as a
globally unique identifier.

4.6.2. Sender Template Object

4.6.2.1. LSP_TUNNEL_IPv4 Sender Template Object

Class = SENDER_TEMPLATE, LSP_TUNNEL_IPv4 C-Type = 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.

4.6.2.2. **LSP_TUNNEL_IPv6 Sender Template Object**

     Class = SENDER_TEMPLATE, LSP_TUNNEL_IPv6 C_Type = 8

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                   IPv6 tunnel sender address                  |
+                                                               +
|                          (16 bytes)                           |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   MUST be zero                 |           LSP ID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

     IPv6 tunnel sender address

        IPv6 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**

4.6.3.1. **LSP_TUNNEL_IPv4 Filter Specification Object**

     Class = FILTER SPECIFICATION, LSP_TUNNEL_IPv4 C-Type = 7

  The format of the LSP_TUNNEL_IPv4 FILTER_SPEC object is identical to
  the LSP_TUNNEL_IPv4 SENDER_TEMPLATE object.

4.6.3.2. **LSP_TUNNEL_IPv6 Filter Specification Object**

     Class = FILTER SPECIFICATION, LSP_TUNNEL_IPv6 C_Type = 8

  The format of the LSP_TUNNEL_IPv6 FILTER_SPEC object is identical to
  the LSP_TUNNEL_IPv6 SENDER_TEMPLATE object.

4.6.4. **Reroute and Bandwidth Increase 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 Session Attribute Class is 207.   Two C_Types are defined,
LSP_TUNNEL,   C-Type = 7 and LSP_TUNNEL_RA, C-Type = 1.   The
LSP_TUNNEL_RA C-Type includes all the same fields as the LSP_TUNNEL
C-Type.   Additionally it carries resource affinity information. The
formats are as follows:

4.7.1. Format without resource affinities

```
    SESSION_ATTRIBUTE class = 207, LSP_TUNNEL C-Type = 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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |   Setup Prio  | Holding Prio  |      Flags    |  Name Length  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    //          Session Name      (NULL padded display string)    //
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.

Flags

  0x01 Local protection desired

        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 Label recording desired

        This flag indicates that label information should be
        included when doing a route record.

  0x04 SE Style desired

        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.

Name Length

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

Session Name

   A null padded string of characters.

[4.7.2](4.7.2). **Format with resource affinities**

   SESSION_ATTRIBUTE class = 207, LSP_TUNNEL_RA 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Exclude-any                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Include-any                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Include-all                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Setup Prio  | Holding Prio  |     Flags     |  Name Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//          Session Name      (NULL padded display string)      //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Exclude-any

      A 32-bit vector representing a set of attribute filters
      associated   with   a tunnel any of which renders a link
      unacceptable.

   Include-any

      A 32-bit vector representing a set of attribute filters
      associated with a tunnel any of which renders a link acceptable
      (with respect to this test). A null set (all bits set to zero)
      automatically passes.

   Include-all

      A 32-bit vector representing a set of attribute filters
      associated   with   a tunnel all of which must be present for a
      link to be acceptable (with respect to this test). A null set
      (all bits set to zero) automatically passes.

   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.

Flags

   0x01 Local protection desired

         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 Label recording desired

         This flag indicates that label information should be
         included when doing a route record.

   0x04 SE Style desired

         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.

Name Length

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

Session Name

   A null padded string of characters.

4.7.3. Procedures applying to both C-Types

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.

The setup and holding priorities are directly analogous to the
preemption and defending priorities as defined in [9]. While the
interaction of these two objects is ultimately a matter of policy,
the following default interaction is RECOMMENDED.

When both objects are present, the preemption priority policy element
is used. A mapping between the priority spaces is defined as
follows. A session attribute priority S is mapped to a preemption
priority P by the formula $P = 2^{(14-2S)}$. The reverse mapping is
shown in the following table.

```
        Preemption Priority     Session Attribute Priority

              0 - 3                         7
              4 - 15                        6
             16 - 63                        5
             64 - 255                       4
            256 - 1023                      3
           1024 - 4095                      2
           4096 - 16383                     1
          16384 - 65535                     0
```

When a new Path message is considered for admission, the bandwidth
requested is compared with the bandwidth available at the priority
specified in 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 recording of the Label subobject in the ROUTE_RECORD object is
controlled by the label-recording-desired flag in the
SESSION_ATTRIBUTE object. Since the Label subobject is not needed
for all applications, it is not automatically recorded. The flag
allows applications to request this only when needed.

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.

4.7.4. Resource Affinity Procedures

Resource classes and resource class affinities are described in [3].
In this document we use the briefer term resource affinities for the
latter term. Resource classes can be associated with links and
advertised in routing protocols. Resource class affinities are used
by RSVP in two ways. In order to be validated a link MUST pass the
three tests below. If the test fails a PathErr with the code "policy
control failure" SHOULD be sent.

When a new reservation is considered for admission over a strict node
in an ERO, a node MAY validate the resource affinities with the
resource classes of that link. When a node is choosing links in
order to extend a loose node of an ERO, the node MUST validate the
resource classes of those links against the resource affinities. If
no acceptable links can be found to extend the ERO, the node SHOULD
send a PathErr message with an error code of "Routing Problem" and an
error value of "no route available toward destination".

In order to be validated a link MUST pass the following three tests.

To precisely describe the tests use the definitions in the object
description above. We also define

    Link-attr   A 32-bit vector representing attributes associated
                  with a link.

The three tests are

  1. Exclude-any

      This test excludes a link from consideration if the link carries any
      of the attributes in the set.

      (link-attr & exclude-any) == 0

  2. Include-any

      This test accepts a link if the link carries any of the
      attributes in the set.

      (include-any == 0) | ((link-attr & include-any) != 0)

  3. Include-all

      This test accepts a link only if the link carries all of the
      attributes in the set.

      (include-all == 0) | (((link-attr & include-all) ^ include-
      all) == 0)

For a link to be acceptable, all three tests MUST pass. If the test
fails a error message

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.

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](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

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   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
      This parameter is set in Resv messages by the receiver based on
      information in arriving RSVP SENDER_TSPEC objects. For shared
      reservations, the smallest value received across all associated
      senders is used.   When the object is contained in Path
      messages, this parameter is updated at each hop with the lesser
      of the received value and the MTU of the outgoing interface.

   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. Hello Extension

   The RSVP Hello extension enables RSVP nodes to detect when a
   neighboring node is not reachable. The mechanism provides node to
   node failure detection. When such a failure is detected it is
   handled much the same as a link layer communication failure. This
   mechanism is intended to be used when notification of link layer
   failures is not available and unnumbered links are not used, or when
   the failure detection mechanisms provided by the link layer are not
   sufficient for timely node failure detection.

   It should be noted that node failure detection is not the same as a
   link failure detection mechanism, particularly in the case of
   multiple parallel unnumbered links.

   The Hello extension is specifically designed so that one side can use
   the mechanism while the other side does not. Neighbor failure
   detection 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.

   The Hello extension is composed of a Hello message, a HELLO REQUEST
   object and a HELLO ACK object. Hello processing between two
   neighbors supports independent selection of, typically configured,
   failure detection intervals. Each neighbor can autonomously issue
   HELLO REQUEST objects. Each request is answered by an
   acknowledgment. Hello Messages also contain enough information so
   that one neighbor can suppress issuing hello requests and still
   perform neighbor failure detection. A Hello message may be included
   as a sub-message within a bundle message.

   Neighbor failure detection is accomplished by collecting and storing
   a neighbor's "instance" value. If a change in value is seen or if
   the neighbor is not properly reporting the locally advertised value,
   then the neighbor is presumed to have reset. When a neighbor's value
   is seen to change or when communication is lost with a neighbor, then
   the instance value advertised to that neighbor is also changed. The
   HELLO objects provide a mechanism for polling for and providing an
   instance value. A poll request also includes the sender's instance
   value. 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.

5.1. **Hello Message Format**

   Hello 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 mechanism is intended for use between immediate neighbors.
   When HELLO messages are being the exchanged between immediate
   neighbors, the IP TTL field of all outgoing HELLO messages SHOULD be
   set to 1.

   The Hello message has a Msg Type of 20. The Hello message format is
   as follows:

      <Hello Message> ::= <Common Header> [<INTEGRITY>]
                              <HELLO>

5.2. **HELLO Object formats**

   The HELLO Class is 22. There are two C_Types defined.

5.2.1. **HELLO REQUEST object**

      Class = HELLO 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
       +-+
       | Src_Instance |
       +-+
       | Dst_Instance |
       +-+

5.2.2. **HELLO ACK object**

     Class = HELLO 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Src_Instance                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Dst_Instance                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

     Src_Instance: 32 bits

     a 32 bit value that represents the sender's instance. The
     advertiser maintains a per neighbor representation/value. This
     value MUST change when the sender is reset, when the node
     reboots, or when communication is lost to the neighboring node
     and otherwise remains the same. This field MUST NOT be set to
     zero (0).

     Dst_Instance: 32 bits

     The most recently received Src_Instance value received from the
     neighbor. This field MUST be set to zero (0) when no value has
     ever been seen from the neighbor.

5.3. **Hello Message Usage**

   The Hello Message is completely OPTIONAL. All messages may be
   ignored by nodes which do not wish to participate in Hello message
   processing. The balance of this section is written assuming that the
   receiver as well as the sender is participating. In particular, the
   use of MUST and SHOULD with respect to the receiver applies only to a
   node that supports Hello message processing.

   A node periodically generates a Hello message containing a HELLO
   REQUEST object for each neighbor who's status is being tracked. The
   periodicity is governed by the hello_interval. This value MAY be
   configured on a per neighbor basis. The default value is 5 ms.

   When generating a message containing a HELLO REQUEST object, the
   sender fills in the Src_Instance field with a value representing it's
   per neighbor instance. This value MUST NOT change while the agent is
   exchanging Hellos with the corresponding neighbor. The sender also
   fills in the Dst_Instance field with the Src_Instance value most

recently received from the neighbor. For reference, call this
variable Neighbor_Src_Instance. If no value has ever been received
from the neighbor or this node considers communication to the
neighbor to have been lost, the Neighbor_Src_Instance is set to zero
(0). The generation of a message SHOULD be suppressed when a HELLO
REQUEST object was received from the destination node within the
prior hello_interval interval.

On receipt of a message containing a HELLO REQUEST object, the
receiver MUST generate a Hello message containing a HELLO ACK object.
The receiver SHOULD also verify that the neighbor has not reset.
This is done by comparing the sender's Src_Instance field value with
the previously received value. If the Neighbor_Src_Instance value is
zero, and the Src_Instance field is non-zero, the
Neighbor_Src_Instance is updated with the new value. If the value
differs or the Src_Instance field is zero, then the node MUST treat
the neighbor as if communication has been lost.

The receiver of a HELLO REQUEST object SHOULD also verify that the
neighbor is reflecting back the receiver's Instance value. This is
done by comparing the received Dst_Instance field with the
Src_Instance field value most recently transmitted to that neighbor.
If the neighbor continues to advertise a wrong non-zero value after a
configured number of intervals, then the node MUST treat the neighbor
as if communication has been lost.

On receipt of a message containing a HELLO ACK object, the receiver
MUST verify that the neighbor has not reset. This is done by
comparing the sender's Src_Instance field value with the previously
received value. If the Neighbor_Src_Instance value is zero, and the
Src_Instance field is non-zero, the Neighbor_Src_Instance is updated
with the new value. If the value differs or the Src_Instance field
is zero, then the node MUST treat the neighbor as if communication
has been lost.

The receiver of a HELLO ACK object MUST also verify that the neighbor
is reflecting back the receiver's Instance value. If the neighbor
advertises a wrong value in the Dst_Instance field, then a node MUST
treat the neighbor as if communication has been lost.

If no Instance values are received, via either REQUEST or ACK
objects, from a neighbor within a configured number of
hello_intervals, then a node MUST presume that it cannot communicate
with the neighbor. The default for this number is 3.5.

When communication is lost or presumed to be lost as described above,
a node MAY re-initiate HELLOs. If a node does re-initiate it MUST
use a Src_Instance value different than the one advertised in the

previous HELLO message. This new value MUST continue to be
advertised to the corresponding neighbor until a reset or reboot
occurs, or until another communication failure is detected. If a new
instance value has not been received from the neighbor, then the node
MUST advertise zero in the Dst_instance value field.

[5.4](5.4). Multi-Link Considerations

As previously noted, the Hello extension is targeted at detecting
node failures not per link failures. When there is only one link
between neighboring nodes or when all links between a pair of nodes
fail, the distinction between node and link failures is not really
meaningful and handling of such failures has already been covered.
When there are multiple links shared between neighbors, there are
special considerations. When the links between neighbors are
numbered, then Hellos MUST be run on each link and the previously
described mechanisms apply.

When the links are unnumbered, link failure detection MUST be
provided by some means other than Hellos. Each node SHOULD use a
single Hello exchange with the neighbor. The case where all links
have failed, is the same as the no received value case mentioned in
the previous section.

[5.5](5.5). Compatibility

The Hello extension does not affect the processing of any other RSVP
message. The only effect is to allow a link (node) down event to be
declared sooner than it would have been. RSVP response to that
condition is unchanged.

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 do not 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.

6. 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 wish 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)
   or LSP_TUNNEL_IPv6 (8).

7. IANA Considerations

   The responsible Internet authority (presently called the IANA)
   assigns values to RSVP protocol parameters. With the current
   document an EXPLICIT_ROUTE object and a ROUTE_RECORD object are
   defined. Each of these objects contain subobjects. This section
   defines the rules for the assignment of subobject numbers. This
   section uses the terminology of BCP 26 "Guidelines for Writing an
   IANA Considerations Section in RFCs".

   EXPLICIT_ROUTE Subobject Type

      EXPLICIT_ROUTE Subobject Type is a 7-bit number that identifies
      the function of the subobject. There are no range restrictions.
      All possible values except zero are available for assignment.

   ROUTE_RECORD Subobject Type

      ROUTE_RECORD Subobject Type is an 8-bit number that identifies the
      function of the subobject. There are no range restrictions. All
      possible values except zero are available for assignment.

8. 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.

9. 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.

10. 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-06.txt, August 1999.

[3]   Awduche, D. et al., "Requirements for Traffic Engineering over
      MPLS", RFC 2702, September 1999.

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

[5]   Rosen, E. et al., "MPLS Label Stack Encoding", Internet Draft,
      draft-ietf-mpls-label-encaps-07.txt, September 1999.

[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]   Herzog, S., "Signaled Preemption Priority Policy Element",
      RFC 2751, January 2000.

[10] Awduche, D. et al., "Applicability Statement for Extensions to RSVP
      for LSP-Tunnels", draft-ietf-mpls-rsvp-tunnel-applicability-00.txt,
      September 1999.

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

[12] Postel, J., "Internet Control Message Protocol", RFC 792,
     September 1981.

[13] Mogul, J. & Deering, S., "Path MTU Discovery", RFC 1191,
     November 1990.

11. 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, LLC
   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
   Cosine Communications, Inc.
   1200 Bridge Parkway
   Redwood City, CA 94065
   Voice: +1 650 628 4892
   Email: vsriniva@cosinecom.com

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